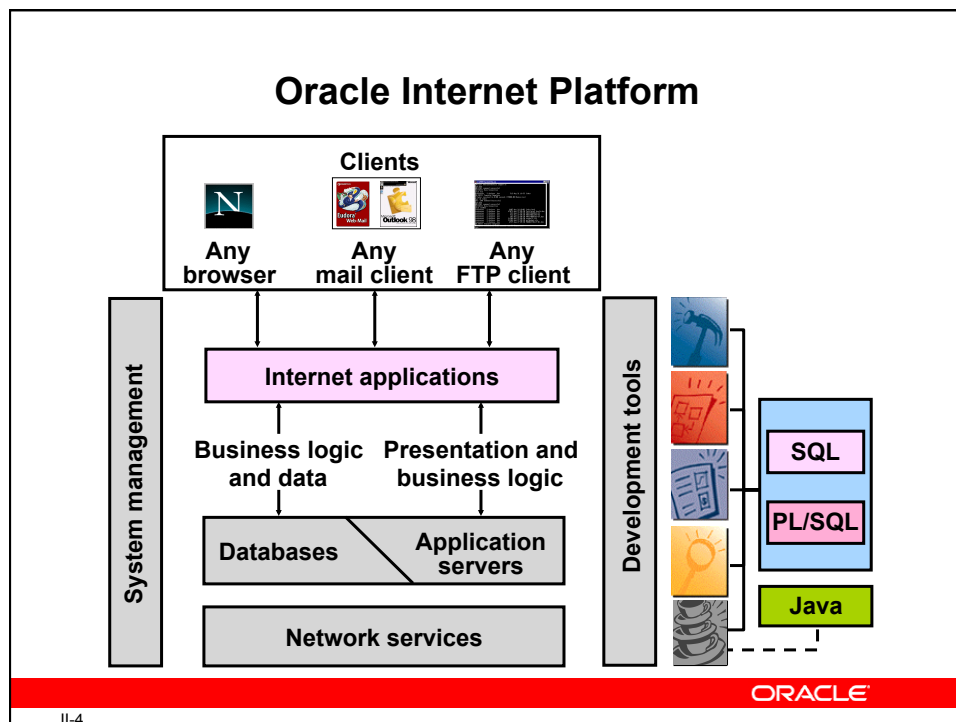


ORACLE

## Objectifs

- Comprendre ce que PL/SQL fournit comme extensions de programmation à SQL
- Écrire du code PL/SQL en interface avec la base de données
- Concevoir des blocks PL/SQL qui s'exécutent efficacement
- Utiliser des constructions PL/SQL pour le traitement conditionnel et de boucle
- Gérer les erreurs d'exécution
- Décrire les fonctions et procédures stockées

ORACLE



## Généralités

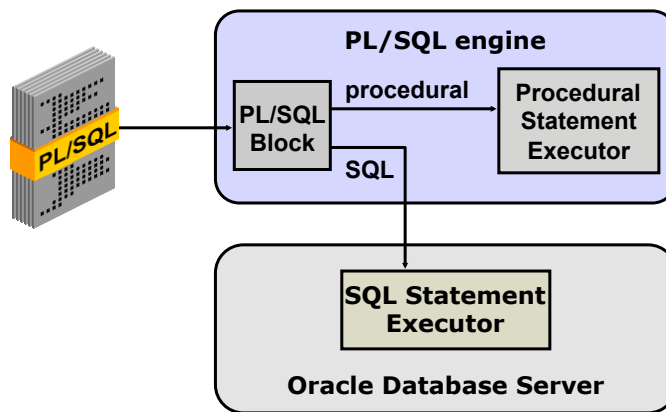
**PL/SQL : Langage procédural**

- Extension de SQL
- Déclaration de variables et de constantes
- Définition de sous-programmes
- Gestion des erreurs à l'exécution (exceptions)
- Manipulation de données avec SQL

**ORACLE**

II-6

## PL/SQL Environment

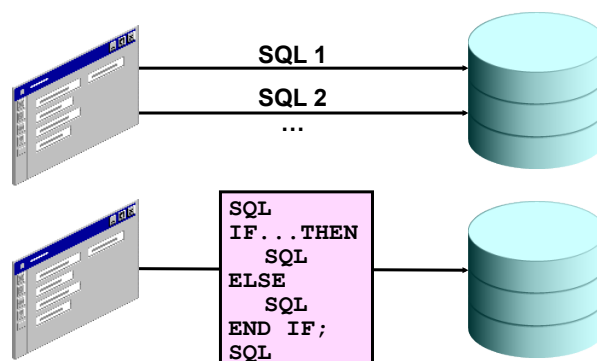


II-7

ORACLE

## Benefits of PL/SQL

- Integration of procedural constructs with SQL
- Improved performance



II-8

ORACLE

## PL/SQL Block Structure

- **DECLARE (optional)**
  - Variables, cursors, user-defined exceptions
- **BEGIN (mandatory)**
  - SQL statements
  - PL/SQL statements
- **EXCEPTION (optional)**
  - Actions to perform when errors occur
- **END; (mandatory)**



ORACLE

II-9

## Block Types

### Anonymous

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

### Procedure

```
PROCEDURE name
IS
BEGIN
  --statements

[EXCEPTION]

END;
```

### Function

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;

[EXCEPTION]

END;
```

ORACLE

II-11

# LES VARIABLES PL/SQL

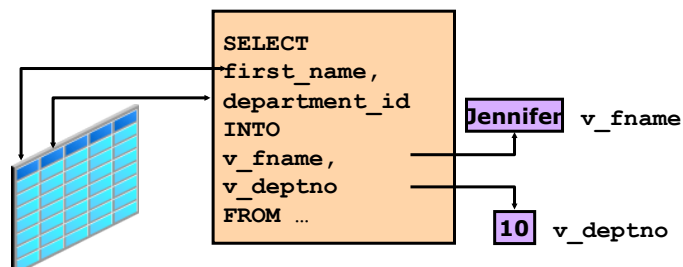
ORACLE

II-13

## Les Variables

Les Variables peuvent être utilisées pour :

- le stockage temporaire des données
- la manipulation des valeurs stockées
- la réutilisabilité



ORACLE

II-14

## Les variables

PL/SQL gère deux types de variables

- Les variables PL/SQL
  - Scalar
  - Composite
  - Reference
  - Large object (LOB)
- Les variables non PL/SQL
  - Les variables champs écrans FORMS
  - Les variables de lien (“ bind ” variables -variables SQL).
  - Les variables du langage hôte dans les langages PRO.
  - Elles sont toujours préfixées de ':' lors de leur utilisation.
  - Les variables PL/SQL déclarées dans les packages sont toujours préfixées du nom du package lors de leur utilisation.

15  
ORACLE

II-15

## Declarer et Initialiser des Variables PL/SQL

