BAB 10 : PEMBUATAN PROSEDUR

Tujuan Pembelajaran :

- Dapat membedakan *anonymous PL/SQL blocks* dengan blok PL/SQL yang diberi nama (subprogram)
- Dapat membuat procedure
- Bisa membedakan antara parameter formal dengan parameter aktual
- Fitur-fitur yang ada pada mode parameter
- Dapat membuat procedure dengan parameter
- Dapat menangani exception di dalam procedure
- Dapat menghapus procedure

10.1. Sub Program PL/SQL

Sub program adalah :

- Sebuah blok PL/SQL bernama yang dapat menerima parameter dan dipanggil dari lingkungan panggilan.
- Ada dua jenis:
  1. Suatu prosedur yang melakukan tindakan, dan
  2. Suatu fungsi yang menghitung nilai
- Berdasarkan standar struktur blok PL / SQL
- Menyediakan modularitas, reusability, exstensibility, dan maintainability
- Memberikan perawatan yang mudah, meningkatkan keamanan data dan integritas, meningkatkan performa/kinerja, dan peningkatan code dengan jelas.

10.2. Struktur Blok untuk Sub Program PL/SQL

Blok struktur untuk Anonymous Blok :

`DECLARE` (optional)
Bagian yang mendeklarasikan obyek PL/SQL

`BEGIN` (mandatory)
Mendefinisikan statement yang bisa dijalankan

`EXCEPTION` (optional)
Mendefinisikan tindakan yang diambil jika terdapat suatu kesalahan (error)

`END;` (mandatory)

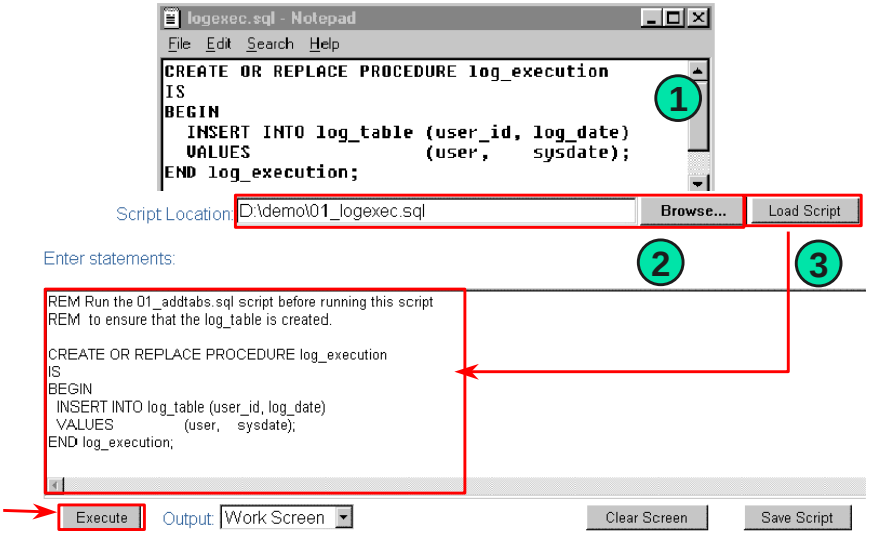
Blok struktur untuk PL/SQL Subprogram :

10.3. Keuntungan Penggunaan Sub Program

Keuntungan dari penggunaan Sub program :

- Mudah pemeliharaannya
- Meningkatkan integritas dan keamanan data
- Meningkatkan unjuk kerja
- Memperjelas maksud dari kode yang ditulis

10.4. Penulisan Program pada iSQL*PLUS



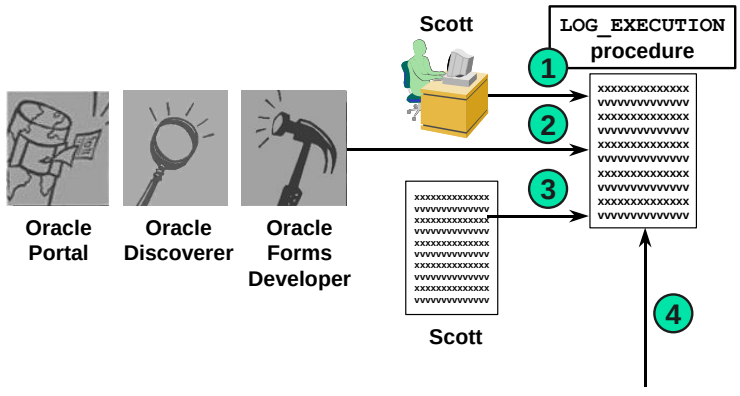
Cara menulis program PL/SQL dengan iSQL*PLUS :

