

# PL/SQL

FINAL

by Mym

# Lecture 2.

| Anonymous   | Procedure   | Function  |
|---|---|---|
| <ul style="list-style-type: none"> <li>- ไม่มีชื่อ</li> <li>- ต้อง run ในส่วนที่คั่ง</li> <li>- ไม่เก็บใน DB</li> <li>- ใช้ใน program อันใหม่ได้</li> <li>- ไม่มีการ return value<br/>(ต้อง return ให้ bind var. ต่างๆ)</li> <li>- ไม่มี parameter</li> </ul> | <ul style="list-style-type: none"> <li>- มีชื่อ Block</li> <li>- Complie code ก่อนเดิม เรียกใช้ได้เลย</li> <li>- เก็บใน DB</li> <li>- สามารถเวลาซึ่งไม่เรียกใช้ใน program อันได้</li> <li>- ไม่ต้อง return value แต่สามารถใช้ mode [OUT] ล่วงค่าออกมาได้</li> <li>- มี parameter</li> </ul> | <ul style="list-style-type: none"> <li>- มีชื่อ Block</li> <li>- Complie code ก่อนเดิม เรียกใช้ได้เลย</li> <li>- เก็บใน DB</li> <li>- สามารถเวลาซึ่งไม่เรียกใช้ใน program อันได้</li> <li>- มีการ return value</li> <li>- ลักษณะ parameter</li> </ul> |

## "Subprogram"

ลักษณะ

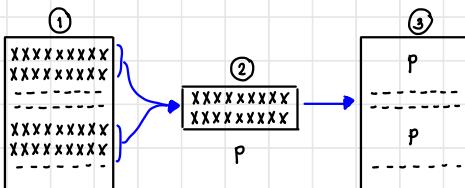
- PROCEDURE
- FUNCTION

- PL/SQL block 里面有
- Complie แล้วคั่งแล้ว
- เก็บ subprogram ลง DB ของ User นั้นๆ
- สามารถเรียกใช้งานใน program อันได้ ยังชื่อ subprogram
- มีการ return value ออกมานะ (กรณี FUNCTION)  
ให้ OUT mode เพื่อส่งค่าตอบกลับ (PROCEDURE)
- กำหนด parameter ได้

## ► Subprogram Design

- ควรแยกออกเป็น module เล็กๆ และ แบ่งเข้าใน Layer

① Modularized : แยกส่วนที่ใช้ร่วมกันออกมารวบรวมเป็น method



② Layered Subprogram : ควรทยอยออกเป็น 2 layer ผู้ดู管 code ได้ง่ายขึ้น

1. Data access : นำ回去ต่อ DB (ลักษณ์ SQL)
2. Business logic : การทำงานต่อๆ

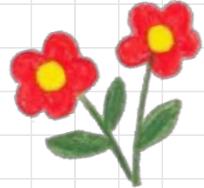
## ► Subprograms ?

- គឺជា block ដែលរារ៉ាងផ្សេងៗពីក្នុង DB សមារតាំងកិច្ចការណ៍នៃ program តើនឹងតើ  
ធ្វើឡើងរវាង store procedure / store function / store package
- Subprogram ប្រភេទខាងក្រោម
  - ① Specification : ខ្លួន procedure/function + parameter តាត់
  - ② Body : សោរណ៍ការ execute
- Subprogram មិន អយ៉ាំ
  - ① PROCEDURE : ដំណឹងការខាងក្រោម action
  - ② FUNCTION : ដំឡើង return តម្លៃទូទៅ
- វិវាទន៍
  - Easy maintenance : ទូរគម្យ & reuse
  - Improved data security and integrity : ការកំណត់សម្រាប់ខ្លួនខ្លួនមុនក្នុង User
  - Improved performance
  - Improved code clarity : ក្រុមក្រាម clear



## ► Parameter

- មិន ប្រភេទ



① Formal parameter : ពន្លាសំង Method ដើម្បីព័ត៌មាន, mode, datatype

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

② Actual parameter : ពន្លានើយកឱ្យ Method ដើម្បីសំងទៀតនៅលើក្រុង

ex raise\_sal( 100, 2000);

