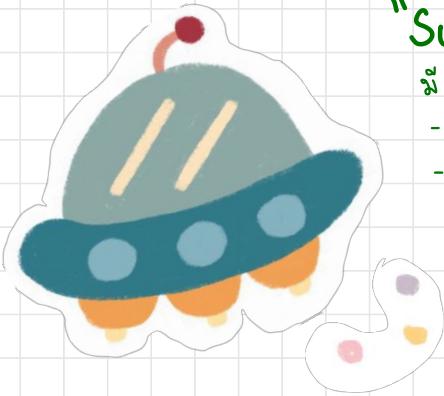


Lecture 2.

Anonymous	Procedure	Function
<ul style="list-style-type: none"> - ไม่มีชื่อ - ต้อง run ในส่วนที่ครุ่ง - ไม่เก็บใน DB - ใช้ใน program อันใหม่ได้ - ไม่มีการ return value (ต้อง return ให้ bind var. ต่างๆ) - ไม่มี parameter 	<ul style="list-style-type: none"> - มีชื่อ Block - Complie code ก่อนเดิม เรียกใช้ได้เลย - เก็บใน DB - สามารถเรียกใช้ใน program อื่นได้ - ไม่ต้อง return value แต่สามารถใช้ mode [OUT] ล่วงค่าออกมานิ่ง - ไม่มี parameter 	<ul style="list-style-type: none"> - มีชื่อ Block - Complie code ก่อนเดิม เรียกใช้ได้เลย - เก็บใน DB - สามารถเรียกใช้ใน program อื่นได้ - มีการ return value - ต้อง parameter



"Subprogram"

ลักษณะ

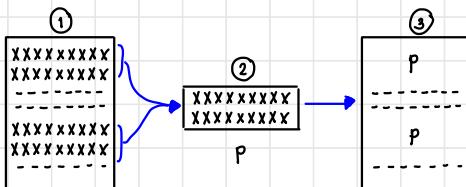
- PROCEDURE
- FUNCTION

- PL/SQL block มีชื่อ
- Complie แล้วรีเซ็ต
- เก็บ subprogram ลง DB ของ User นั้นๆ
- สามารถเรียกใช้งานใน program อื่นได้
- ยังคงชื่อ subprogram
- มีการ return value ออกมานิ่ง (กรณี FUNCTION)
หรือ OUT mode เพื่อส่งค่าตอบกลับ (PROCEDURE)
- กำหนด parameter ได้

Subprogram Design

- ควรแยกออกเป็น module เล็กๆ และแบ่งเข้า Layer

① Modularized : แยกส่วนที่ใช้ร่วมกันออกมารวบรวมเป็น method



② Layered Subprogram : ควรทยอยออกเป็น 2 layer ผู้ดูแล manage code ได้ง่ายขึ้น

1. Data access : นำเข้าต่อ DB (ลักษณะ SQL)
2. Business logic : การทำงานต่อๆ

► Subprograms ?

- គែង block ដែលទាន់រក្សាទុកដំណឹង DB សមារមធ្វើការនៅលើ program តូចតាមតម្លៃ
ឱ្យរើរក្សា store procedure / store function / store package
- Subprogram ប្រភពរបស់យើង
 - ① Specification : ខ្លួន procedure/function + parameter ទាំងអស់
 - ② Body : សោរណ៍ការ execute
- Subprogram មិន គឺជាការ

 - ① PROCEDURE : ដំណឹងការខ្លួន action
 - ② FUNCTION : ដំណឹង return តម្លៃរបស់វា

- ក្រោមផែនក្រោម
 - Easy maintenance : ទូរសព្ទ & reuse
 - Improved data security and integrity : ការកំណត់សកម្មដោយខ្លួនឯងក្នុង User
 - Improved performance
 - Improved code clarity : កើតឡើងយ៉ាង clear

► Parameter

- មិន គឺជាការ

① Formal parameter : ពេលវេលា Method ដែលត្រូវបានបញ្ជាក់ថា តើមានចំណាំណែនាំ និងតម្លៃ

ex CREATE PROCEDURE raise_sal (p-id IN NUMBER, p-sal IN NUMBER) IS ... END;

② Actual parameter : ពេលវេលាដែលបានបញ្ជាក់ថា តើមានចំណាំណែនាំដោយខ្លួនឯង

ex raise_sal(100, 2000);