```
identifieur [CONSTANT] datatype [NOT NULL]  
[:= | DEFAULT expr];
```

### Exemples:

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

ORACLE

II-16

## Declarer et Initialiser des Variables PL/SQL

1

```
DECLARE
  v_myName VARCHAR2(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName);
  v_myName := 'John';
  DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName);
END;
/
```

2

```
DECLARE
  v_myName VARCHAR2(20) := 'John';
BEGIN
  v_myName := 'Steven';
  DBMS_OUTPUT.PUT_LINE('My name is: ' || v_myName);
END;
/
```

ORACLE

II-17

## Comment déclarer et initialiser des Variables PL/SQL

- Suivre les conventions d'affectation de noms.
- Utiliser des identificateurs significatifs pour les variables.
- Initialiser des variables désignés comme non NULL et constante.
- Initialiser des variables avec l'opérateur d'assignation (: =) ou le mot clé DEFAULT :

```
v_myName VARCHAR2(20) := 'John';
```

```
v_myName VARCHAR2(20) DEFAULT 'John';
```

- Déclarer chaque identificateur sur une ligne à part pour faciliter la maintenance de code et sa lisibilité.

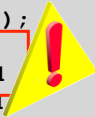
ORACLE

II-18

## Comment déclarer et initialiser des Variables PL/SQL

- Évitez d'utiliser des noms de colonne en tant qu'identificateurs.

```
DECLARE
  employee_id NUMBER(6);
BEGIN
  SELECT employee_id
  INTO   employee_id
  FROM   employees
  WHERE  last_name = 'Kochhar';
END;
/
```



- Utiliser la contrainte NOT NULL lorsque la variable doit recevoir une valeur.

ORACLE

II-19

## Types basiques

- CHAR [(maximum\_length)]
- VARCHAR2 (maximum\_length)
- NUMBER [(precision, scale)]
- BINARY\_INTEGER
- PLS\_INTEGER
- BOOLEAN
- BINARY\_FLOAT
- BINARY\_DOUBLE

ORACLE

II-20



## L' Attribut %TYPE

- Est utilisé pour déclarer une variable selon :
  - une définition de colonne de base de données
  - le type d'une variable déjà déclaré
- Est préfixé par:
  - le nom de la table et de la colonne
  - le nom de la variable déjà déclaré

II-21

ORACLE

## L' Attribut %TYPE

### Syntaxe

```
identifieur      table.column_name%TYPE;
```

### Exemples

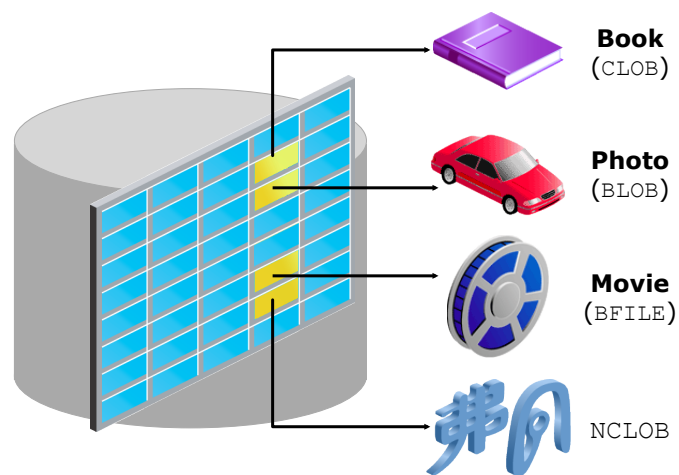
```
...  
emp_lname          employees.last_name%TYPE;  
...
```

```
...  
balance            NUMBER(7,2);  
min_balance        balance%TYPE := 1000;  
...
```

II-23

ORACLE


## LOB Data Type Variables



ORACLE®

11-24

## Composite Data Types

TRUE	23-DEC-98	ATLANTA	
------	-----------	---------	---

## PL/SQL table structure

1	SMITH
2	JONES
3	NANCY
4	TIM

```

└─VARCHAR2
PLS_INTEGER

```

## PL/SQL table structure

1	5000
2	2345
3	12
4	3456

```

└ NUMBER
PLS_INTEGER

```

ORACLE®

11-25

## Composite Data Types

- Peut contenir plusieurs valeurs
- Peut être de deux types:
  - PL/SQL records
  - PL/SQL collections
    - INDEX BY tables or associative arrays
    - Nested table
    - VARRAY

ORACLE

II-26

## %ROWTYPE Attribute

- Déclare une variable selon une collection de colonnes dans une table de base de données ou une vue.
- Préfixer %ROWTYPE avec la table de base de données ou la vue.
- Les champs dans l'enregistrement prennent les noms et les types de données des colonnes de la table ou la vue.

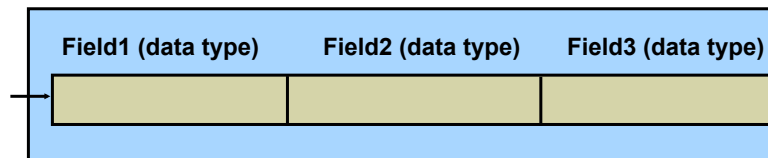
Syntaxe:

```
DECLARE  
  identifier reference%ROWTYPE;
```

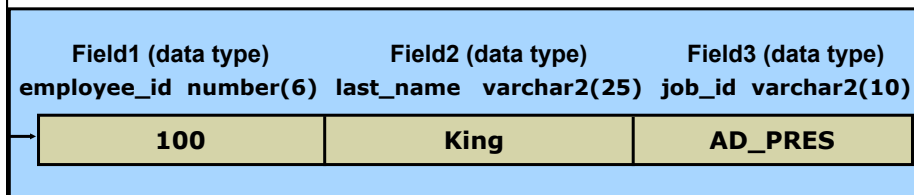
ORACLE

II-27

## PL/SQL Record Structure



### Example:



II-28

ORACLE

## Creating a PL/SQL Record

### Syntax:

**1** `TYPE type_name IS RECORD  
(field_declaration[, field_declaration]...);`

**2** `identifier type_name;`

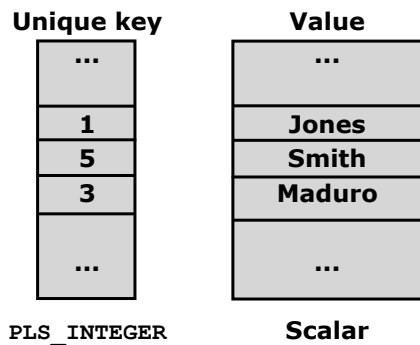
*field\_declaration:*