- ការសំង parameter មិន ឈើបាន

① Positional : សំងតាមចំណេះ/តាមលំដាប់ទៅរួម

② Named : សំង parameter តាមខ្លួនចំណេះថាដីជាប្រព័ន្ធដីជាប្រព័ន្ធ formal parameter ឬឯកសារនៃ associate ( $\Rightarrow$ )

③ Mixed : ឈើសម្រាប់រាយ position + name គោរពទៅរួមទៅក្នុងការសំងតាមចំណេះរាយក្នុងការសំង

↳ ! ចំណេះរាយ —

ចំណេះរាយក្នុង 2 គឺនឹងតើ (តាមលំដាប់ទៅរួម ឬឯកសារនៃក្រុង)

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

↳ EXECUTE demo( 5, 7, b=>9);

\* ចំណេះរាយ b តើត្រូវតើលើក្នុងក្រុងទៅរួមទៅក្នុងក្រុង

នៅក្នុង b ឈើសម្រាប់រាយក្នុង 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

( นิยม local variable ตั่ง, ไม่ต้อง Declare )

BEGIN

:

END procedure-name ;

\* Datatype ของ parameter ไม่ต้องระบุขนาด

## \* Parameter Mode

① IN : ค่าที่ส่งเข้าไปน้ำหนึ่งเดียวแล้ว  
(DEFAULT) นำม assign ค่า, set ค่า ให้

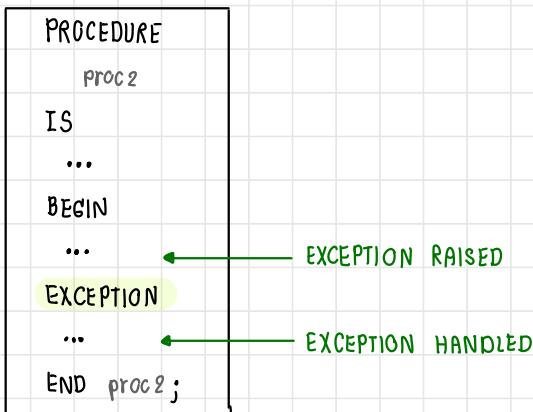
② OUT : ตัวเปลี่ยนที่ส่งเข้ามาเพื่อรับค่า  
แล้วกลับไป

③ IN OUT : ส่งค่าไป สามารถเปลี่ยนค่าໄอ์  
แล้วล่าค่านั้นออกกันต่างแปรเปลี่ยน

\* 1 schema ห้ามมีชื่อซ้ำกัน

ถ้าอยากรวบซึ่งชื่อตัวให้ใส่ใน package และทำ Overloading

\* สามารถล็อก EXCEPTION ได้ (ใช้ในร่วม 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
    ( မွေး local variable တဲ့ အတွက် Declare)
BEGIN
    :
    RETURN expression;
END function_name ;

```

- \* function ဆာမရစိနိုင်ခဲ့တဲ့ SQL Statement ၏  
မှတ်ယူတဲ့ return datatype ဖော် SQL  
(မွေး record, index of, Boolean ဒါးဘို့တဲ့ SQL မြို့ခဲ့)

- \* mode မှတ် function ဆိုတဲ့ IN
- \* တော်သံ return ချို့ယော်ခဲ့တဲ့  
ကိုဘို့လဲ return ဆိုခဲ့တဲ့ datatype  
အော်ယော်ပြုရန်ဖြစ်
- \* เพော် return မျိုးခေါ်ခြင်း အော်ဘို့  
ဆိုတဲ့ ပြုလုပ်တဲ့ return အော်ဘို့

- fn ဆာမရ ရှိရော်ကို
- 1) bind variable
  - 2) anonymous block
  - 3) ရှိ EXECUTE ဆောင်
  - 4) ရှိရော်ကို SQL statement

## ▶ ចុះចាក់ទុកទាំង function

① តារឹងនៃ SELECT នៃ SQL ដែលបានរួមទៅ fn ក្នុងការការណា DML (INSERT, UPDATE, DELETE)

ex `SELECT EMPLOYEE-ID, GET-MAX(JOB-ID)`  
FROM EMPLOYEES;

ໄង់

