www.itsci.mju.ac.th/sayan

LEC 08: COMPOSITE DATA TYPES

SAYAN UNANKARD
1/2558

COMPOSITE DATA TYPES

- ใช้สำหรับจัดการค่าข้อมูลที่ประกอบด้วยหลาย ๆ ค่า (multiple values)
- ประกอบด้วย 2 ประเภท
  - PL/SQL records
  - PL/SQL collections
    - INDEX BY tables หรือ associative arrays

COMPOSITE DATA TYPES

- ใช้ PL/SQL records เมื่อต้องการจัดเก็บข้อมูลที่ชนิดข้อมูลแตกต่างกัน แต่จะเข้าถึงได้เพียงครั้งเดียว
- ใช้ PL/SQL collections เมื่อต้องการจัดเก็บข้อมูลที่มีชนิดของข้อมูลเหมือนกัน

PL/SQL RECORDS

- ประกอบด้วยหนึ่ง หรือ มากกว่าฟิลด์ข้อมูล ของตัวแปร scalar, RECORD, หรือ INDEX BY table data type
- มีลักษณะเหมือนตัวแปรแบบโครงสร้าง (structures) ของโปรแกรมภาษา C และ C++
- นักพัฒนาระบบกำหนดเองได้ หรือ สามารถเป็นแถวข้อมูลในตารางได้
- จัดการข้อมูลในลักษณะของ logical unit
- สะดวกสำหรับการอ่านข้อมูลจากตารางเพื่อมาประมวลผล

```
type product  is record  (
        productid number(2),
        productname varchar2(50)
);
v_product  product;
```

CREATING A PL/SQL RECORD

Syntax:

1

```
TYPE type_name IS RECORD
     (field_declaration[, field_declaration]…);
```

2

```
identifiertype_name;
```

field_declaration:

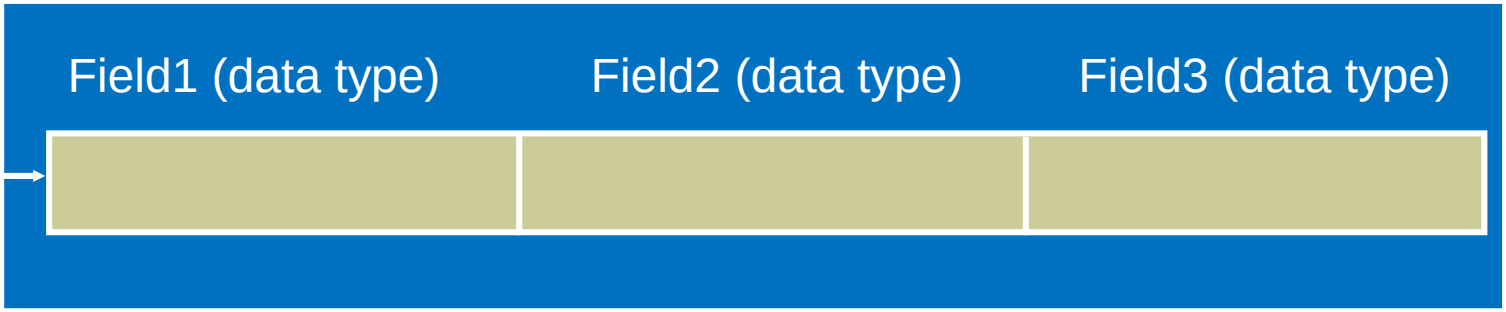
```
field_name {field_type | variable%TYPE
           | table.column%TYPE | table%ROWTYPE}
           [[NOT NULL] {:= | DEFAULT} expr]
```