```
field_name {field_type | variable%TYPE  
           | table.column%TYPE | table%ROWTYPE}  
[[NOT NULL] {:= | DEFAULT} expr]
```

II-30

ORACLE

## INDEX BY Table Structure



II-31

ORACLE

## Creating an INDEX BY Table

```

DECLARE
  TYPE ename_table_type IS TABLE OF
    employees.last_name%TYPE
    INDEX BY PLS_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
    INDEX BY PLS_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;
/

```

	ENAME	HIREDT
1	CAMERON	23-FEB-09

II-32

ORACLE

# ECRIRE DES BLOCKS PL/SQL

ORACLE

II-33

## Commenter le Code

- Préfixer les commentaires monolignes avec deux traits d'Union (--).
- Placer les commentaires de plusieurs lignes entre les symboles /\* et \*/.

### Exemple:

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

ORACLE

II-34

## Utiliser les Functions SQL dans le code PL/SQL: Examples

- Obtenir la longueur d'une chaîne :

```
v_desc_size INTEGER(5);  
v_prod_description VARCHAR2(70):='You can use this  
product with your radios for higher frequency';  
  
-- get the length of the string in prod description  
v_desc_size:= LENGTH(v_prod_description);
```

- Obtenir le nombre de mois, que l'employé a travaillé :

```
v_tenure:= MONTHS_BETWEEN (CURRENT_DATE, v_hiredate);
```

ORACLE

II-35

## Data Type Conversion

- Converts data to comparable data types
- Is of two types:
  - Implicit conversion
  - Explicit conversion
- Functions:
  - TO\_CHAR
  - TO\_DATE
  - TO\_NUMBER
  - TO\_TIMESTAMP

ORACLE

II-36

## Conversion de Type

- ① `date_of_joining DATE:= '02-Feb-2000';`
- ② `date_of_joining DATE:= 'February 02,2000';`
- ③ `date_of_joining DATE:= TO_DATE('February 02,2000','Month DD, YYYY');`

ORACLE

II-38

## Blocks Imbriqués

Les blocs PL/SQL peuvent être imbriqués.

- Une section exécutable (BEGIN ... END) peut contenir des blocs imbriqués.
- Une section d'exception peut contenir des blocs imbriqués.



ORACLE

II-39



## Nested Blocks: Example

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL VARIABLE';
  BEGIN
    DBMS_OUTPUT.PUT_LINE(v_inner_variable);
    DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
  DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```

```
anonymous block completed
LOCAL VARIABLE
GLOBAL VARIABLE
GLOBAL VARIABLE
```

ORACLE

II-40

## Visibilité et Porté des Variables

```
DECLARE
  v_father_name VARCHAR2(20):='Patrick';
  v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20):='Mike';
    v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: '||v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: '||v_child_name);
  END;
  DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/
```

1

2

ORACLE

II-41

## Visibilité et Porté des Variables

```
BEGIN <<outer>>
DECLARE
  v_father_name VARCHAR2(20) := 'Patrick';
  v_date_of_birth DATE := '20-Apr-1972';
BEGIN
  DECLARE
    v_child_name VARCHAR2(20) := 'Mike';
    v_date_of_birth DATE := '12-Dec-2002';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Father's Name: ' || v_father_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: '
      || outer.v_date_of_birth);
    DBMS_OUTPUT.PUT_LINE('Child's Name: ' || v_child_name);
    DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || v_date_of_birth);
  END;
END;
END outer;
```

ORACLE

II-43

## Visibilité et Porté des Variables: Exemple

```
BEGIN <<outer>>
DECLARE
  v_sal      NUMBER(7,2) := 60000;
  v_comm     NUMBER(7,2) := v_sal * 0.20;
  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;
    1 outer.v_comm := v_sal * 0.30;
    END;
    2 v_message := 'SALESMAN' || v_message;
  END;
END outer;
/
```

ORACLE

II-44

## Exercise

```
DECLARE
v_weight    NUMBER(3) := 600;
v_message   VARCHAR2(255) := 'Product 10012';
BEGIN
  DECLARE
    v_weight    NUMBER(3) := 1;
    v_message   VARCHAR2(255) := 'Product 11001';
    v_new_locn  VARCHAR2(50) := 'Europe';
  BEGIN
    v_weight := v_weight + 1;
    v_new_locn := 'Western ' || v_new_locn;
    ① → v_weight at position 1 = ?
        v_new_locn at position 1 = ?
    END;
    v_weight := v_weight + 1;
    v_message := v_message || ' is in stock';
    v_new_locn := 'Western ' || v_new_locn;
    ② → v_message at position 2 = ?
        v_new_locn at position 2 = ?
  END;
/
```

ORACLE

II-46

## INTERACTION AVEC LE SERVEUR DE BASE DE DONNÉES ORACLE

ORACLE

II-51

## SELECT dans PL/SQL

Récupérer des données de la base de données avec une instruction **SELECT**.

Syntaxe :

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```

ORACLE

II-52

## SELECT dans PL/SQL

- La clause **INTO** est requise.
- Les requêtes ne doivent retourner qu'une seule ligne.

```
DECLARE
  v_fname VARCHAR2(25);
BEGIN
  SELECT first_name INTO v_fname
  FROM employees WHERE employee_id=200;
  DBMS_OUTPUT.PUT_LINE(' First Name is : '||v_fname);
END;
/
```

```
anonymous block completed
First Name is : Jennifer
```

ORACLE

II-54

## Récupération de données par PL/SQL: Exemple

Recupérer hire\_date et salary pour un employé.

```
DECLARE
  v_emp_hiredate  employees.hire_date%TYPE;
  v_emp_salary    employees.salary%TYPE;
BEGIN
  SELECT  hire_date, salary
  INTO    v_emp_hiredate, v_emp_salary
  FROM    employees
  WHERE   employee_id = 100;
END;
/
```

ORACLE

II-56

## Récupération de données par PL/SQL

Retourne la somme des salaires pour tous les  
employés d'un département spécifié.

Exemple:

```
DECLARE
  v_sum_sal  NUMBER(10,2);
  v_deptno   NUMBER NOT NULL := 60;
BEGIN
  SELECT SUM(salary) -- group function
  INTO v_sum_sal FROM employees
  WHERE department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum of salary is ' || v_sum_sal);
END;
```

```
anonymous block completed
The sum of salary is 28800
```

ORACLE

II-57

## Récupérer des données : Exemple

Declare variables to store the name, job, and salary of a new employee.

