PRAKTIKUM 13

PROSEDURE, FUNCTION DAN TRIGGER

A.DASAR TEORI

1. PROSEDURE

Sebenarnya prosedur pada PL/SQL hampir sama dengan prosedur bahasa pemrograman lainnya. Prosedur memiliki header dan body. Header mengandung nama prosedur dan parameter atau variabel yang digunakan oleh prosedur. Sedangkan body mengandung bagian deklarasi, bagian eksekusi, dan bagian exception handling.

Sintaks umum untuk membuat prosedur

```
CREATE [OR REPLACE] PROCEDURE nama_prosedur (parameter1 tipedata,
parameter2 tipedata,…)
IS
   Variable variabel lokal
BEGIN
   Statemen ;

END;
```

Eksekusi Prosedur Unutuk melakukan eksekusi terhadap suatu prosedur dari luar blok PL/SQL, kita harus menggunkan statemen EXECUTE, atau cukup dituliskan dengan EXEC saja. Namun apabila Anda ingin memanggil prosedur dari dalam blok PL/SQL, maka statemen EXECUTE tidak perlu di tuliskan.

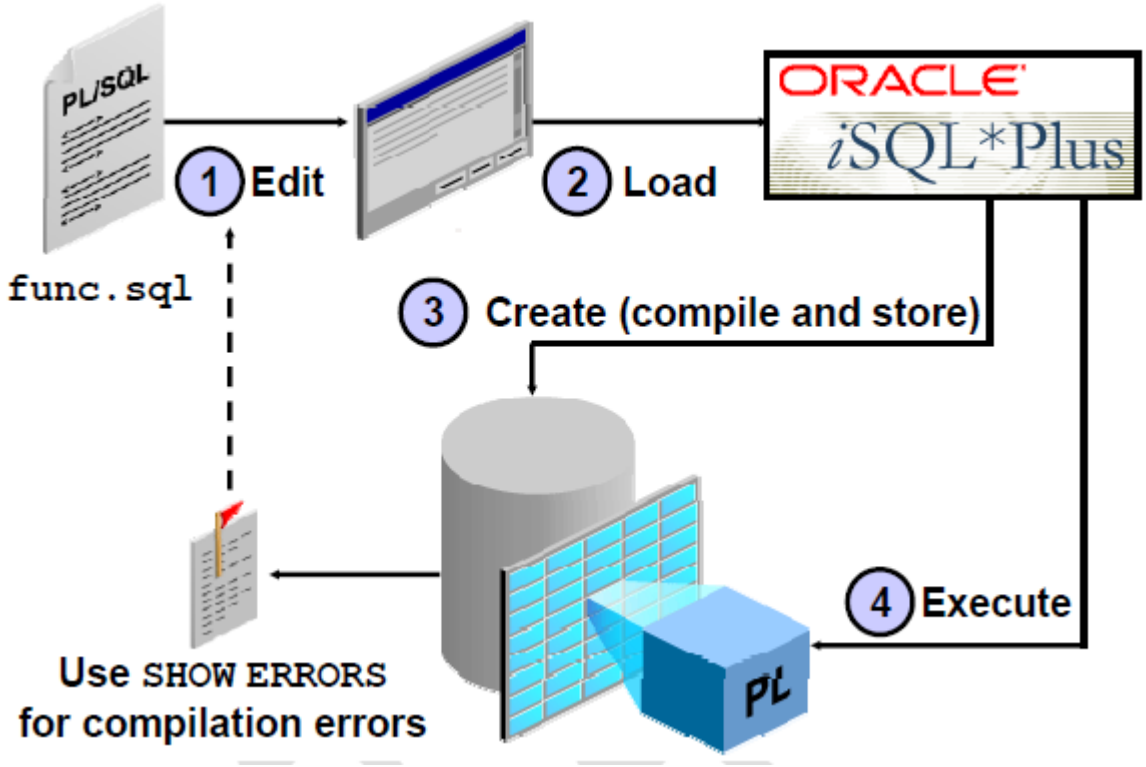
Berikut ini bentuk umumnya.

```
EXEC[UTE] NamaProsedur (daftar_parameter);
```

2. FUNGSI

Function adalah jenis PL/SQL block yang menghasilkan satu nilai. Secara umum, function digunakan melakukan perhitungan, mengecek eksistensi dan kevalidan suatu data. Function bisa dilibatkan dalam expresi. Function bisa disimpan dalam database sebagai object schema, sehingga suatu function bisa digunakan berulangkali tanpa harus melakukan parsing dan compile ulang.