- ការសំណើ parameter មិន បានបញ្ជាក់ថា តើមានចំណាំណែនាំ និងតម្លៃ

① Positional : សំណើតាមចំណាំណែនាំ / តាមលំដាប់ទៅរីប

② Named : សំណើ parameter តាមចំណាំណែនាំ formal parameter តាមការត្រួតពិនិត្យ associate (\Rightarrow)

③ Mixed : សម្រាប់រាយការ position + name គោរពទាន់រំនៀកទៅកាន់ការសំណើតាមចំណាំណែនាំ

↳ ! ចំណាំណែនាំ

ចំណាំណែនាំត្រូវបានបញ្ជាក់ថា តើមានចំណាំណែនាំ និងតម្លៃ

ex PROCEDURE demo(a NUMBER, b NUMBER, c NUMBER)

↳ EXECUTE demo(5, 7, b=9);

* ចំណាំណែនាំ b ត្រូវបានបញ្ជាក់ថា តើមានចំណាំណែនាំ និងតម្លៃ

នៅក្នុងការបញ្ជាក់ថា b = 9 (b=9)

{ :: ERROR

▶ Procedure : Method ที่ไม่มีการ return ค่า, มี parameter 3 mode

```
CREATE OR REPLACE PROCEDURE procedure-name (parameter1 [MODE] datatype1,  
parameter2 [MODE] datatype2)
```

) IS

(นูก local variable ต่างๆ ตาม Declare)

BEGIN

:

END procedure-name ;

* Datatype ของ parameter ไม่ต้องระบุขนาด

* Parameter Mode

① IN : ค่าที่ส่งเข้าไปน้ำมันเป็นอันเปลว (DEFAULT) น้ำมัน assign ค่า, set ค่า ให้

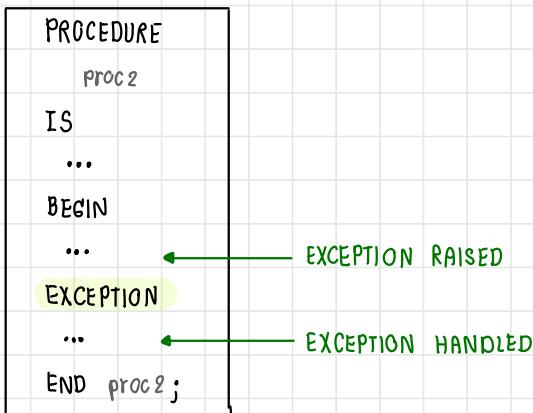
② OUT : ตัวแปรที่ส่งเข้ามาเพื่อรับค่า ส่งกลับไป

③ IN OUT : ส่งค่าไป สามารถเปลี่ยนค่าໄอ์ และส่งค่านั้นออกยังไงแบบเดิม

* 1 schema ห้ามมีชื่อซ้ำกัน

ต้องยกสวิงชื่อซ้ำตัวใด้ใน package และห้าม Overloading

* สามารถล็อก EXCEPTION ได้ (ใช้ในส่วน BEGIN)



Note

- Formal Parameter : Local variables declared in the parameter list of a subprogram specification.
- Actual Parameter : Literal values, variables, and expressions used in the parameter list of the calling subprogram.

Lecture 3.

► Function

```
CREATE OR REPLACE FUNCTION function_name (
    parameter1 [mode] datatype, ..
) RETURN datatype
IS
    ( မາក local variable တဲ့ အတောက် Declare)
BEGIN
    :
    RETURN expression;
END function_name ;
```

- * function ဆာမရစိနိုင်ခဲ့တဲ့ SQL Statement ၏
မှတ်ယူတဲ့ return datatype ဖော်ပြန် SQL
(မှာက record, index of, Boolean စားစိတ် SQL မျိုး)

- * mode မှာ function ဆိုတဲ့ IN
- * တော်သွေ့ return ချောင်းနေစွာ ၁ တော်
ကားဘဲလဲတဲ့ return ဆောင်ရွက်သွေ့ datatype
အောင်ကိုပဲပြောက်စွာ
- * เพော် return မျိုးခေါ်ခြင်း၊
လျှော်စွဲယူလော်တဲ့ return အဲမျိုးဘာ

- fn ဆာမရစိနိုင်ခဲ့တဲ့
- 1) bind variable
 - 2) anonymous block
 - 3) ၃ဗ္ဗ EXECUTE မောင်များ
 - 4) ၃ဗ္ဗရော်ကို SQL statement