```
DECLARE
  type t_rec is record
    (v_sal number(8),
     v_minsal number(8) default 1000,
     v_hire_date employees.hire_date%type,
     v_rec1 employees%rowtype);
  v_myrec t_rec;
BEGIN
  v_myrec.v_sal := v_myrec.v_minsal + 500;
  v_myrec.v_hire_date := sysdate;
  SELECT * INTO v_myrec.v_rec1
    FROM employees WHERE employee_id = 100;
  DBMS_OUTPUT.PUT_LINE(v_myrec.v_rec1.last_name || ' ' ||
    to_char(v_myrec.v_hire_date) || ' ' || to_char(v_myrec.v_sal));
END;
```

```
anonymous block completed
King 16-FEB-09 1500
```

ORACLE

II-58

## Récupérer des données pour un %ROWTYPE Attribute: Exemple

```
DECLARE
  v_employee_number number:= 124;
  v_emp_rec employees%ROWTYPE;
BEGIN
  SELECT * INTO v_emp_rec FROM employees
  WHERE employee_id = v_employee_number;
  INSERT INTO retired_emps(empno, ename, job, mgr,
    hiredate, leavedate, sal, comm, deptno)
  VALUES (v_emp_rec.employee_id, v_emp_rec.last_name,
    v_emp_rec.job_id, v_emp_rec.manager_id,
    v_emp_rec.hire_date, SYSDATE,
    v_emp_rec.salary, v_emp_rec.commission_pct,
    v_emp_rec.department_id);
END;
/
```

EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124 Mourgos	ST_MAN	100	16-NOV-99	16-FEB-09	5800	(null)	50

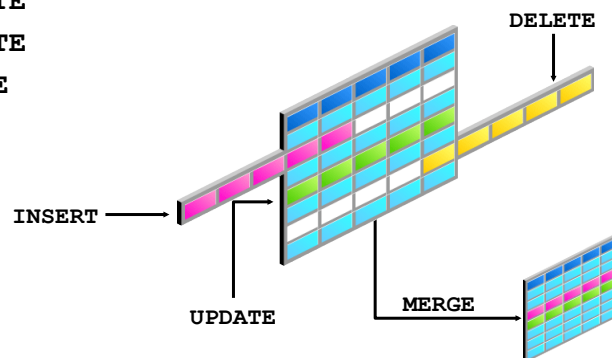
ORACLE

II-59

## Utiliser PL/SQL pour Manipuler des Données

Apporter des modifications aux tables de base de données à l'aide de commandes DML :

- INSERT
- UPDATE
- DELETE
- MERGE



ORACLE

II-68

## Inserer des données : Exemple

```
BEGIN
  INSERT INTO employees
    (employee_id, first_name, last_name, email,
     hire_date, job_id, salary)
  VALUES (employees_seq.NEXTVAL, 'Ruth', 'Cores',
           'RCORES', CURRENT_DATE, 'AD_ASST', 4000);
END;
/
```

ORACLE

II-69

## Inserting a Record by Using %ROWTYPE

```
...
DECLARE
    v_employee_number number := 124;
    v_emp_rec retired_emps%ROWTYPE;
BEGIN
    SELECT employee_id, last_name, job_id, manager_id,
           hire_date, hire_date, salary, commission_pct,
           department_id INTO v_emp_rec FROM employees
    WHERE employee_id = v_employee_number;
    INSERT INTO retired_emps VALUES v_emp_rec;
END;
/
SELECT * FROM retired_emps;
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-99	16-NOV-99	5800	(null)	50

ORACLE

II-70

## Mettre à jour des données: Exemple

```
DECLARE
    sal_increase employees.salary%TYPE := 800;
BEGIN
    UPDATE employees
    SET salary = salary + sal_increase
    WHERE job_id = 'ST_CLERK';
END;
/
```

```
anonymous block completed
FIRST_NAME      SALARY
-----
Julia            4000
Irene            3500
James            3200
Steven           3000
```

```
...
Curtis           3900
Randall          3400
Peter            3300
20 rows selected
```

ORACLE

II-71



## Updating a Row in a Table by Using a Record

```
SET VERIFY OFF
DECLARE
  v_employee_number number:= 124;
  v_emp_rec retired_emps%ROWTYPE;
BEGIN
  SELECT * INTO v_emp_rec FROM retired_emps;
  v_emp_rec.leavedate:=CURRENT_DATE;
  UPDATE retired_emps SET ROW = v_emp_rec WHERE
    empno=v_employee_number;
END;
/
SELECT * FROM retired_emps;
```

	EMPNO	ENAME	JOB	MGR	HIREDATE	LEAVEDATE	SAL	COMM	DEPTNO
1	124	Mourgos	ST_MAN	100	16-NOV-99	16-FEB-09	5800	(null)	50

ORACLE

II-72

## Supprimer des données: Exemple

```
DECLARE
  deptno employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM employees
  WHERE department_id = deptno;
END;
/
```

ORACLE

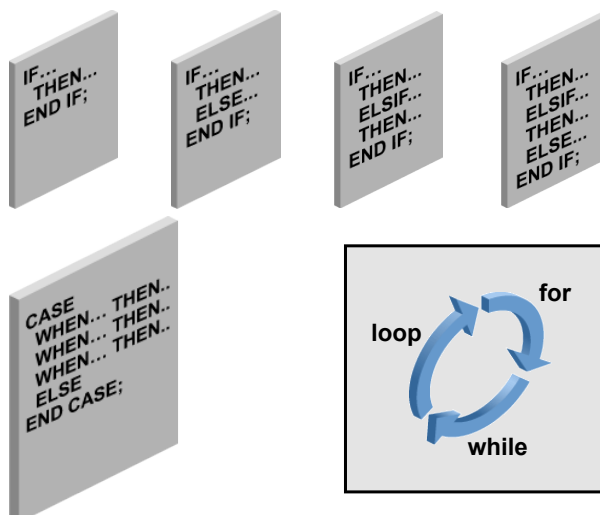
II-73

# STRUCTURES DE CONTRÔLE

ORACLE

II-77

## Structures de contrôle et boucle



ORACLE

II-78

## IF Statement

### Syntax:

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

ORACLE

11-79

## Simple IF Statement

```
DECLARE
    v_myage number:=31;
BEGIN
    IF v_myage < 11
    THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child ');
    END IF;
END;
/
```

anonymous block completed

ORACLE

11-81

## IF THEN ELSE Statement

```
DECLARE
v_myage  number:=31;
BEGIN
IF v_myage < 11
THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

ORACLE

11-82

## IF ELSIF ELSE Clause