```
CREATE OR REPLACE FUNCTION GET-MAX(p-job-id VARCHAR2(200))  
RETURN NUMBER IS  
BEGIN  
    UPDATE ... ;  
END;
```

\* fn GET-MAX គឺជានេះ SQL ໄង់ទៅ  
គិតឱ្យនេះ PL/SQL នៅ

② តារឹងនៃ UPDATE / DELETE ដែលបានរួមទៅ fn ក្នុងការ SELECT

នូវការការណា DML នៃការការណ៍ឡើងក្នុង ໄង់

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)`

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;

នូវ DML នេះ

ក្នុងការការណ៍ឡើងក្នុង

"EMPLOYEES"

③ បានរួមទៅ fn តារា, ក្នុងការការណ៍ commit / rollback / ងារការសំឡុង DDL នៃ SQL ក្នុងការការណា Transaction ក្នុងការការណា

# Lecture 4+5



## ► Package

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

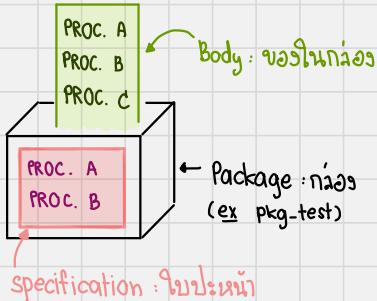
\* ชื่อ package ไม่ใช่ schema ห้ามซ้ำกัน

- Package จะประกอบด้วย

\* ① Specification (Public) : เป็นไปปะหน้ากล่อง / ของ header เก็บไว้บอกเจย์ว่า มีตัวแปรอะไรอย่าง, 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 : ขั้นตอน → spec., body

CREATE OR REPLACE PACKAGE package\_name IS

(ตัวแปร / PROC. / FN ที่จะประกาศให้เป็น public type)

END package\_name;

① Specification  
(Public Type)

CREATE OR REPLACE PACKAGE BODY package\_name IS

(ตัวแปร / PROC. / FN ที่จะประกาศให้เป็น private type)

(PROC/FN ที่ประกาศใน spec ทั้งหมด)

[BEGIN ตามด้วย statement ที่จะให้ initial ]

END package\_name;

② Body  
(Private Type)

# ▶ การทำงานของ Package

## - OVERLOADING

เป็นการสร้าง PROC / FN ที่มีชื่อเดียวกันได้ แต่! parameter ต้องต่างกัน

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);
```

มี param 2 ตัว ~

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

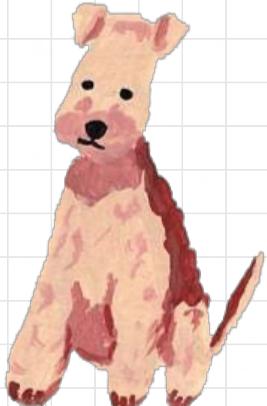
ต่างจำนวน  
ต่าง data type  
ต่างตำแหน่ง

\* ex PROCEDURE test (p\_firstname VARCHAR2(200)); | X สร้างไม่ได้ เมื่อ: มี param  
 PROCEDURE test (p\_lastname VARCHAR2(200)); } ที่จำนวนเท่ากัน และ data type  
 เดียวกัน