▶ ចំណាំកាត់ទូទៅ function

① ជាដែនាំ SELECT នៃ SQL ដែលបានរួមក្នុង fn ក្នុងការការា DML (INSERT, UPDATE, DELETE) ໄមតែ

ex `SELECT EMPLOYEE-ID, GET-MAX(JOB-ID)`
FROM EMPLOYEES;
CREATE OR REPLACE FUNCTION GET-MAX(p-job-id VARCHAR2(200))
RETURN NUMBER IS
BEGIN
 UPDATE ... ;
END;

* fn GET-MAX គឺជានៃ SQL ໄមតែ
គឺជានៃ PL/SQL ទេ

② ជាដែនាំ UPDATE / DELETE ដែលបានរួមក្នុង fn ក្នុងការ SELECT
និងការការា DML នៃការការាប័ណ្ណការ ໄមតែ

ex `UPDATE employees SET salary = query-call-sql(100)`
WHERE employee-id = 170;
CREATE OR REPLACE FUNCTION query-call-sql(p-a NUMBER) RETURN NUMBER IS
V-S NUMBER;
BEGIN
 SELECT salary INTO V-S FROM EMPLOYEES WHERE employee-id = 170;
 RETURN (V-S + p-a);
END;

ex `UPDATE EMPLOYEES SET salary = dml-call-sql(2000)`

ក្នុង DML នៃ
ការការាប័ណ្ណការ
"EMPLOYEES"
WHERE employee-id = 170;
CREATE OR REPLACE FUNCTION dml-call-sql(p-sal NUMBER) RETURN NUMBER IS
BEGIN
 INSERT INTO EMPLOYEES (...) values;
 RETURN (p-sal + 100);
END;

③ បានរួមក្នុង fn តាម, ក្នុងការរួមក្នុង COMMIT / ROLLBACK / ងាយការសំឡែង DDL នៃ SQL ក្នុងការការា Transaction ក្នុងការការាប័ណ្ណការ

Lecture 4+5

► Package

- เป็น object ตั้งหนึ่ง ๆ table, schema, procedure แต่เป็นการ group object ต่าง ๆ เกี่ยวกับกัน เข้าไว้ด้วยกัน

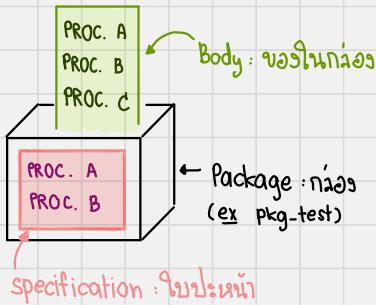
* ชื่อ package ไม่ใช่ schema ห้ามซ้ำกัน

- Package ประกอบด้วย

* ① Specification (Public) : เป็นไปปะหน้ากล่อง / ของ header เก็บไว้บอกเจ้ายังไง
มีตัวแปรอะไรบ้าง, fn, procedure ต่าง ๆ

* สิ่งที่อยู่ใน spec เป็นส่วนที่ USER สามารถเรียกใช้ได้

② Body (Private) : ส่วนที่เขียน code ของรากหรืองานที่ไม่ต้อง fn, procedure
ที่ประกาศไว้ใน spec.



วิธีใช้

EXECUTE PKG_TEST. PROC A ✓

EXECUTE PKG_TEST. PROC B ✓

EXECUTE PKG_TEST. PROC C X

เรียกใช้ PROC C ไม่ได้

เพราะไม่ได้ประกาศใน spec (public)

User ไม่สามารถเรียกใช้ได้

► การสร้าง Package : แบ่ง 2 ส่วน → spec., body

CREATE OR REPLACE PACKAGE package_name IS

(ตัวแปร / PROC. / FN ที่จะประกาศไว้เป็น public type)

END package_name;

CREATE OR REPLACE PACKAGE BODY package_name IS

(ตัวแปร / PROC. / FN ที่จะประกาศไว้เป็น private type)

(PROC/FN ที่ประกาศใน spec ทั้งหมด)

[BEGIN ตามด้วย statement ที่จะให้ initial]

END package_name;

① Specification
(Public Type)

② Body
(Private Type)