```
DECLARE
    v_myage number:=31;
BEGIN
    IF v_myage < 11 THEN
        DBMS_OUTPUT.PUT_LINE(' I am a child ');
    ELSIF v_myage < 20 THEN
        DBMS_OUTPUT.PUT_LINE(' I am young ');
    ELSIF v_myage < 30 THEN
        DBMS_OUTPUT.PUT_LINE(' I am in my twenties');
    ELSIF v_myage < 40 THEN
        DBMS_OUTPUT.PUT_LINE(' I am in my thirties');
    ELSE
        DBMS_OUTPUT.PUT_LINE(' I am always young ');
    END IF;
END;
/
```

```
anonymous block completed
I am in my thirties
```

ORACLE

11-83

## NULL Value in IF Statement

```
DECLARE
  v_myage  number;
BEGIN
  IF v_myage < 11 THEN
    DBMS_OUTPUT.PUT_LINE(' I am a child ');
  ELSE
    DBMS_OUTPUT.PUT_LINE(' I am not a child ');
  END IF;
END;
/
```

```
anonymous block completed
I am not a child
```

ORACLE

11-84

## CASE Expressions

- Une expression CASE sélectionne un résultat et le retourne.
- Pour sélectionner le résultat, l'expression CASE utilise des expressions. La valeur retournée par ces expressions est utilisée pour sélectionner une ou plusieurs alternatives.

```
CASE selector
  WHEN expression1 THEN result1
  WHEN expression2 THEN result2
  ...
  WHEN expressionN THEN resultN
  [ELSE resultN+1]
END;
/
```

ORACLE

11-85

## CASE Expressions: Example

```
SET VERIFY OFF
DECLARE
    v_grade CHAR(1) := UPPER('&grade');
    appraisal VARCHAR2(20);
BEGIN
    appraisal := CASE v_grade
        WHEN 'A' THEN 'Excellent'
        WHEN 'B' THEN 'Very Good'
        WHEN 'C' THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || '
                          Appraisal ' || appraisal);
END;
/
```

ORACLE

11-86

## Searched CASE Expressions

```
DECLARE
    v_grade CHAR(1) := UPPER('&grade');
    appraisal VARCHAR2(20);
BEGIN
    appraisal := CASE
        WHEN v_grade = 'A' THEN 'Excellent'
        WHEN v_grade IN ('B', 'C') THEN 'Good'
        ELSE 'No such grade'
    END;
    DBMS_OUTPUT.PUT_LINE ('Grade: ' || v_grade || '
                          Appraisal ' || appraisal);
END;
/
```

ORACLE

11-87

## CASE Statement

```
DECLARE
  v_deptid NUMBER;
  v_deptname VARCHAR2(20);
  v_emps NUMBER;
  v_mngid NUMBER:= 108;
BEGIN
  CASE v_mngid
    WHEN 108 THEN
      SELECT department_id, department_name
        INTO v_deptid, v_deptname FROM departments
        WHERE manager_id=108;
      SELECT count(*) INTO v_emps FROM employees
        WHERE department_id=v_deptid;
    WHEN 200 THEN
      ...
  END CASE;
  DBMS_OUTPUT.PUT_LINE ('You are working in the '|| deptname||
    ' department. There are '||v_emps ||' employees in this
    department');
END;
/
```

ORACLE

II-88

## Exercice 1:

- **Ecrire un programme permettant**
  - de saisir en entrée le nom d'un employé
  - de mettre à jour le salaire de cet employé en lui ajoutant :
    - 500 \$ s'il appartient au département vente
    - 300 \$ s'il appartient au département opérateurs
    - 400 \$ s'il appartient au département SI
    - 600 \$ s'il appartient au département Administrateurs

**NB:** Faites l'exercice avec

- IF ..END IF
- CASE Expression
- CASE Statement

ORACLE

II-89

## Exercice 2:

Écrire un programme permettant

**de saisir en entrée le nom d 'un employé**

**de mettre à jour le salaire d 'un employé suivant les conditions suivantes:**

Si son année d 'entrée est 1990, augmenter son salaire de 50%

Si son année d 'entrée est 1991, augmenter son salaire de 25%

Si son année d 'entrée est 1992, augmenter son salaire de 10%

ORACLE

II-90

## Traitements itératifs : LOOP Statements

- Une boucle répète une instruction (ou la séquence d'instructions) plusieurs fois.
- Trois types de boucle:
  - Basic loop
  - FOR loop
  - WHILE loop



ORACLE

II-91



## Boucle de base

### Syntaxe:

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

Boucle de base qui permet la répétition d'une séquence d'instructions.

ORACLE

II-92

## Basic Loops

### Example:

```
DECLARE
  v_countryid    locations.country_id%TYPE := 'CA';
  v_loc_id       locations.location_id%TYPE;
  v_counter      NUMBER(2) := 1;
  v_new_city     locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_loc_id FROM locations
  WHERE country_id = v_countryid;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```

ORACLE

II-93

## La boucle WHILE

### Syntaxe:

```
WHILE condition LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

La boucle WHILE répète les instructions tant que condition est TRUE.

ORACLE

II-94

## WHILE Loops: Example

```
DECLARE
    v_countryid    locations.country_id%TYPE := 'CA';
    v_loc_id       locations.location_id%TYPE;
    v_new_city     locations.city%TYPE := 'Montreal';
    v_counter      NUMBER := 1;
BEGIN
    SELECT MAX(location_id) INTO v_loc_id FROM locations
    WHERE country_id = v_countryid;
    WHILE v_counter <= 3 LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_loc_id + v_counter), v_new_city, v_countryid);
        v_counter := v_counter + 1;
    END LOOP;
END;
/
```

ORACLE

II-95

## La boucle FOR

- Le nombre d'itérations est connu avant d'entrer dans la boucle.
- Ne pas déclarer le compteur ; Il est déclaré implicitement.

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
    statement1;
    statement2;
    . . .
END LOOP;
```

ORACLE

II-96

## La boucle FOR : Exemple

```
DECLARE
    v_countryid  locations.country_id%TYPE := 'CA';
    v_loc_id     locations.location_id%TYPE;
    v_new_city   locations.city%TYPE := 'Montreal';
BEGIN
    SELECT MAX(location_id) INTO v_loc_id
    FROM locations
    WHERE country_id = v_countryid;
    FOR i IN 1..3 LOOP
        INSERT INTO locations(location_id, city, country_id)
        VALUES((v_loc_id + i), v_new_city, v_countryid );
    END LOOP;
END;
/
```