1. Menggunakan teks editor untuk membuat file skrip SQL untuk mendefinisikan program yang dibuat. Pada ilustrasi diatas dibuat stored procedure LOG_EXECUTION tanpa parameters. Procedure menyimpan username dan tanggal saat ini dalam tabel database yang bernama LOG_TABLE Kemudian pada window browser *i*SQL*Plus :
2. Gunakan tombol Browse untuk menempatkan file SQL script.
3. Gunakan tombol Load Script untuk memanggil script ke dalam *i*SQL*Plus buffer.
4. Gunakan tombol Execute untuk menjalankan kode yang ditulis. Secara default output ditampilkan pada screen.

PL/SQL subprograms juga dapat dibuat dengan menggunakan Oracle development tools semisal Oracle Forms Developer.

10.5. Memanggil Stored Procedure dan Fungsi

Gambar berikut ini mengilustrasikan pemanggilan stored procedure dan fungsi :



Procedure dan Function yang telah dibuat sebelumnya, dapat dipanggil dari bermacam environment semisal : *i*SQL*Plus, Oracle Forms Developer, Oracle Discoverer, Oracle Portal, tools Oracle atau precompiler application yang lainnya.

10.6. Prosedur

Procedure adalah tipe subprogram yang membentuk suatu tindakan atau aksi.
Procedure dapat disimpan dalam database sebagai schema object, agar dapat dipanggil berulang kali.

10.7.Sintak untuk Pembuatan Prosedur

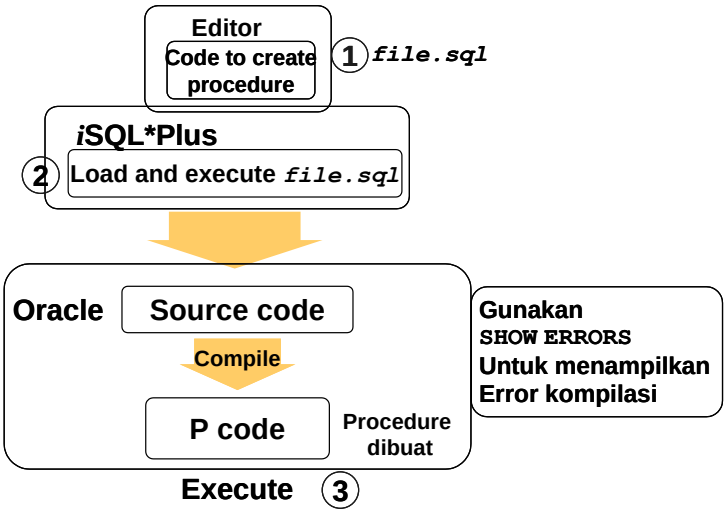
Berikut ini sintak untuk pembuatan procedure :

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode1] datatype1,
  parameter2 [mode2] datatype2,
  . . .)]
IS|AS
PL/SQL Block;
```

Option **REPLACE** menyatakan jika procedure telah ada sebelumnya, maka procedure tersebut akan dihapus dan diganti dengan procedure versi baru yang dibuat dalam statement. Blok dari PL/SQL dimulai dari **BEGIN** atau deklarasi local variabel dan diakhiri dengan **END** atau **END** ***procedure_name***.

10.8.Mengembangkan Prosedur

Berikut ini tahapan dalam pembuatan procedure :



10.9.Parameter Formal vs. Aktual

Parameter formal adalah variabel yang dideklarasikan dalam daftar parameter yang ada pada spesifikasi subprogram.
Contoh :

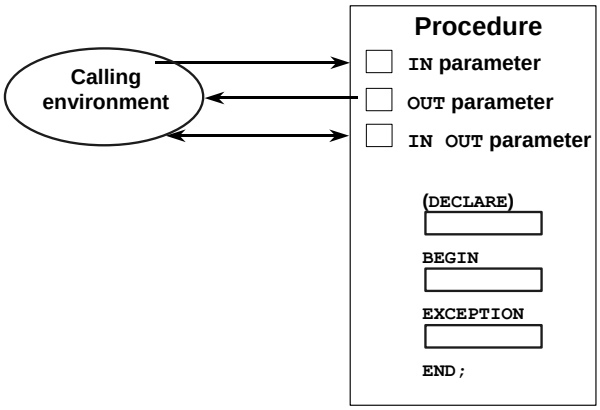
```
CREATE PROCEDURE raise_sal(p_id NUMBER, p_amount NUMBER)
...
END raise_sal;
```

Parameter aktual adalah variabel atau ekspresi yang ditulis dalam daftar parameter pada procedure yang dipanggil.
Contoh :