▶ การทำงานของ Package

- OVERLOADING

เป็นการสร้าง PROC / FN ที่มีชื่อเดียวกันได้ แต่! parameter ต้องต่างกัน

ต่างจำนวน
ต่าง data type
ต่างตำแหน่ง

มี param 3 ตัว

```
CREATE OR REPLACE PACKAGE dept_pkg IS
  PROCEDURE add_department
    (p_deptno departments.department_id%TYPE,
     p_name departments.department_name%TYPE := 'unknown',
     p_loc   departments.location_id%TYPE := 1700);

  PROCEDURE add_department
    (p_name departments.department_name%TYPE := 'unknown',
     p_loc   departments.location_id%TYPE := 1700);
END dept_pkg;
/
```

มี param 2 ตัว

* ex PROCEDURE test (pfirstname VARCHAR2(200)); } X สร้างไม่ได้ เพราะ มี param
PROCEDURE test (plastname VARCHAR2(200)); กี่จำนวนเท่ากัน และ data type
เดียวกัน

* ถ้าใน PACKAGE มีชื่อ PROC / FN ซ้ำกัน fn ที่ oracle ให้ (build-in function)
แล้วอย่างเดียว fn ของ oracle นี้ใน package → ต้องใช้ STANDARD.TO_CHAR(...)

- A package named STANDARD defines the PL/SQL environment and built-in functions.
- Most built-in functions are overloaded. An example is the TO_CHAR function:

```
FUNCTION TO_CHAR (p1 DATE) RETURN VARCHAR2;
FUNCTION TO_CHAR (p2 NUMBER) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 DATE, P2 VARCHAR2) RETURN VARCHAR2;
FUNCTION TO_CHAR (p1 NUMBER, P2 VARCHAR2) RETURN
  VARCHAR2;
. . .
```

- FORWARD DECLARATIONS

ในกรณี package ต้องเรียกชื่อ PROC/FN ตามตัวอักษร (A-Z) ท้าให้มีการ Method ขึ้น เรียกใช้ method ล่าสุด จะเรียกใช้ไม่ได้ ต้องทำ FORWARD DECLARATION

```
CREATE OR REPLACE PACKAGE BODY forward_pkg IS
  PROCEDURE calc_rating (...) ; -- forward declaration
  -- Subprograms defined in alphabetical order
  PROCEDURE award_bonus(...) IS
  BEGIN
    calc_rating (...); -- reference resolved!
    .
    .
    .
  END;
END forward_pkg;
```

- * เป็นการบอกว่า มี method นี้อยู่ สามารถเรียกใช้ได้ เมื่อการเรียกวะได้ ไม่ต้องอักษร (A-Z)
- * forward declaration จะเขียนคล้ายใน spec คือไม่ต้องเปลี่ยนการเขียน

- Initialize

- เป็นชนิด BEGIN ของ package (อยู่ล่างสุดของ block) จะเรียกไว้ไม่เรียกไว้ default เป็น NULL
- มีหน้าทำการ RUN เป็นครั้งแรกของ package (first time a component in a package is referenced)
- ex. ใช้ในการ assign ค่าสักอย่างที่ต้องใช้ใน package,
ex. ใช้ลบข้อมูลใน table ทั้งหมดก่อนนำไปใช้ package

```
CREATE OR REPLACE PACKAGE taxes IS
  v_tax NUMBER;
  . . . -- declare all public procedures/functions
END taxes;
/
CREATE OR REPLACE PACKAGE BODY taxes IS
  . . . -- declare all private variables
  . . . -- define public/private procedures/functions
BEGIN
  SELECT rate_value INTO v_tax
  FROM tax_rates
  WHERE rate_name = 'TAX';
END taxes;
/
```

} ไปเก็บค่าใน DB
แล้ว assign ค่าให้
กับตัวแปร v_tax

Lecture 6.

► Package ใน Oracle ที่มีให้ (เน้น 3 ตัว)

- DBMS_OUTPUT

• ไว้แสดงผลของข้อมูล ใช้ในการตรวจสอบ/debug ช่องทาง

- มี method ให้ใช้ 5 ตัว

① PUT : เติม MSG พื้น Buffer แบบต่อจากของเดิม

② PUT_LINE : ขึ้นบรรทัดใหม่ในนี้และ เติม msg เข้า buffer ≈ put + new-line

③ NEW_LINE : ขึ้นบรรทัดใหม่ เฉยๆ

④ GET_LINE : อ่านข้อมูลใน buffer ที่ลับบรรทัด (ระหว่างค่า)

⑤ GET_LINES : อ่านข้อมูลใน buffer ทั้งหมด

* msg ที่ส่งเข้าไปจะยังไม่ส่องออกมา จนกว่า program จะทำงานเสร็จ.