ORACLE

II-98

## Exercice 3

- 2..** Create a PL/SQL block that inserts an asterisk in the `stars` column for every \$1,000 of the employee's salary.
- In the declarative section of the block,
    - declare a variable `v_empno` of type `emp.employee_id` and initialize it to 176.
    - declare a variable `v_asterisk` of type `emp.stars` and initialize it to `NULL`.
    - create a variable `sal` of type `emp.salary`.
  - In the executable section, write logic to append an asterisk (\*) to the string for every \$1,000 of the salary amount. For example, if the employee earns \$8,000, the string of asterisks should contain eight asterisks. If the employee earns \$12,500, the string of asterisks should contain 13 asterisks.
  - Update the `stars` column for the employee with the string of asterisks. Commit before the end of the block.

ORACLE

II-99

## Exercice 4

Écrire un programme permettant la mise à jour des salaires de tous les employés suivant les conditions de l'exercice2.

Remarques ..

ORACLE

II-102

# USING EXPLICIT CURSORS

ORACLE

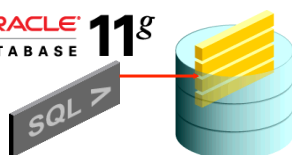
II-103

## Cursors

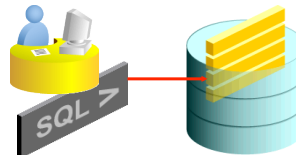
Chaque instruction SQL exécutée par le serveur Oracle a un curseur individuel associé qui est soit un:

- curseur implicite : Déclaré et géré par le PL/SQL pour toutes les instructions DML et PL/SQL SELECT
- curseur explicite : déclaré et géré par le programmeur

ORACLE 11<sup>g</sup>  
DATABASE



Implicit cursor

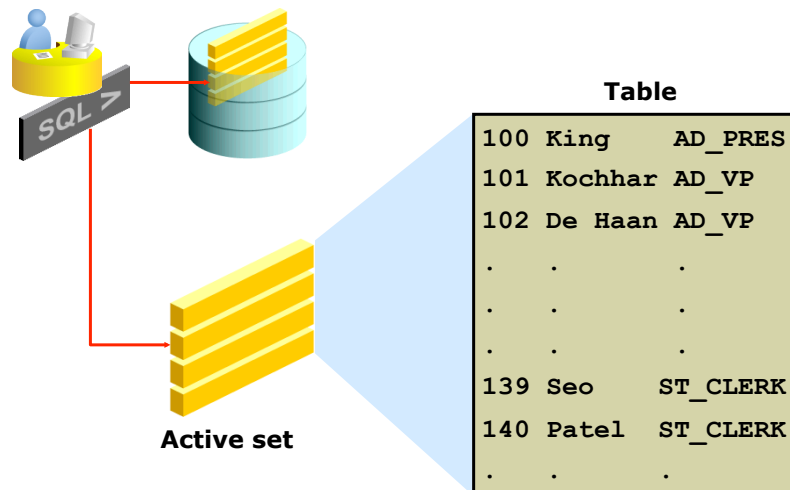


Explicit cursor

ORACLE

II-104

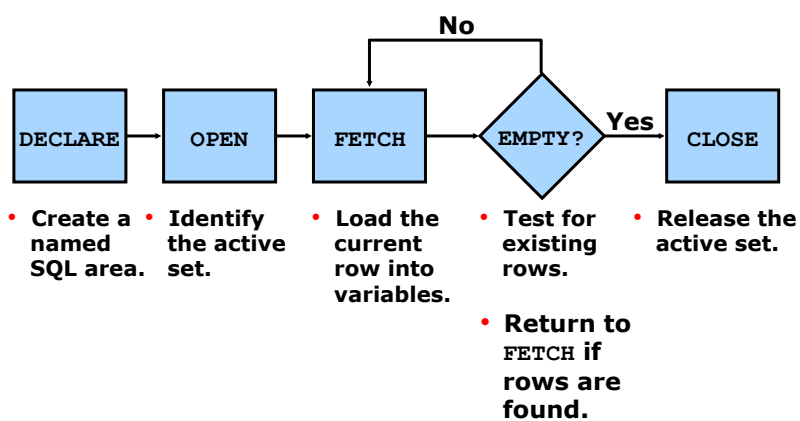
## Explicit Cursor Operations



II-105

ORACLE

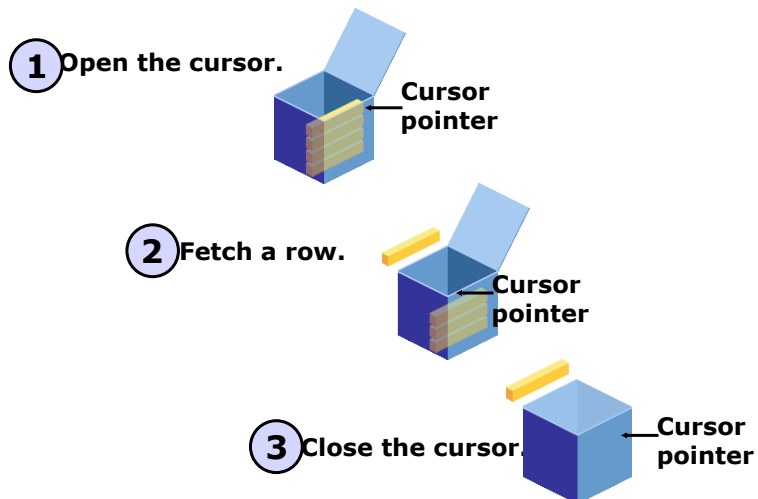
## Controlling Explicit Cursors



II-106

ORACLE

## Controlling Explicit Cursors



II-107

ORACLE

## Declaring the Cursor

### Syntax:

```
CURSOR cursor_name IS  
    select_statement;
```

### Examples:

```
DECLARE  
    CURSOR c_emp_cursor IS  
        SELECT employee_id, last_name FROM employees  
        WHERE department_id = 30;
```

```
DECLARE  
    v_locid NUMBER := 1700;  
    CURSOR c_dept_cursor IS  
        SELECT * FROM departments  
        WHERE location_id = v_locid;  
    ...
```