```
raise_sal(v_id, 2000)
```

10.10. Mode Parameter pada Prosedur

Kita dapat mentransfer nilai ke dan dari lingkungan pemanggilan dengan menggunakan parameter. Ada **3 (tiga) mode parameter** yang bisa digunakan yaitu : **IN**, **OUT** atau **IN OUT**.

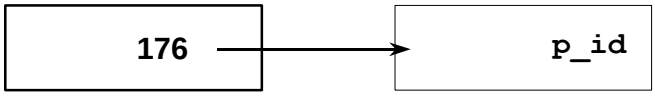


Catatan : TIPE DATA yang digunakan hanya definisi %TYPE, definisi %ROWTYPE, atau tipe data explicit tanpa spesifikasi ukuran (size).

Karakteristik dari tiap mode parameter :

IN	OUT	IN OUT
Mode default	Harus ditentukan	Harus ditentukan
Nilai dilewatkan ke dalam subprogram	Dikembalikan ke calling environment	Dilewatkan ke dalam subprogram; dikembalikan ke calling environment
Parameter formal berlaku sebagai konstanta	Variabel tidak perlu diinisialisasi	Variabel diinisialisasi
Parameter aktual, bisa literal, ekspresi, konstanta, atau variabel yang diinisialisasi	Berupa variabel	Berupa variabel
Dapat ditandai dengan nilai default	Tidak dapat ditandai dengan nilai default	Tidak dapat ditandai dengan nilai default

10.11. Parameter IN




```
CREATE OR REPLACE PROCEDURE raise_salary
  (p_id IN employees.employee_id%TYPE)
IS
BEGIN
  UPDATE employees
  SET    salary = salary * 1.10
  WHERE  employee_id = p_id;
END raise_salary;
/
```

```
Procedure created.
```

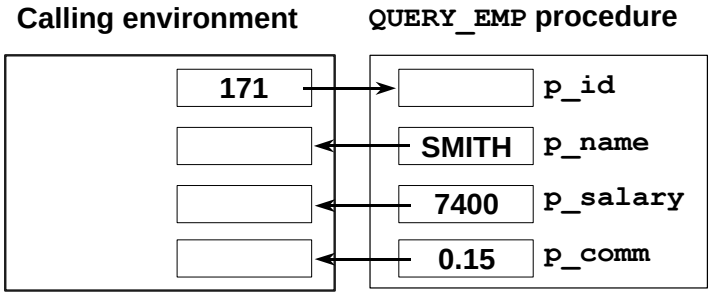
Gambar diatas memperlihatkan procedure dengan satu parameter **IN**. Pada saat dipanggil, procedure RAISE_**SALARY** menerima parameter untuk **employee ID** dan mengubah record dengan perubahan nilai salary yang dinaikkan sebesar 10%. Untuk menjalankan procedure dalam iSQL*PLUS digunakan perintah **`EXECUTE`**, misal : **`EXECUTE raise_salary (176)`**

Untuk **memanggil procedure dari procedure yang lain** digunakan pemanggilan **langsung** dengan menuliskan : **raise_salary (176);**

Parameter **IN** dilewatkan sebagai konstanta dari calling environment ke dalam procedure. Tindakan yang dilakukan untuk merubah parameter IN, akan menyebabkan terjadinya kesalahan (error).

Catatan :
Perintah **UPDATE** atau **DELETE** yang tidak menghasilkan satu barispun akan menyebabkan error, karenanya gunakan atribut cursor **SQL%FOUND, SQL%NOTFOUND** dan **SQL%ROWCOUNT** untuk memeriksa jumlah baris yang dihasilkan.

10.12. Parameter OUT



Pada contoh diatas, dibuat procedure dengan parameter **OUT** untuk mendapatkan informasi tentang data pegawai. Procedure ini menerima nilai **171** untuk **employee ID**, dan mendapatkan **nama**, **salary** dan **prosentase komisi** sebagai **parameter output**, yang didapat dari data pegawai yang memiliki employee ID 171. Kode untuk membuat procedure QUERY_EMP sebagai berikut :