- DEFAULT ผันจะ disable อยู่ → ต้องรัน enable ต่อหนึ่งจะใช้งานได้

SET SERVEROUTPUT ON [size n]

↳ default ผัน = 2000 แต่ถ้าในข้อมูลใน buffer มีมากเกิน 2000

จะ error ว่า overflow buffer ที่ไหนต้องก้านลดขนาด

[2,000 - 1,000,000]

- UTL_FILE

• ไว้เขียน/อ่านไฟล์ที่เป็น text → โหลด file ที่เขียนจะเก็บอยู่บน DB server

• ปลด USER ธรรมชาติจะไม่ได้ ต้องให้ Admin (DBA) GRANT สิทธิ์ให้ต้องใช้ไฟล์ (DBA จะสร้าง folder ไว้ใน)

- method ที่ใช้ ≈ DBMS_OUTPUT

• การใช้งาน UTL_FILE

① ปิด file > FOPEN

② ทำการบุญการ อ่าน/เขียน จาก file

↳ ex PUT, PULLINE, NEW_LINE,

GET_LINE, GET_LINES

③ เหตุการณ์อะไรบ้าง?

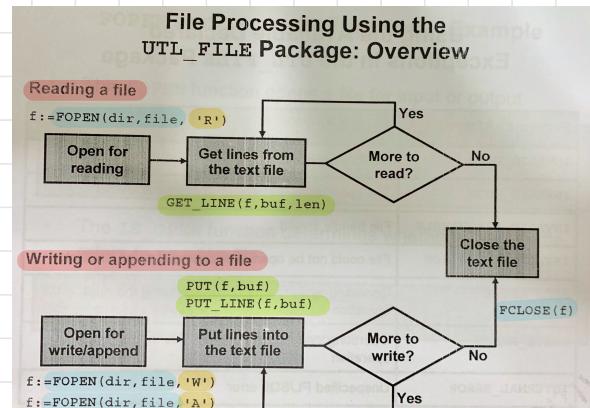
④ ปิด file > FCLOSE

* mode ของการใช้ file

'R' : READ > ใช้อ่าน

'W' : WRITE > ถ้ามี file อยู่แล้ว จะเขียนทั้ง

'A' > ถ้ามี file อยู่แล้ว จะเพิ่มต่อจากเดิม



● ຕົວຢ່າງການໃຊ້ UTL_FILE (Write)

```

CREATE OR REPLACE PROCEDURE sal_status(
    p_dir IN VARCHAR2, p_filename IN VARCHAR2) IS
    f_file UTL_FILE.FILE_TYPE;
    CURSOR cur_emp IS
        SELECT last_name, salary, department_id
        FROM employees ORDER BY department_id;
    v_newdeptno employees.department_id%TYPE;
    v_olddeptno employees.department_id%TYPE := 0;
BEGIN
    f_file := UTL_FILE.FOPEN(p_dir, p_filename, 'W');
    UTL_FILE.PUT_LINE(f_file,
    'REPORT: GENERATED ON ' || SYSDATE);
    UTL_FILE.NEW_LINE(f_file);
    ...

```

```

FOR emp_rec IN cur_emp LOOP
    IF emp_rec.department_id <> v_olddeptno THEN
        UTL_FILE.PUT_LINE(f_file,
        'DEPARTMENT: ' || emp_rec.department_id);
        UTL_FILE.NEW_LINE(f_file);
    END IF;
    UTL_FILE.PUT_LINE(f_file,
        'EMPLOYEE: ' || emp_rec.last_name ||
        ' earns: ' || emp_rec.salary);
    v_olddeptno := emp_rec.department_id;
    UTL_FILE.NEW_LINE(f_file);
END LOOP;
UTL_FILE.PUT_LINE(f_file, '*** END OF REPORT ***');
UTL_FILE.FCLOSE(f_file);
EXCEPTION
    WHEN UTL_FILE.INVALID_FILEHANDLE THEN
        RAISE APPLICATION_ERROR(-20001, 'Invalid File.');
    WHEN UTL_FILE.WRITE_ERROR THEN
        RAISE APPLICATION_ERROR (-20002, 'Unable to write to file');
END sal_status;

```

→ FOPEN: ຜົນຂໍ: return ເປັນ UTL_FILE.FILE_TYPE
 ຕົວມີຕົວແປງຈາກປ່ອງທີ່ຈະ type ເລືຍກັນ ເພື່ອເວົາ
 ຕົວແປງນີ້ເປັນຕົວຮັບຂູ້ທຳ file ໃຫນ.