\* ถ้าใน PACKAGE มีชื่อ PROC / FN ซ้ำกัน fn ที่ oracle ให้ (build-in function)  
 แล้วอย่างเดียว fn ของ oracle จะเป็น 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;
```

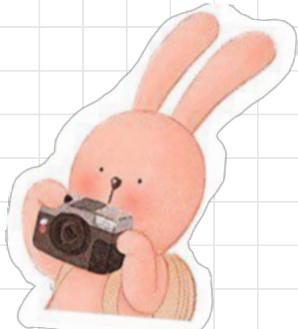
- \* เป็นการระบุก่อนว่า มี method ชื่ออย่างไร สามารถเรียกใช้ได้ เมื่อการเรียกว่าจะได้ ไม่ต้องอ่าน (A-Z)
- \* forward declaration จะเขียนคล้ายใน spec คือไม่ต้องเปลี่ยนการเขียน

## - Initialize

- เป็นชนิด BEGIN ของ package (อยู่ล่างสุดของ block) จะเขียนก็ได้ ไม่เขียนก็ได้ default เป็น NULL
- วัสดุทำการ RUN เป็นครั้งแรกของ package (first time a component in a package is referenced)
- ex. ใช้ในการ assign ค่าสักอย่างที่ต้องใช้ใน package,  
ex. ใช้ลบข้อมูลใน table ที่นั่นจะดีกว่าใน 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 ใหม่ Oracle ที่มีให้ (เน้น 3 ตัว)

### - DBMS\_OUTPUT

- ไว้แสดงผลของข้อมูล ใช้ในการตรวจสอบ/debug ช่องจูบ
- มี method ใหม่เพิ่มลักษณะ 5 ตัว

① PUT : เติม MSG พื้น Buffer แบบต่อจากของเดิม

② PUT\_LINE : ขึ้นบรรทัดใหม่ในนี้และ เติม msg เข้า buffer คือ put + new-line

③ NEW\_LINE : ขึ้นบรรทัดใหม่ เฉยๆ

④ GET\_LINE : อ่านข้อมูลใน buffer ที่ลับบรรทัด (ระบบ.ยາ)

⑤ GET\_LINES : อ่านข้อมูลใน buffer ที่ร่วมกัน

\* msg ที่ส่งเข้าไปจะยังไม่ส่องออกมา จนกว่า program จะทำงานเสร็จ.

- DEFAULT ผู้คนจะ disable อยู่ → ต้องรีส์ทใน enable ต่อหนึ่งวันใช้งานได้

SET SERVEROUTPUT ON [size n]

↳ default คือ = 20000 แต่ถ้าในข้อมูลใน buffer มีมากเกิน 20000

จะ error ว่า overflow buffer ที่เกินต้องการขนาดของ buffer

[2,000 - 1,000,000]

### - UTL\_FILE

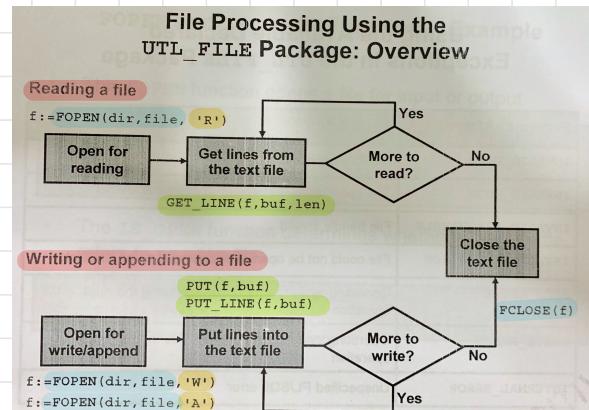
- ไว้เขียน/อ่านไฟล์ที่เป็น text → โหลด file ที่เขียนจะเก็บอยู่บน DB server
- ปลด USER ธรรมชาติจะไม่ได้ ต้องให้ Admin (DBA) GRANT สิทธิ์ให้ตั้งแต่ไฟล์ (DBA จะสร้าง folder ไว้ใน)
- method ที่ใช้ ≈ DBMS\_OUTPUT
- การใช้งาน UTL\_FILE
  - ① ปิด file > FOPEN
  - ② ทำการบุญการ อ่าน/เขียน จาก file
    - ↳ ex PUT, PULLINE, NEW\_LINE,  
GET\_LINE, GET\_LINES
  - ③ เหตุว่ามีอ่านไหม?
  - ④ ปิด file > FCLOSE

\* mode ของการใช้ file

'R' : READ > ใช้อ่าน

'W' : WRITE > ถ้ามี file อยู่แล้ว จะเขียนทั้งหมด

'A' > ถ้ามี file อยู่แล้ว จะเพิ่มต่อจากเดิม



## ● ចំណាំការងារ 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: មិនត្រូវ return ដែល UTL\_FILE.FILE\_TYPE  
ត្រូវម៉ោងបញ្ជាក់ថា type ត្រូវការពើខ្លួន  
ត្រូវបញ្ជាក់ថា file នេះ

សារធានាឌី method ≈ DBMS\_OUTPUT  
ដើម្បីរាយការណ៍ file ទៅ ត្រូវបានបញ្ជាក់  
file នេះ, និង msg នេះ

បង្ហាញការ loop ទៅ CURSOR  
រាយការណ៍ employee ត្រូវការពើ recode  
នៃស័ណិត file

→ ជួយ file

## ● ចំណាំការងារ 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
        f_file := UTL_FILE.FOPEN(dir, filename, 'R');
        DBMS_OUTPUT.PUT_LINE(' Open ');
    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;
/