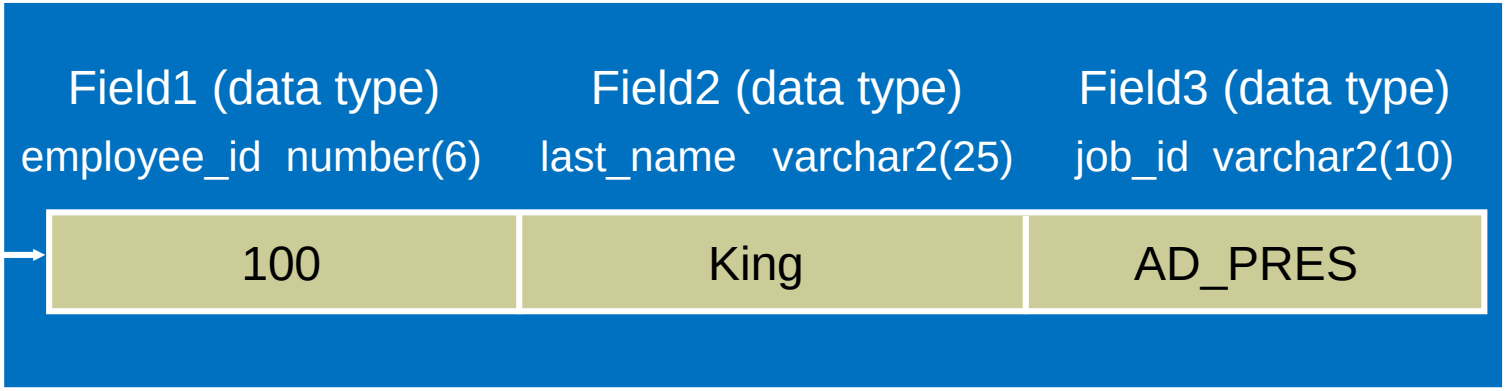
CREATING A PL/SQL RECORD

ประกาศตัวแปรเพื่อจัดเก็บข้อมูล name, job และ salary ของพนักงานใหม่ เช่น

```
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;
```

PL/SQL RECORD STRUCTURE





%ROWTYPE ATTRIBUTE

- เป็นการประกาศตัวแปรตามชนิดข้อมูลในตารางหรือวิว ของฐานข้อมูล
- ต้องนำหน้าคำสั่ง %ROWTYPE ด้วยชื่อตาราง หรือ วิว
- ฟิลด์ในเรคคอร์ดของข้อมูลจะได้ชื่อและชนิดของข้อมูล มาจากคอลัมน์ในตารางหรือ วิว นั่นเอง

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

```
v_product  product%ROWTYPE;
```

ADVANTAGES OF USING %ROWTYPE

ข้อดีคือ

- จำนวนและชนิดของข้อมูล ในตารางฐานข้อมูลไม่จำเป็นต้องรู้ และสามารถเปลี่ยนแปลงได้ในขณะ run time
- %ROWTYPE attribute มีประโยชน์เมื่อต้องการนำข้อมูลออกมาจากฐานข้อมูลเมื่อใช้คำสั่ง SELECT *

%ROWTYPE ATTRIBUTE: EXAMPLE

```
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;
/
```