→ ສາມາດໃຊ້ method ≈ DBMS_OUTPUT
 ເພື່ອວ່ານາເຊື່ອນ file ໄດ້ ໄດຍຕ້ອງຮະບູກ
 file ໃຫນ, ຂີ msg ຂະໜາ

→ ເປັນການ loop ຂອງ CURSOR

ກະເວົາ ຂ່ອມູນ employee ແຕ່ກີ: recode
 ອີສິນ file

→ ປຶດ file

● ຕົວຢ່າງການໃຊ້ UTL_FILE (Read)

```

CREATE OR REPLACE PROCEDURE read_file(dir VARCHAR2, filename VARCHAR2) IS
    f_file UTL_FILE.FILE_TYPE;
    buffer VARCHAR2(200);
    lines PLS_INTEGER := 0;
BEGIN
    DBMS_OUTPUT.PUT_LINE(' Start ');
    IF NOT UTL_FILE.IS_OPEN(f_file) THEN
        DBMS_OUTPUT.PUT_LINE(' Open ');
        f_file := UTL_FILE.FOPEN(dir, filename, 'R');
        DBMS_OUTPUT.PUT_LINE(' Opened ');
    BEGIN
        LOOP
            UTL_FILE.GET_LINE(f_file, buffer);
            lines := lines + 1;
            DBMS_OUTPUT.PUT_LINE(TO_CHAR(lines, '099') || ' ' || buffer);
        END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE(' ** End of File **');
    END;
    DBMS_OUTPUT.PUT_LINE(lines || ' lines read from file');
    UTL_FILE.FCLOSE(f_file);
    END IF;
END read_file;
/

```

● ການໃຊ້ງານ

EXECUTE sal_status ('UTL_FILE', 'SAL_STATUSXXX.TXT'); ▶ Writing file
 EXECUTE read_file ('UTL_FILE', 'SAL_STATUSXXX.TXT'); ▶ reading file
 path ↗ ↗
 ↗ ↗

- UTL_MAIL

- ใช้สำหรับส่งเมล (ข้อมูลจาก db) ไม่ต้องสร้าง App ชั้นนำเพื่อส่งเมล
- มี 3 method (ผ่าน PROCEDURE)
 - SEND : ส่งข้อความธรรมดา
 - SEND_ATTACH_RAW : ส่งข้อความ + แนบไฟล์ได้ (ฟอร์mat image, video..)
 - SEND_ATTACH_VARCHAR2 : ส่งข้อความได้ + แนบไฟล์แบบ text ที่ยาวๆ ได้
- จะต้อง Package นี้ได้ต้อง install ลง DB Server
และ set ค่า SMTP_OUT_SEVER (เพื่อของกันใน mail ส่งจาก server อะไร)

```
BEGIN
    UTL_MAIL.SEND('otn@oracle.com', 'user@oracle.com',
        message => 'For latest downloads visit OTN',
        subject => 'OTN Newsletter');
END;
```

```
CREATE OR REPLACE PROCEDURE send_mail_logo IS
BEGIN
    UTL_MAIL.SEND_ATTACH_RAW(
        sender => 'me@oracle.com',
        recipients => 'you@somewhere.net',
        message =>
            '<HTML><BODY>See attachment</BODY></HTML>',
        subject => 'Oracle Logo',
        mime_type => 'text/html'
    );
    attachment => get_image('oracle.gif'),
    att_inline => true,
    att_mime_type => 'image/gif',
    att_filename => 'oralogo.gif');
END;
/
```

mime_type : สามารถกำหนดให้ว่าส่ง msg เป็นแบบไหน
default = 'text/plain'

Lecture 7. (Dynamic SQL)