```
VARIABLE g_name      VARCHAR2(25)
VARIABLE g_sal       NUMBER
VARIABLE g_comm      NUMBER

EXECUTE query_emp(171, :g_name, :g_sal, :g_comm)

PRINT g_name
```

PL/SQL procedure successfully completed.

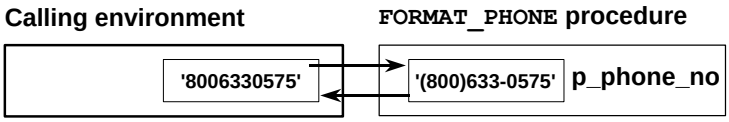
G_NAME
Smith

Berikut ini contoh cara menjalankan procedure yang telah dibuat : Deklarasikan host variabel, jalankan procedure query_emp, dan cetak nilai dari variabel global G_NAME.

```
CREATE OR REPLACE PROCEDURE query_emp
  (p_id       IN   employees.employee_id%TYPE,
   p_name     OUT  employees.last_name%TYPE,
   p_salary   OUT  employees.salary%TYPE,
   p_comm     OUT  employees.commission_pct%TYPE)
IS
BEGIN
  SELECT   last_name, salary, commission_pct
   INTO    p_name, p_salary, p_comm
   FROM    employees
   WHERE   employee_id = p_id;
END query_emp;
/
```

Procedure created.

10.13. Parameter IN-OUT




```
CREATE OR REPLACE PROCEDURE format_phone
  (p_phone_no IN OUT VARCHAR2)
IS
BEGIN
  p_phone_no := '(' || SUBSTR(p_phone_no,1,3) ||
                ')' || SUBSTR(p_phone_no,4,3) ||
                '-' || SUBSTR(p_phone_no,7);
END format_phone;
/
```

Procedure created.

Contoh diatas menunjukkan pemakaian parameter **IN OUT**.

Dengan parameter **IN OUT**, kita dapat **melewatkan** sebuah **nilai ke dalam procedure** sekaligus **mengembalikan nilai** baru **ke calling environment**. Nilai yang dikembalikan bisa nilai variabel yang dilewatkan tanpa mengalami perubahan atau nilai baru yang berbeda dari sebelumnya. Parameter **IN OUT** berlaku sebagai **variabel inisialisasi**, karena pertama variabel diinisialiasasi terlebih dahulu kemudian selanjutnya nilainya bisa berubah.

```
VARIABLE g_phone_no VARCHAR2(15)
BEGIN
  :g_phone_no := '8006330575';
END;
/
PRINT g_phone_no
EXECUTE format_phone (:g_phone_no)
PRINT g_phone_no
```

PL/SQL procedure successfully completed.

G_PHONE_NO
8006330575

PL/SQL procedure successfully completed.

G_PHONE_NO
(800)633-0575

Untuk menampilkan Parameter IN OUT dengan iSQL*PLUS :

1. Buat ***host variabel*** dengan perintah VARIABLE.
2. Beri nilai pada *host variabel* di dalam anonymous block.
3. Panggil procedure FORMAT_PHONE yang memakai host variabel sebagai parameter IN OUT. Gunakan tanda (:) untuk merefer host variabel.
4. Untuk menampilkan nilai gunakan perintah PRINT.

10.14. Melewatkan (passing) Parameter

Ada beberapa metode untuk melewatkan (passing) parameter :

- Positional, mencantumkan parameter aktual dengan urutan yang sama dengan parameter formal.
- Berdasarkan nama (named), mendaftar parameter aktual sesuai dengan tiap-tiap asosiasi dengan parameter formal yang berkorespondensi.
- Kombinasi, mencantumkan beberapa parameter aktual dengan cara positional dan sebagian yang lainnya dengan cara berdasarkan nama (named).

Berikut ini contoh pelewatan (passing) parameter dengan menggunakan ketiga metode yang sudah dibahas sebelumnya :