```



## ● ការងារ

EXECUTE sal\_status ('UTL\_FILE', 'SAL\_STATUSXXX.TXT'); ▶ Writing file  
 EXECUTE read\_file ('UTL\_FILE', 'SAL\_STATUSXXX.TXT'); ▶ reading file  
 path file

## - UTL\_MAIL

- ใช้สำหรับส่งเมล (ข้อมูลจาก db) ไม่ต้องสร้าง App ชั้นนำเพื่อส่งเมล
- มี 3 method (ผ่าน PROCEDURE)
  - ① SEND : ส่งข้อความธรรมดา
  - ② SEND\_ATTACH\_RAW : ส่งข้อความได้ + แนบไฟล์ได้ (ฟอร์mat image, video..)
  - ③ SEND\_ATTACH\_VARCHAR2 : ส่งข้อความได้ + แนบไฟล์แบบ text ที่ยาวๆ ได้
- จะต้อง Package นี้ได้ต้อง install ลง DB Server  
และ set ค่า SMTP\_OUT\_SERVER (เพื่อของกันใน mail ส่งจาก server อะไร)

```

    ↗ ผู้รับ
    ↗ ผู้ส่ง
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 : สามารถกำหนดได้ว่าส่ง 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) เพื่อดึงลิสต์ select return ออกมาก็叫 record
- Dynamic SQL จะทำขั้นตอนนี้กัน ตอน RUN TIME เมรา dynamic SQL จะมองเป็น string ตอน compile ยังไม่ถูก execute ให้มีผลตอน run time
- Embedded SQL : คำสั่ง SQL statement ปักต์ ในรูปแบบ ' ... ' ครอบ
 

จะทำขั้นตอน 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

ใช้ช่อง空位 คือเป็น 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

▪ ใช้คำสั่ง

`EXECUTE IMMEDIATE string-dynamic`

[ INTO `variable` ] : ใช้เมื่อ dynamic SQL เป็นค่าสั่ง SELECT  
(เพื่อเวลาต้อง select งานเก็บไว้ตัวแปร `variable`)

[ USING [IN/OUT] / IN OUT] `variable` : 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] จึงต้องมี INTO และมีการให้ placeholder จึงต้องใช้ 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  
ตัว变量เป็น table of

**CURSOR Dynamic SQL**

▶ query ဆ่องขนาด record ที่ cursor ref ได้รับมาจาก

- ผู้อ่านการใช้ 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;
```

หน้า 16

- สร้าง type ref ของ CURSOR
- สร้างผู้ใช้แบบ type ref cursor
- OPEN-FOR
- FETCH . INTO
- CLOSE

ใช้ USING แทน :&lt;id&gt;

- Dynamic สามารถใช้ร่วมกับ Anonymous block ได้

```
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 กรณี USING ≈ PROCEDURE

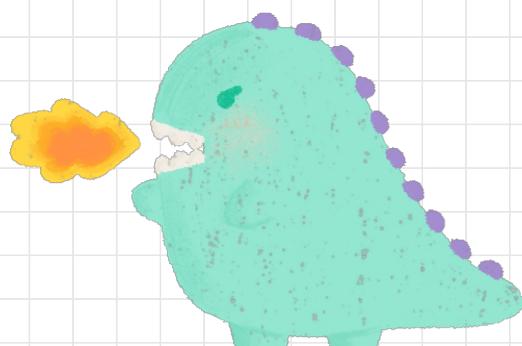
IN : ส่งค่าเข้า นำมไปยังแผลง

OUT : ส่งค่าออก

IN OUT : รับค่าเข้า และ ส่งค่าออก

\* เรียกตามลักษณะที่เป็น placeholder

ใน string dynamic



## ② - DBMS\_SQL

- ใช้ในกรณีไม่รู้อะไรเลย ทั้งจำนวน column, datatype ลำดับตัวเลขต่างๆ, จึงต้องเขียนทุกหันต่อนเอง ซึ่งช่วงกว่า ยังยุ่งกว่า (เขียนตามชื่อตัวเอง)
- 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

### เตือน!