- Dynamic SQL

- SQL statement ที่ครอบคลุม single quote (' ... ') : คำสั่งใช้ในมุต DML, DDL, DCL, session-control
- ขั้นตอนการทำ
 - Parse : ตรวจสอบค่าสั่ง syntax, constant ต่างๆ
 - Bind : เอาตัวแปรมาถูกกับ dynamic SQL (อ่านตัวแปรจากไฟล์ dynamic SQL)
 - Execute : เอาค่าสั่งที่เตรียมไว้ run บน server
 - Fetch : นำค่าสั่งไป query (SELECT) เพื่อดึงลิ๊งค์ select return ออกมายัง record
- Dynamic SQL จะทำขั้นตอนนี้กัน ตอน RUN TIME เมรา dynamic SQL จะมองเป็น string ตอน compile ยังไม่ถูก execute แล้วเมื่อตอน run time
- * Embedded SQL : คำสั่ง SQL statement ปักต์ ในรูปแบบ ' ... ' ครอบจะทำขั้นตอน parse กับ bind ตอน COMPILE TIME

ex : SELECT * FROM EMPLOYEES ► dynamic SQL

SELECT * FROM EMPLOYEES ► embedded SQL

- Working with Dynamic SQL : การที่ini SQL → Dynamic SQL ฝั่ง 2 วิธี

ex SELECT * FROM EMPLOYEES

WHERE JOB-ID = 'ST_MAX' and salary > 3000

```
'SELECT * FROM EMPLOYEES  
WHERE JOB-ID = '|| v_jobid || '' and salary > '|| v_sal  
concat value ที่ datatype เป็น string  
จะต้องมี single quote คู่ๆ'
```

{ 1) Concat

```
'SELECT * FROM EMPLOYEES
```

```
WHERE JOB-ID = :z and salary > :a'
```

{ 2) Placeholder หรือเรียกว่า host variable

ใช้ช่องให้กับ ค่าเป็น string มันจะ
เพิ่ม single quote ให้ (แปลงให้)
* มี datatype ให้เปลี่ยนเป็น

- จะ EXECUTE Dynamic SQL ได้ 2 แบบ
 - Native Dynamic SQL (NDS)
 - DBMS_SQL

▶ กิจ EXECUTE dynamic SQL

①- Native Dynamic SQL

- จะค่าสั่ง

```
EXECUTE IMMEDIATE string-dynamic
```

[INTO www] : ใช้เมื่อ dynamic SQL เป็นค่าสั่ง SELECT
(เพื่อเอาค่าที่ไป select มาเก็บไว้ตัวแปร www)

[USING [IN/OUT] / IN OUT] www : ใช้เมื่อ dynamic SQL
มี placeholder (host variable)

ex DECLARE

```
  V_Stmt VARCHAR2(255);  
  :
```

BEGIN

non-query without var. | V_Stmt := 'CREATE TABLE TEST (id NUMBER, name VARCHAR2(200))';
 | EXECUTE IMMEDIATE v_stmt ; ไม่ต้องมีทั้ง INTO และ USING

non-query with variable { V_Stmt := 'CREATE TABLE :table (:col_spec)';
 | EXECUTE IMMEDIATE v_stmt USING v_table, v_col;
 | # ถ้า placeholder [:table, :col_spec] ร่วมกับการ binding ▷ ใช้ USING (ตามล่าสุด)

Single row Query

V_Stmt := 'SELECT * FROM EMPLOYEES WHERE EMPLOYEE_id :id'
EXECUTE IMMEDIATE v_stmt INTO v_record USING v_empid;
มีการ query [SELECT] ร่วมกับ INTO และมีการใช้ placeholder ร่วมกับ USING
SELECT ① record

MUTIPLE ROW QUERY

V_Stmt := 'SELECT * FROM EMPLOYEES';
EXECUTE IMMEDIATE v_stmt BULK COLLECT INTO v_arr_rec;
ถ้ามีการ query ที่ return จำนวนมาก record ใช้ BULK COLLECT INTO
ตัวไปใช้ตัวแปรที่เป็น table of

CURSOR Dynamic SQL

▷ query ဆ่องนานหลาย record ใช้ cursor ref ได้รวมเอาค่าอ่องมา

• ពេញចិត្តការវិទ្យា CURSOR Dynamic

- Use OPEN-FOR, FETCH, and CLOSE processing:

```

CREATE PROCEDURE list_employees( p_deptid NUMBER ) IS
  TYPE emp_refcsr_type IS REF CURSOR;
  cur_emp emp_refcsr_type;
  rec_emp employees%ROWTYPE;
  v_stmt varchar2(200) := 'SELECT * FROM employees';
BEGIN
  IF p_deptid IS NULL THEN OPEN cur_emp FOR v_stmt;
  ELSE
    v_stmt := v_stmt || ' WHERE department_id = :id';
    OPEN cur_emp FOR v_stmt USING p_deptid;
  END IF;
  LOOP
    FETCH cur_emp INTO rec_emp;
    EXIT WHEN cur_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(rec_emp.department_id||' '|| rec_emp.last_name);
  END LOOP;
  CLOSE cur_emp;
END;