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;
```

UPDATING A ROW IN A TABLE BY USING A RECORD

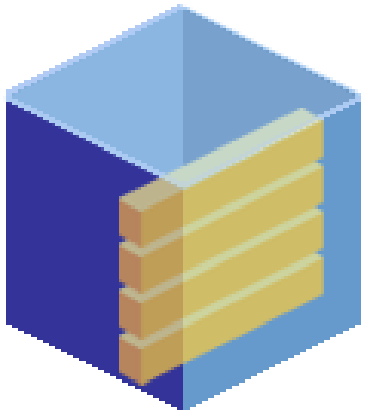
```
SET VERIFY OFF
DECLARE
  v_employee_number number:= 124;
  v_emp_rec retired_emps%ROWTYPE;
BEGIN
 SELECT * INTO v_emp_rec FROM retired_emps;
 v_emp_rec.leavedate:=CURRENT_DATE;
 UPDATE retired_emps SET ROW = v_emp_rec WHERE
  empno=v_employee_number;
END;
/
SELECT * FROM retired_emps;
```

จัดการข้อมูลด้วยเทคนิคของ PL/SQL CURSOR

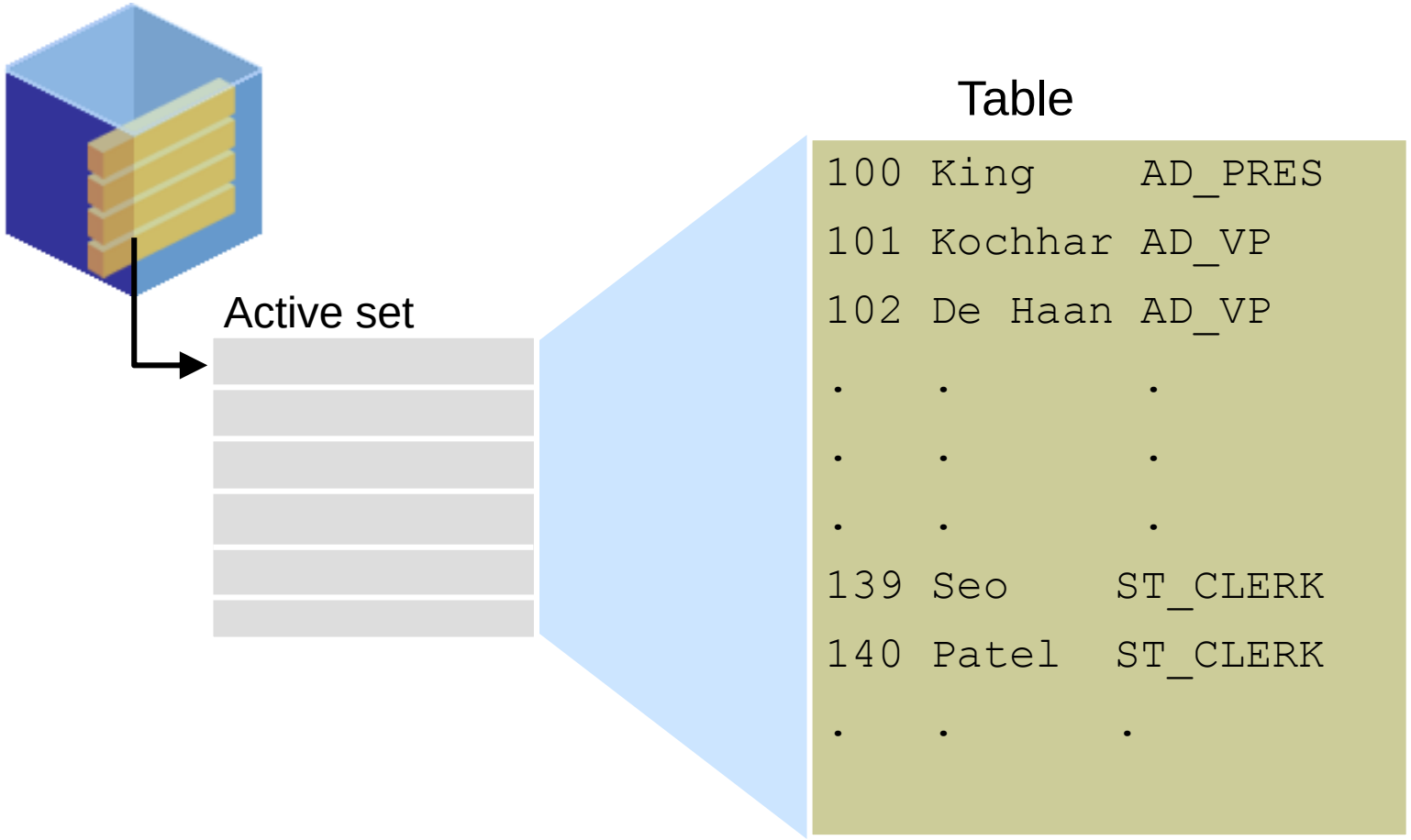
CURSORS

ทุก ๆ คำสั่ง SQL ที่ประมวลผลโดย Oracle server จะถูกกำหนดให้ใช้เคอร์เซอร์ในแต่ละคำสั่ง SQL ซึ่งประกอบด้วย 2 ประเภทของเคอร์เซอร์

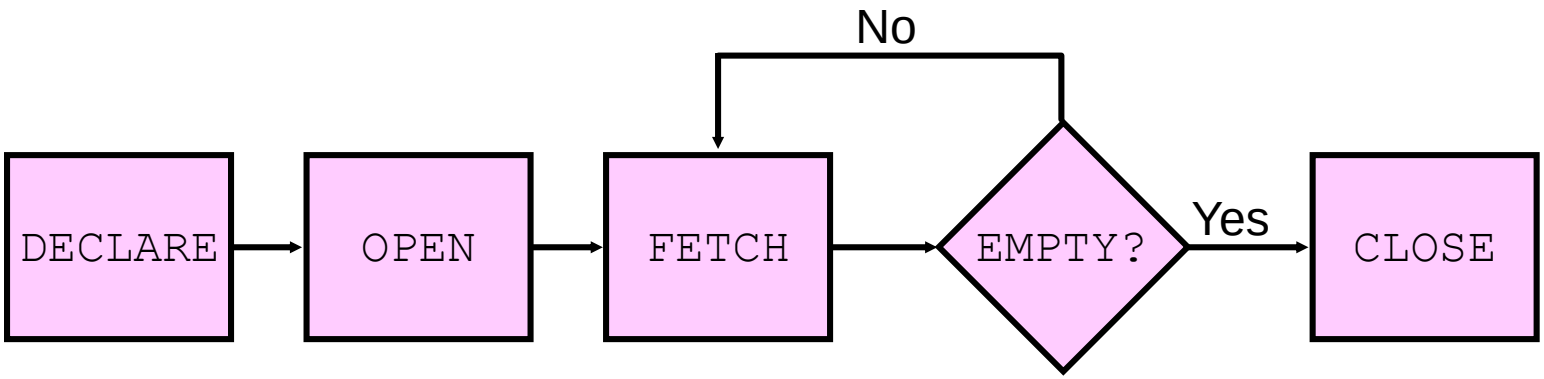