```
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_SQL.CLOSE_CURSOR(v_cur_id);
  DBMS_OUTPUT.PUT_LINE(v_rows_added|| ' row added');
END; /
```

OPEN\_CURSOR : q: return cursor id ที่เป็น INTEGER ออกรอก

เมื่อเข้ามาจะ DBMS\_SQL

\* ร่วมกับลูกศิษย์ทัวร์เพื่อการรับ

} ใน stmt มี placeholder 3 ตัว ก็ต้อง bind\_variable 3 คำสั่ง ผูกตามชื่อใน stmt แบบตัวเปลี่ยน

- EXECUTE คำสั่งเป็น SELECT ดูไม่คล่องไว ไม่ต้องอ่านเป็น มากับกันได้

- ถ้าคำสั่งเป็น INSERT/UPDATE/

DELETE ตัว method EXECUTE

จะ return จำนวน ROW ที่ทำการ

ออกมาก



# Lecture 9. (Trigger)

## ► TRIGGER

- คือ program (PL/SQL) ที่ถูกกิจกรรมการณ์ (event) ที่เกิดขึ้นบน Schema / db / table / view พอดี compile แล้ว จะเก็บ database จากนั้นถ้ามี event เกิดขึ้นก็จะ Object จะทำอัตโนมัติ
- Application & Database Trigger
  - trigger บน App
  - trigger บน DB

ex กดปุ่มบันทึก... ex ทำ DML ข้อมูล変わแล้วให้ทำ...
- ใช้ Trigger ในการ Business
  - Security : ex ตรวจสอบ USER ที่ทำงานอยู่ในเวลาที่กำหนดให้
  - Auditing : กรณีเก็บ log ex เก็บไว้ในตารางข้าง
  - Data integrity : ทำให้ข้อมูลที่คงอยู่ต้อง
  - Referential integrity
  - Table replication : ex การ insert ตารางหนึ่ง และ trigger ที่จะ insert อีกตารางหนึ่ง (ลักษณะ DB)
  - Computing derived data automatically
  - Event logging ≈ Auditing

## ► โครงสร้าง

```

CREATE [OR REPLACE] TRIGGER trigger_name
[timing] -- when to fire the trigger
[event] [OR event2 OR event3]
ON object_name
[REFERENCING OLD AS old / NEW AS new]
FOR EACH ROW -- default is statement level trigger
WHEN (condition)
DECLARE
BEGIN
... trigger body -- executable statements
[EXCEPTION . . .]
END [trigger name];

timing = BEFORE | AFTER | INSTEAD OF
event = INSERT | DELETE | UPDATE | UPDATE OF column list
  
```

FOR EACH ROW : ใช้กับได้ ไม่ใช่กับได้

- ถ้าไม่ใช่ ▶ เป็น statement-level ทำครั้งเดียว  
ถ้ามี record / ไม่มี record ก็ทำ
- ถ้าใช่ ▶ เป็น row-level  
เป็น trigger ที่ run ก่อน 1 record  
(สามารถใช้ :OLD, :NEW ได้ / มากกว่า 1 ครั้ง WHEN ได้)

▶ record ในที่สุดเมื่อมีอะไรก็ทำ trigger

### Timer (Available Trigger Type)

#### \* Simple DML trigger

- BEFORE : ทำ trigger ก่อน DML
- ALTER : ทำ trigger หลังจาก DML
- INSTEAD OF : 代替 view

#### Compound trigger

#### Non-DML trigger

DDL : CREATE, DROP  
Database : Start up, Shutdown

### Trigger Event Type

#### \* Trigger event : DML statement

- INSERT
- UPDATE [OF column] : รับ column ได้
- DELETE

#### \* Trigger body : statement เรียกว่า PROC / FN / PACKAGE

## NOTE

### • Trigger ຈີ່ກົບ

- Database  
- Schema  
- Table } event ທີ່ໄດ້ໃຫ້ຄ່ອງ INSERT, UPDATE, DELETE  
timer ທີ່ໄດ້ BEFORE, AFTER

- View } event ທີ່ໄດ້ໃຫ້ຄ່ອງ INSERT, UPDATE, DELETE  
timer ທີ່ໄດ້ INSTEAD OF

### • Statement-level VS Row-level

#### \* Statement-level

- ເປັນ default ໃນການຮັດງານ trigger
- ຈະທຳ trigger ແລ້ວຄົງເລືຍ  
(ໄລ່ຈ່າກວ່ານີ້ກໍ record / ໄນສີ record) ກ່ອງ 1 ຄຽງ

#### \* Row-level

- ຕ້ອງໃຊ້ ! FOR EACH ROW
- ຈະທຳ trigger ຕັ້ງ 1 record  
(ກ່າວຖຸກ record ກ່າວຈົບຕົວໃໝ່ WHEN ຕັ້ງ)
- ຈະໄລ່ຈ່າກວ່ານີ້ໃນໆ record

### ► ຕັ້ງຂ່າຍ່າງ Statement-level → ໄນສີ FOR EACH ROW