Siklus Stored Function



Berikut ini siklus dari sebuah stored function:

1. Pada poin 1 dan 2 stored function dibuat atau di edit menggunakan SQL*Plus ataupun editor lain (misal: Oracle SQL Developer).
2. Poin 3 dilakukan jika stored function tersebut hanya di compile untuk mengetahui ada tidaknya kesalahan dari baris program.
3. Poin 4 dilakukan ketika stored function yang telah di buat dieksekusi, execute dapat dilakukan setelah proses compile maupun tanpa proses compile terlebih dahulu (auto compiled).

Jika ditemukan kesalahan pada stored function yang di buat, pesan error akan muncul. Pesan error ini dijadikan acuan oleh pembuat stored function untuk memperbaiki kesalahan yang terjadi. Hanya satu buah RETURN yang dikembalikan oleh stored function. Jika menggunakan kondisi (IF), RETURN dapat berisi beberapa nilai yang berbeda.

Aturan penulisan stored function

```
CREATE OR REPLACE FUNCTION nama_fungsi (parameter1 tipedata, parameter2
tipedata,…)
RETURN tipedata_fungsi
IS
      Variable_variabel_lokal
BEGIN
      Statemen;
      …
RETURN nilai_fungsi;
END;
```

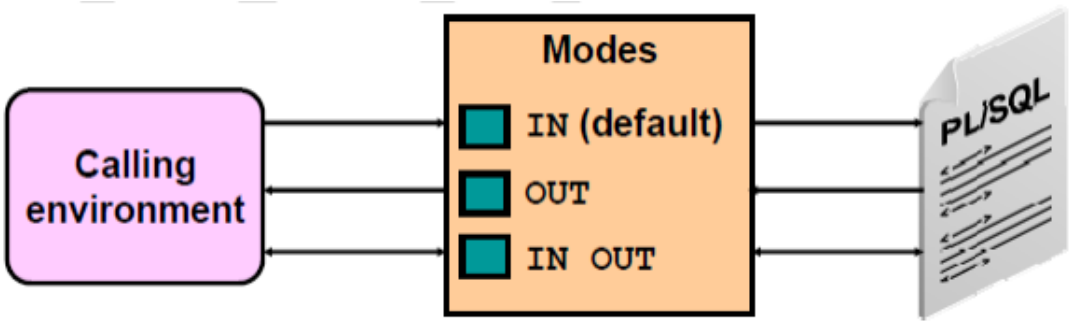
Ekesekusi Fungsi

Tidak seperti prosedur yang dalam eksekusinya membutuhkan perintah EXECUTE, pada fungsi kita dapat langsung memasukkan nilainya ke dalam suatu variable yang sama tipedatanya maupun melalui statemen SQL. Atau dapat juga melalui statemen SQL seperti yang ditunjukkan oleh kode berikut .

```
SELECT nama_fungsi FROM DUAL ;
```

Tabel Dual adalah table dummy yang disediakan oleh Oracle.

4. PARAMETER



Dalam Oracele, parameter diklasifikasikan menjadi tiga jenis, yaitu parameter masukan, keluarn dan masukan-keluaran.

1. Parameter Masukan Parameter masukan adalah parameter yang berguna untuk menyimpan nilai yang digunakn sebagai input di dalam badan prosedur maupun fungsi. Parameter ini ditandai dengan mode IN.

2. Paramater Keluaran Parameter keluaran adalah parameter yang menampung nilai hasil dari suatu proses yang dilakukan dalam sub program.Pada kenyataannya parameter keluaran lebih sering di temui di dalam prosedur dari pada didalam fungsi. Parameter ini ditandai dengan mode OUT.