- Implicit cursors : ถูกประกาศและจัดการโดย PL/SQL สำหรับคำสั่ง DML และ PL/SQL SELECT statements
- Explicit cursors : ถูกประกาศและจัดการขึ้นเองโดยนักพัฒนาโปรแกรม



EXPLICIT CURSOR OPERATIONS



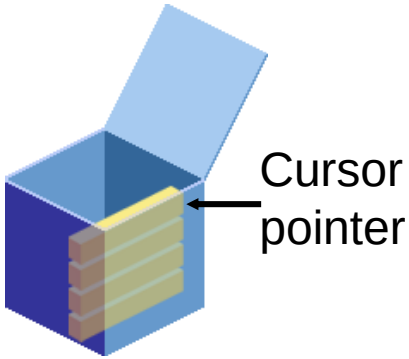
CONTROLLING EXPLICIT CURSORS



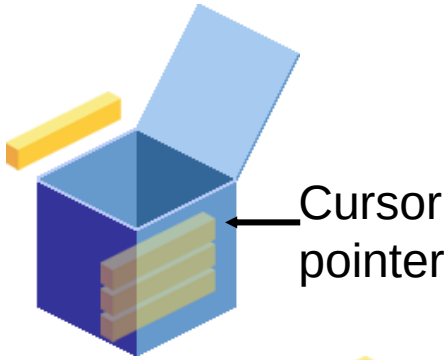
- สร้างชุดคำสั่ง SQL
- กำหนด active set
- อ่านข้อมูลแถวปัจจุบันลงในตัวแปร
- ตรวจสอบเพื่อออกจาก loop
- กลับไปยังส่วนของ FETCH หากยังมีข้อมูลอยู่
- คืนค่า active set

CONTROLLING EXPLICIT CURSORS

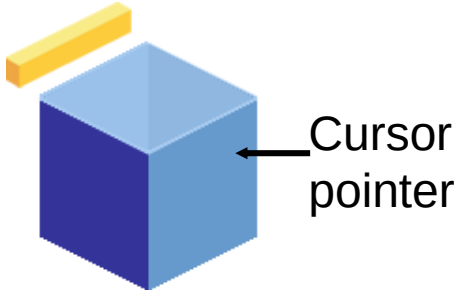
1 Open the cursor.



2 Fetch a row.



3 Close the cursor.



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;
...
```