```
BEGIN
  add_dept;
  add_dept ('TRAINING', 2500);
  add_dept ( p_loc => 2400, p_name =>'EDUCATION');
  add_dept ( p_loc => 1200) ;
END;
/
SELECT department_id, department_name, location_id
FROM departments;
```

PL/SQL procedure successfully completed.

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
30	Purchasing	1700
40	Human Resources	2400
...		
290	TRAINING	2500
300	EDUCATION	2400
310	unknown	1200

31 rows selected.

10.15. Parameter DEFAULT

Parameter IN dapat diinisialisasi dengan menggunakan nilai DEFAULT, seperti contoh berikut :

```
CREATE OR REPLACE PROCEDURE add_dept
   (p_name  IN departments.department_name%TYPE
                              DEFAULT 'unknown',
    p_loc   IN departments.location_id%TYPE
                              DEFAULT 1700)
IS
BEGIN
  INSERT INTO departments(department_id,
              department_name, location_id)
  VALUES (departments_seq.NEXTVAL, p_name, p_loc);
END add_dept;
/
```

Procedure created.

10.16. Memanggil Prosedur dari Prosedur yang lain

Berikut ini pemanggilan procedure dari procedure yang lain :

```
CREATE OR REPLACE PROCEDURE process_emps
IS
   CURSOR emp_cursor IS
    SELECT employee_id
    FROM   employees;
BEGIN
   FOR emp_rec IN emp_cursor
   LOOP
    raise_salary(emp_rec.employee_id);
   END LOOP;
   COMMIT;
END process_emps;
/
```

10.17. Memanggil Prosedur dari Anonymous Block

Berikut ini pemanggilan procedure dari Anonymous Block :

```
DECLARE
  v_id NUMBER := 163;
BEGIN
  raise_salary(v_id);    --invoke procedure
  COMMIT;
...
END;
```

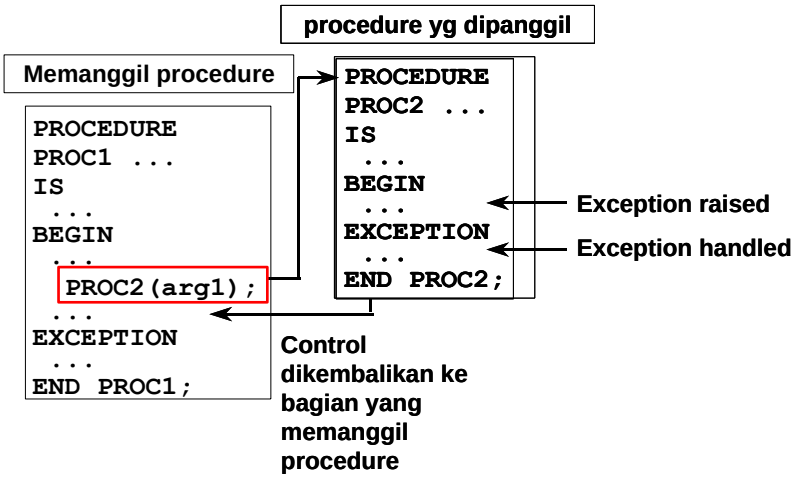
10.18. Prosedur Bersarang

Berikut ini penulisan procedure dan pemanggilannya dari procedure yang lain :

```
CREATE OR REPLACE PROCEDURE leave_emp2
  (p_id  IN  employees.employee_id%TYPE)
IS
  PROCEDURE log_exec
  IS
  BEGIN
    INSERT INTO log_table (user_id, log_date)
    VALUES (USER, SYSDATE);
  END log_exec;
BEGIN
  DELETE FROM employees
  WHERE employee_id = p_id;
  log_exec;
END leave_emp2;
/
```

10.19. Penanganan Exception

Ilustrasi berikut ini menggambarkan cara penanganan exception dalam procedure :

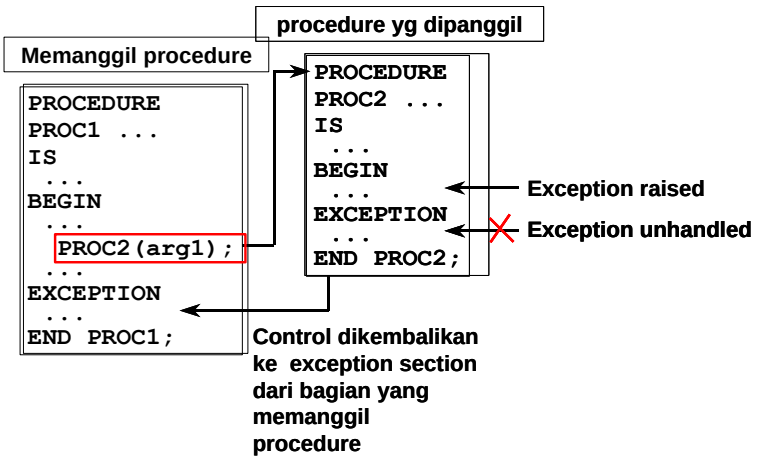


Berikut ini contoh penanganan exception dalam program PL/SQL :