3. Parameter Masukan – Keluaran Parameter ini merupakan parameter gabungan dari kedua jenis diatas. Parameter ini berperan sebagai parameter masukan sekaligus parameter keluaran. Parameter ini ditandai dengan mode IN OUT.

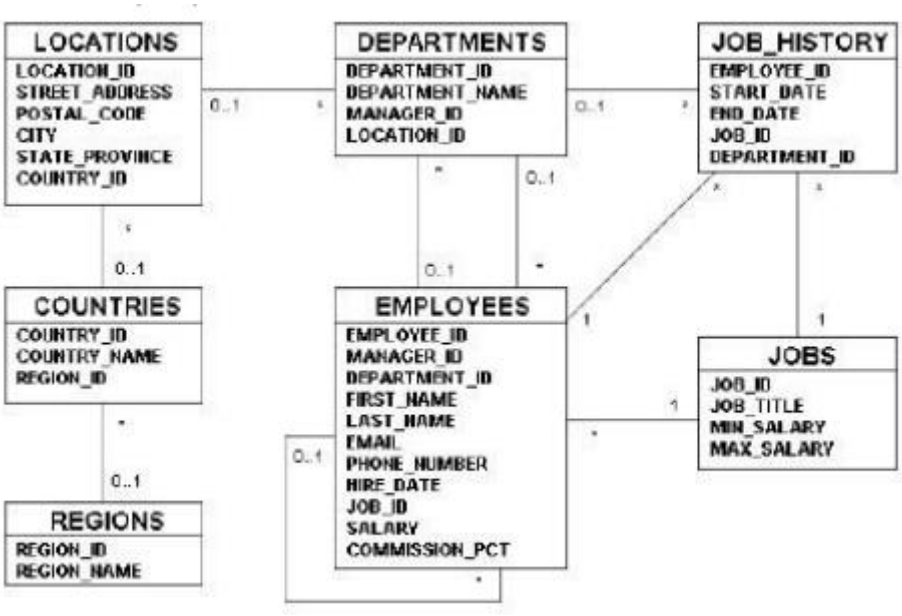
Parameter pada stored procedure dapat menggunakan tipe data :

• skalar

• %TYPE

• %ROWTYPE

Human Resources (HR) Schema



Schema HR merupakan bagian dari schema contoh yang sudah ada pada saat pertama kali kita melakukan instalasi oracle. Pada praktikum ini dan seterusnya kita akan menggunakan schema HR untuk mengerjakan latihan soal. Adapun tabel-tabel yang ada pada schema HR adalah sebagai berikut:

REGIONS: merupakan tabel yang berisi data region/ wilayah seperti Americas, Asia, dan lain-lain.

COUNTRIES: merupakan tabel yang berisi data negara yang berhubungan dengan data wilayah.

LOCATIONS: merupakan alamat lengkap dari kantor, gudang, atau tempat produksi dari perusahaan di suatu negara.

DEPARTMENTS: merupakan tabel yang berisi data detail department tempat pegawai bekerja. Tabel ini juga berelasi dengan tabel employees dalam bentuk penempatan pegawai dan manajer dari department tersebut.

EMPLOYEES: merupakan tabel yang menyimpan data pegawai yang ada di perusahaan.

JOB : merupakan tabel yang menyimpan detail data jenis-jenis job/ posisi dari pegawai.

JOB_HISTORY: merupakan tabel yang menyimpan data riwayat posisi pegawai.