II-108

ORACLE

## Opening the Cursor

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
  ...
BEGIN
  OPEN c_emp_cursor;
```

ORACLE

11-110

## Fetching Data from the Cursor

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  FETCH c_emp_cursor INTO v_empno, v_lname;
  DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
END;
/
```

```
anonymous block completed
114 Raphaely
```

ORACLE

11-111



## Fetching Data from the Cursor

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
  v_empno employees.employee_id%TYPE;
  v_lname employees.last_name%TYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_empno, v_lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP;
END;
/
```

ORACLE

II-113

## Closing the Cursor

```
...
  LOOP
    FETCH c_emp_cursor INTO empno, lname;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_empno ||' '||v_lname);
  END LOOP;
  CLOSE c_emp_cursor;
END;
/
```

ORACLE

II-114

## Cursors and Records

Process the rows of the active set by fetching values into a PL/SQL record.

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id = 30;
  v_emp_record c_emp_cursor%ROWTYPE;
BEGIN
  OPEN c_emp_cursor;
  LOOP
    FETCH c_emp_cursor INTO v_emp_record;
    EXIT WHEN c_emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
                          || ' ' || v_emp_record.last_name);
  END LOOP;
  CLOSE c_emp_cursor;
END;
```

ORACLE

II-115

## Cursor FOR Loops

Syntax:

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- The cursor FOR loop is a shortcut to process explicit cursors.
- Implicit open, fetch, exit, and close occur.
- The record is implicitly declared.

ORACLE

II-116

## Cursor FOR Loops

```
DECLARE
  CURSOR c_emp_cursor IS
    SELECT employee_id, last_name FROM employees
    WHERE department_id =30;
BEGIN
  FOR emp_record IN c_emp_cursor
  LOOP
    DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
      || ' ' || emp_record.last_name);
  END LOOP;
END;
/
```

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

ORACLE

11-117

## Explicit Cursor Attributes

Use explicit cursor attributes to obtain status information about a cursor.

Attribute	Type	Description
%ISOPEN	Boolean	Evaluates to TRUE if the cursor is open
%NOTFOUND	Boolean	Evaluates to TRUE if the most recent fetch does not return a row
%FOUND	Boolean	Evaluates to TRUE if the most recent fetch returns a row; complement of %NOTFOUND
%ROWCOUNT	Number	Evaluates to the total number of rows returned so far

ORACLE

11-118

## %ISOPEN Attribute

- Fetch rows only when the cursor is open.
- Use the %ISOPEN cursor attribute before performing a fetch to test whether the cursor is open.

Example:

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

ORACLE

II-119

## %ROWCOUNT and %NOTFOUND: Example

```
DECLARE
    CURSOR c_emp_cursor IS SELECT employee_id,
        last_name FROM employees;
    v_emp_record c_emp_cursor%ROWTYPE;
BEGIN
    OPEN c_emp_cursor;
    LOOP
        FETCH c_emp_cursor INTO v_emp_record;
        EXIT WHEN c_emp_cursor%ROWCOUNT > 10 OR
            c_emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE( v_emp_record.employee_id
            || ' ' || v_emp_record.last_name);
    END LOOP;
    CLOSE c_emp_cursor;
END ; /
```

```
anonymous block completed
198 OConnell
199 Grant
200 Whalen
201 Hartstein
202 Fay
203 Mavris
204 Baer
205 Higgins
206 Gietz
100 King
```

ORACLE

II-120

## Cursor FOR Loops Using Subqueries

There is no need to declare the cursor.

```
BEGIN
  FOR emp_record IN (SELECT employee_id, last_name
    FROM employees WHERE department_id =30)
  LOOP
    DBMS_OUTPUT.PUT_LINE( emp_record.employee_id
      || ' ' || emp_record.last_name);
  END LOOP;
END;
/
```

```
anonymous block completed
114 Raphaely
115 Khoo
116 Baida
117 Tobias
118 Himuro
119 Colmenares
```

ORACLE

II-121

## Cursors with Parameters

**Syntax:**

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

- Passer des valeurs de paramètre à un curseur lorsque le curseur est ouvert et que la requête est exécutée.
- Ouvrir un curseur explicite plusieurs fois avec un actif différent à chaque fois.

```
OPEN cursor_name(parameter_value,.....) ;
```

ORACLE

II-122

## Cursors with Parameters

```
DECLARE
  CURSOR c_emp_cursor (deptno NUMBER) IS
    SELECT employee_id, last_name
    FROM employees
    WHERE department_id = deptno;
  ...
BEGIN
  OPEN c_emp_cursor (10);
  ...
  CLOSE c_emp_cursor;
  OPEN c_emp_cursor (20);
  ...
```

```
anonymous block completed
200 Whalen
201 Hartstein
202 Fay
```

ORACLE

II-123

## FOR UPDATE Clause

### Syntax:

```
SELECT ...
FROM ...
FOR UPDATE [OF column_reference] [NOWAIT | WAIT n];
```

- Utilise le verrouillage explicite pour refuser l'accès aux autres sessions pendant toute la durée d'une transaction.
- Verrouille les lignes avant la mise à jour ou la suppression.

ORACLE

II-124

## WHERE CURRENT OF Clause

### Syntax:

```
WHERE CURRENT OF cursor ;
```

- Utilise le curseur pour mettre à jour ou supprimer la ligne actuelle.
- Inclure la clause FOR UPDATE dans la requête de curseur pour verrouiller les lignes d'abord.
- Use the WHERE CURRENT OF clause to reference the current row from an explicit cursor.

```
UPDATE employees  
  SET    salary = ...  
  WHERE CURRENT OF c_emp_cursor;
```

ORACLE

II-126

## Cursors with Subqueries: Example

```
DECLARE  
  CURSOR my_cursor IS  
    SELECT t1.department_id, t1.department_name,  
           t2.staff  
    FROM   departments t1, (SELECT department_id,  
                                COUNT(*) AS staff  
                           FROM employees  
                           GROUP BY department_id) t2  
    WHERE  t1.department_id = t2.department_id  
    AND    t2.staff >= 3;  
  ...
```

ORACLE

II-127

## Exercices 1

1. Créez un bloc PL/SQL qui effectue les opérations suivantes:

- a. dans la section déclarative, déclarez une variable `v_deptno` de type numérique et assigner une valeur qui contient l'ID du département.
- b. Déclarer un curseur, `c_emp_cursor`, qui extrait le `last_name`, `salaire` et `manager_id` des employés travaillant dans le département spécifié dans `v_deptno`.
- c. dans la section exécutable, utilisez le curseur FOR loop pour opérer sur les données récupérées.
  - i. Si le salaire de l'employé est inférieur à 5 000, et si l'ID du manager est 101 ou 124, afficher le message de `<<last_name>> Due for a raise.`
  - ii. Dans le cas contraire, afficher le message `<<last_name>> Not due for a raise.`