```
CREATE PROCEDURE p2_ins_dept(p_locid NUMBER) IS
 v_did NUMBER(4);
BEGIN
 DBMS_OUTPUT.PUT_LINE('Procedure p2_ins_dept started');
 INSERT INTO departments VALUES (5, 'Dept 5', 145, p_locid);
 SELECT department_id INTO v_did FROM employees
  WHERE employee_id = 999;
END;
```

```
CREATE PROCEDURE p1_ins_loc(p_lid NUMBER, p_city VARCHAR2)
IS
 v_city VARCHAR2(30); v_dname VARCHAR2(30);
BEGIN
 DBMS_OUTPUT.PUT_LINE('Main Procedure p1_ins_loc');
 INSERT INTO locations (location_id, city) VALUES (p_lid, p_city);
 SELECT city INTO v_city FROM locations WHERE location_id = p_lid;
 DBMS_OUTPUT.PUT_LINE('Inserted city '||v_city);
 DBMS_OUTPUT.PUT_LINE('Invoking the procedure p2_ins_dept ...');
 p2_ins_dept(p_lid);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE('No such dept/loc for any employee');
END;
```

10.20. Exception yang tidak bisa ditangani (Unhandled Exception)



Ilustrasi berikut ini menggambarkan apa yang terjadi jika exception tidak tertangani dalam procedure :
Berikut ini contoh unhandled exception dalam program PL/SQL :

```
CREATE PROCEDURE p2_noexcep(p_locid NUMBER) IS
 v_did NUMBER(4);
BEGIN
 DBMS_OUTPUT.PUT_LINE('Procedure p2_noexcep started');
 INSERT INTO departments VALUES (6, 'Dept 6', 145, p_locid);
 SELECT department_id INTO v_did FROM employees
  WHERE employee_id = 999;
END;
```

```
CREATE PROCEDURE p1_noexcep(p_lid NUMBER, p_city VARCHAR2)
IS
 v_city VARCHAR2(30); v_dname VARCHAR2(30);
BEGIN
 DBMS_OUTPUT.PUT_LINE(' Main Procedure p1_noexcep');
 INSERT INTO locations (location_id, city) VALUES (p_lid, p_city);
 SELECT city INTO v_city FROM locations WHERE location_id = p_lid;
 DBMS_OUTPUT.PUT_LINE('Inserted new city '||v_city);
 DBMS_OUTPUT.PUT_LINE('Invoking the procedure p2_noexcep ...');
 p2_noexcep(p_lid);
END;
```

10.21. Ringkasan

Pada bab ini telah dipelajari tentang :

- Procedure sebagai sub program yang membentuk suatu aksi
- Procedure dapat dibuat dengan menggunakan perintah CREATE PROCEDURE
- Procedure dapat dicompile dan disimpan dalam database.
- Parameter digunakan untuk melewatkan data dari lingkungan yang memanggil ke dalam procedure.
- Ada 3 (tiga) macam mode parameter : IN, OUT dan IN OUT.

10.22. Latihan Soal

1. Buat procedure ADD_JOB untuk menambahkan baris data baru ke dalam table JOBS. Procedure memiliki dua parameter yang menampung kode job dan nama job.

2. Buat procedure UPD_JOB untuk memodifikasi data pada table job. Procedure memiliki dua parameter kode job dan nama job yang baru. Buat exception handling untuk menangani kode job yang tidak ada pada table JOBS, berikan pesan bahwa ‘kode job tidak ada dalam table JOBS’.
Jalankan procedure untuk mengganti kode job IT_DBA diganti dengan title ’Data Administrator’.
Cobalah dengan kode job yang tidak ada di table JOBS , missal kode job = ‘IT_WEB’, dan perhatikan apakah exception handling telah berjalan dengan benar.