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;
```

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;

/
```

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;
/
```

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;
/
```

CURSORS AND RECORDS

ประมวลผลข้อมูลจาก active set โดยอ่านข้อมูลเก็บในตัวแปร 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;
```

CURSOR FOR LOOPS

Syntax:

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

- การใช้คำสั่ง FOR loop สำหรับการทำงานวนซ้ำของเคอร์เซอร์
- จะทำ open, fetch, exit และ close เคอร์เซอร์เอง
- ตัวแปร record จะถูกประกาศอัตโนมัติเช่นกัน

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;
/
```

EXPLICIT CURSOR ATTRIBUTES

ส่วนของเคอร์เซอร์ที่ต้องประกาศเอง จะมีส่วนของแอททริบิวส์ที่ถูกเรียกใช้ในการจัดการข้อมูล

SQL%FOUND	คืนค่า boolean หากผลลัพธ์ที่ได้ มี ข้อมูลจะคืนค่า True
SQL%NOTFOUND	คืนค่า boolean หากผลลัพธ์ที่ได้ ไม่มี ข้อมูลจะคืนค่า True
SQL%ROWCOUNT	คืนค่าจำนวนแถวของผลลัพธ์ที่ได้
%ISOPEN	คืนค่า boolean หากมีการเปิดใช้งาน cursor จะคืนค่า True

%ISOPEN ATTRIBUTE

- จะทำการ Fetch ข้อมูลได้เมื่อมีการเปิดใช้ cursor แล้วเท่านั้น
- ใช้คำสั่ง %ISOPEN ใช้ตรวจสอบว่าเคอร์เซอร์ถูกเปิดขึ้นหรือยัง

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

%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 ;
/
```

CURSOR FOR LOOPS USING SUBQUERIES

สามารถใช้คำสั่ง SQL แทนตัวแปรเคอร์เซอร์ได้ โดยไม่ต้องประกาศตัวแปรเคอร์เซอร์

Example:

```
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;
/
```

CURSORS WITH PARAMETERS

Syntax:

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

- การส่งพารามิเตอร์ไปยังเคอร์เซอร์ เมื่อมีการเปิดใช้
- ซึ่งทำให้ได้ผลลัพธ์ที่แตกต่างกันในแต่ละครั้งขึ้นอยู่กับพารามิเตอร์

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

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);
  ...
```

FOR UPDATE CLAUSE

Syntax:

```
SELECT ...
FROM       ...
FOR UPDATE [OF column_reference,column_reference]
[NOWAIT];
```

- มีการล็อคเทคนิคในการป้องกันการเข้าถึงข้อมูลระหว่างที่มี ทรานแซคชันอื่นๆ กำลังทำงาน
- การล็อคจะทำก่อนการแก้ไข และ ลบ

WHERE CURRENT OF CLAUSE

Syntax:

```
WHERE CURRENT OF cursor ;
```

- ใช้เคอร์เซอร์ในการแก้ไขและลบแถวข้อมูลปัจจุบัน
- ประกอบด้วยคำสั่ง UPDATE ในเคอร์เซอร์เพื่อล็อคข้อมูลแถวแรก
- มีคำสั่ง WHERE CURRENT OF เพื่ออ้างอิงแถวข้อมูลปัจจุบันจากเคอร์เซอร์

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

WHERE CURRENT OF CLAUSE

```
DECLARE
CURSOR sal_coursor IS
        SELECT e.department_id, employee_id, last_name, salary
        FROM employee e, department d
        WHERE d.department_id = e.department_id and d.department_id = 60
        FOR UPDATE OF salary
 BEGIN
      FOR emp_record IN sal_coursor Loop
        IF emp_record. salary < 5000 THEN
                UPDATE employee
                SET salary = emp_record.salary *1.10
                WHERE CURRENT OF sal_coursor;
        END IF
      END LOOP
END;
```

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;
...
```