ORACLE

II-130

## Exercice 2

- a. Dans la section déclarative, déclarez un curseur `dept_cursor` pour récupérer `departement_id` et `department_name` avec `departement_id` inférieur à 100, ordonnées par `departement_id`.
- b. Déclarer un autre curseur, `emp_cursor`, qui prend le numéro de département en tant que paramètre et récupère `last_name`, `job_id`, `hire_date` et le salaire des employés dont `employee_id` est inférieure à 120 et qui travaillent dans ce département.
- c. Déclarer des variables pour contenir des valeurs extraites de chaque curseur.
- d. Ouvrir `dept_cursor`, utilisez une boucle simple et extraire les valeurs dans les variables déclarées. Afficher le numéro et le nom du département .
- e. Pour chaque département, ouvrez `emp_cursor` en passant le numéro actuel du département en tant que paramètre. Commencer une autre boucle et extraire les valeurs d'`emp_cursor` dans des variables et imprimer tous les détails qui provient de la table `employees` . *Remarque* : Vous pouvez imprimer une ligne après avoir affiché les détails de chaque département. Utilisez des attributs appropriés pour la condition de sortie. En outre, déterminer si un curseur est déjà ouvert avant de l'ouvrir.
- a. Fermez tous les curseurs et les boucles et puis terminer la section exécutable.

ORACLE

II-131



### Exercice 3

Create a PL/SQL block that determines the top *n* salaries of the employees.

- a. In the declarative section, declare a variable `v_num` of type `NUMBER` that holds a number *n* representing the number of top *n* earners from the `employees` table. For example, to view the top five salaries, enter 5. Declare another variable `sal` of type `employees.salary`. Declare a cursor, `c_emp_cursor`, that retrieves the salaries of employees in descending order.
- b. In the executable section, open the loop and fetch top *n* salaries and insert them into `top_salaries` table. You can use a simple loop to operate on the data. Also, try and use `%ROWCOUNT` and `%FOUND` attributes for the exit condition.

**Note:** Make sure you add an exit condition to avoid having an infinite loop.

ORACLE

II-132

### Cursor Variables

- Variables de curseur sont comme des pointeurs C ou Pascal, qui détiennent l'emplacement de la mémoire (adresse) d'un élément au lieu de l'élément lui-même.
- Dans PL/SQL, un pointeur est déclaré comme `REF X`, où `REF` est une abréviation de `REFERENCE` et `X` représente une classe d'objets.
- Une variable de curseur dispose du type `REF CURSOR`.
- Un `CURSOR` est statique alors qu'un `REF CURSOR` est dynamique.

ORACLE

II-133

## Using Cursor Variables

- Vous pouvez utiliser des variables de curseur pour passer des jeux de résultats de requête entre les sous-programmes stockés PL/SQL et divers clients.
- PL/SQL peut partager un pointeur vers la zone de travail de requête dans lequel est stocké le jeu de résultats
- Vous pouvez passer la valeur d'une variable de curseur librement d'un champ d'application à une autre.

ORACLE

II-134

## Defining REF CURSOR Types

**Define a REF CURSOR type:**

```
Define a REF CURSOR type  
TYPE ref_type_name IS REF CURSOR [RETURN return_type];
```

**Declare a cursor variable of that type:**

```
ref_cv ref_type_name;
```

**Example:**

```
DECLARE  
TYPE DeptCurTyp IS REF CURSOR RETURN  
departments%ROWTYPE;  
dept_cv DeptCurTyp;
```

ORACLE

II-135

## Using the OPEN-FOR, FETCH, and CLOSE Statements

- L'instruction **OPEN-FOR**, associe un curseur à une requête multi ligne, exécute la requête, identifie le jeu de résultats et positionne le curseur vers la première ligne du jeu de résultats.
- L'instruction **FETCH** retourne une ligne du résultat de la requête multi ligne, assigne les valeurs des éléments de la liste de sélection à des variables ou des champs dans la clause **INTO**, incrémente le décompte tenu par **% ROWCOUNT** et avance le curseur à la ligne suivante.
- L'instruction **CLOSE** désactive une variable de curseur.

ORACLE

II-138

## Opening REF CURSOR

```
OPEN NOM_CURSEUR FOR REQUETE  
[ USING [ IN | OUT | IN OUT ] ARGUMENT[,... ] ] ;
```

**USING:** Cette clause permet le paramétrage de la requête SQL dynamique en utilisant une liste des arguments.

**IN ARGUMENT :** L'argument est passé à la requête SQL dynamique lors de son invocation. Il ne peut pas être modifié à l'intérieur de la requête SQL dynamique.

**OUT ARGUMENT :** L'argument est ignoré lors de l'invocation de la requête SQL dynamique. À l'intérieur de celle-ci, l'argument se comporte comme une variable PL/SQL n'ayant pas été initialisée, contenant donc la valeur « NULL » et supportant les opérations de lecture et d'écriture. Au terme de la requête SQL dynamique, il retourne à la valeur affectée.

**IN OUT ARGUMENT:** L'argument combine les deux propriétés « IN » et « OUT ».

ORACLE

II-141

## Example of Fetching

```
DECLARE
  TYPE EmpCurTyp IS REF CURSOR;
  emp_cv   EmpCurTyp;
  emp_rec  employees%ROWTYPE;
  sql_stmt VARCHAR2(200);
  my_job   VARCHAR2(10) := 'ST_CLERK';
BEGIN
  sql_stmt := 'SELECT * FROM employees
              WHERE job_id = :j';
  OPEN emp_cv FOR sql_stmt USING my_job;
  LOOP
    FETCH emp_cv INTO emp_rec;
    EXIT WHEN emp_cv%NOTFOUND;
    -- process record
  END LOOP;
  CLOSE emp_cv;
END;
/
```

ORACLE

II-142

## Exercice

Créez le bloc PL/SQL qui permet d'afficher toute les lignes de l'une des tables :

- EMPLOYES\_1996,
- EMPLOYES\_1997,
- EMPLOYES\_1998, ...
- EMPLOYES\_YYYY

Dynamiquement, suivant l'année passée en argument,

- vous testez que la table existe et vous affichez tous les enregistrements de la table.
- Si la table n'existe pas, vous affichez tous les enregistrements de la table EMPLOYES pour l'année qui a été passée en argument.

ORACLE

II-143