```
CREATE OR REPLACE TRIGGER secure_emp BEFORE
INSERT OR UPDATE OR DELETE ON employees
BEGIN
    IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
    (TO_CHAR(SYSDATE,'HH24') NOT BETWEEN '08' AND '18') THEN
        IF DELETING THEN RAISE_APPLICATION_ERROR(
            -20502,'You may delete from EMPLOYEES table' ||
            'only during normal business hours.');
        ELSIF INSERTING THEN RAISE_APPLICATION_ERROR(
            -20500,'You may insert into EMPLOYEES table' ||
            'only during normal business hours.');
        ELSIF UPDATING ('SALARY') THEN
            RAISE_APPLICATION_ERROR(-20503, 'You may ' ||
            'update SALARY only normal during business hours.');
        ELSE RAISE_APPLICATION_ERROR(-20504,'You may' ||
            ' update EMPLOYEES table only during' ||
            ' normal business hours.');
        END IF;
    END IF;
END;
```

ສ້າມາຣັກໃຊ້ Reserve Word  
ໃນການກຳ event ແລ້ວ

- DELETING
- INSERTING
- UPDATE

## ► ຕົ້ນອຍ່າງ Row-level : ຂັ້ນ FOR EACH ROW

```

CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
    IF INSERTING THEN
        :NEW.commission_pct := 0;
    ELSIF :OLD.commission_pct IS NULL THEN
        :NEW.commission_pct := 0;
    ELSE
        :NEW.commission_pct := :OLD.commission_pct+0.05;
    END IF;
END;
/

```

\* ດ້ວຍ FOR EACH ROW ຂະໜາກຮູບໃຈ  
 :OLD, :NEW ແລະ WHEN ໄດ້  
 ຄັ້ນເປັນ statement level ສະໜັບໄດ້

• ຖືກ record ທີ່ມີ ເຊັ່ນໃຈ  
 job\_id ອິນໄສ = 'SA\_REP'  
 ດຳວ່າທີ່ trigger ນີ້

\* NEW, OLD ອີ່ໃນ WHEN ໃນຕົວນີ້ :

## ► ຕົ້ນອຍ່າງການໃຈ INSTEAD OF : ຊັ້ນກົບ VIEW

Creating an INSTEAD OF Trigger (continued)

```

CREATE OR REPLACE TRIGGER new_emp_dept
INSTEAD OF INSERT OR UPDATE OR DELETE ON emp_details
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO new_emps
        VALUES (:NEW.employee_id, :NEW.last_name,
                :NEW.salary, :NEW.department_id);
        UPDATE new_depts
        SET dept_sal = dept_sal + :NEW.salary
        WHERE department_id = :NEW.department_id;
    ELSIF DELETING THEN
        DELETE FROM new_emps
    END IF;
END;
/

```

emp\_details  
 ເປັນ view ຂອງມາຈັນ  
 ສະໜັບ new\_emps ໃລະ  
 new\_depts  
 "ກະຈຸ INSTEAD OF"