```

ចុងចម្លោះ

- សំនៀត type ref ទៅ CURSOR
- សំនៀតចំណែរតាម type ref cursor
- OPEN-FOR
- FETCH INTO
- CLOSE

នឹង USING អនុវត្ត :id

• Dynamic នាមរាលិទិន្នន័យក្នុង Anonymous block ។

```

CREATE FUNCTION annual_sal( p_emp_id NUMBER )
RETURN NUMBER IS
  v_plsql varchar2(200) := 
    'DECLARE ' ||
    '   rec_emp employees%ROWTYPE; ' ||
    'BEGIN ' ||
    '   rec_emp := get_emp(:empid); ' ||
    '   :res := rec_emp.salary * 12; ' ||
    'END;';
  v_result NUMBER;
BEGIN
  EXECUTE IMMEDIATE v_plsql
    USING IN p_emp_id, OUT v_result;
  RETURN v_result;
END;
/
EXECUTE DBMS_OUTPUT.PUT_LINE(annual_sal(100))

```

Anonymous Block

mode នៃ IN USING & PROCEDURE

IN : ស៊ីគោលការណ៍បញ្ជី

OUT : ស៊ីគោលការណ៍បញ្ជី

IN OUT : ឱ្យគោលការណ៍បញ្ជី ឬ ស៊ីគោលការណ៍បញ្ជី

* នឹងការងារតាមលំដាប់ទៅ placeholder

នឹង string dynamic

② - DBMS_SQL

- ใช้ในกรณีไม่รู้อะไรเลย ทั้งจำนวน column, datatype ลำดับตัวเลขต่างๆ,
จะต้องเขียนทุกหันตอนเมื่อ ซึ่งซ่อนกกว่า ยังย่นตามที่ต้องๆ)
- บันทุณการท่า (method ที่ใช้)
 - * 1. OPEN_CURSOR : เปิด
 - * 2. PARSE : สร้างรากของค่าสั่ง

≈ USING → 3. BIND_VARIABLE : ถ้ามีตัวแปร placeholder ต้องใช้ในproc binding

* 4. EXECUTE : ให้ run คำสั่ง

≈ INTO → 5. FETCH_ROWS : ถ้าคำสั่งเป็น SELECT จะใช้ FETCH_ROW หวานเจา

* 6. CLOSE_CURSOR : ปิด

- ตัวอย่าง

```
CREATE PROCEDURE insert_row (p_table_name VARCHAR2,
    p_id VARCHAR2, p_name VARCHAR2, p_region NUMBER) IS
    v_cur_id      INTEGER;
    v_stmt         VARCHAR2(200);
    v_rows_added  NUMBER;
BEGIN
    v_stmt := 'INSERT INTO '|| p_table_name |||
              ' VALUES (:cid, :cname, :rid)';
    v_cur_id := DBMS_SQL.OPEN_CURSOR;
    DBMS_SQLPARSE(v_cur_id, v_stmt, DBMS_SQL.NATIVE);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cid', p_id);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':cname', p_name);
    DBMS_SQL.BIND_VARIABLE(v_cur_id, ':rid', p_region);
    rows_added := DBMS_SQL.EXECUTE(v_cur_id);
    DBMS_SQLCLOSE_CURSOR(v_cur_id);
    DBMS_OUTPUT.PUT_LINE(v_rows_added|| ' row added');
END; /
```

OPEN_CURSOR : q: return
cursor id ที่เป็น INTEGER อีก個
เนื่องจากภาษาชุด DBMS_SQL
* ร่วมกับ cursor แบบร่วมกัน

} ใน stmt มี placeholder 3 ตัว
ก็ต้อง bind_variable 3 คำสั่ง
ผูกตามชื่อใน stmt แบบตัวเปลี่ยน

- EXECUTE ถ้าคำสั่งเป็น SELECT
จะไม่ผลลัพธ์ ไม่ต้องกล่าวถึงแบบ
มารับกันได้
- ถ้าคำสั่งเป็น INSERT/UPDATE/
DELETE ต้อง method EXECUTE
q: return จำนวน ROW ที่ทำการ