Contoh Stored Procedure

```
CREATE OR REPLACE PROCEDURE coba AS
v_var1 VARCHAR2(20);
 v_var2 VARCHAR2(6);
v_var3 NUMBER(5,3);
BEGIN
v_var1 := 'string literal';
v_var2 := '12.345';
```

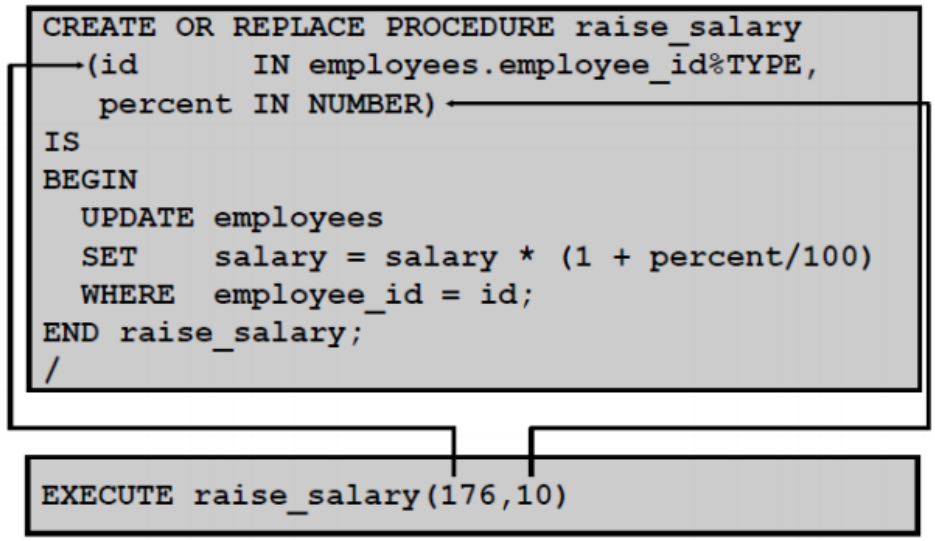
```
v_var3 := 12.345;
DBMS_OUTPUT.PUT_LINE('v_var1: '||v_var1);
DBMS_OUTPUT.PUT_LINE('v_var2: '||v_var2);
DBMS_OUTPUT.PUT_LINE('v_var3: '||v_var3);
END;
/
```

Untuk melakukan eksekusi pada stored procedure coba ketikkan perintah berikut : EXECUTE coba

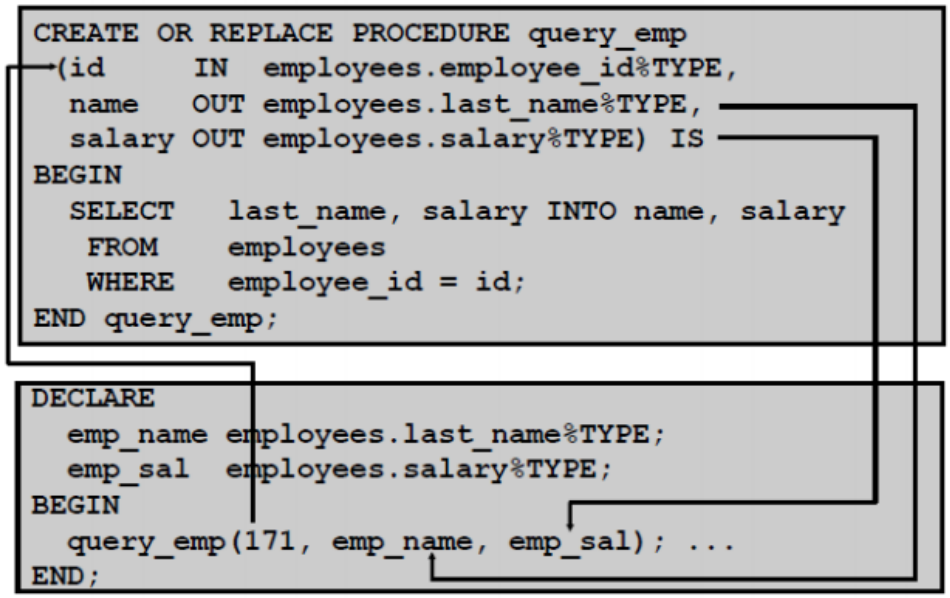
```
SQL> execute coba
v_var1: string literal
v_var2: 12.345
v_var3: 12.345

PL/SQL procedure successfully completed.
```

Contoh Stored Procedure Parameter IN

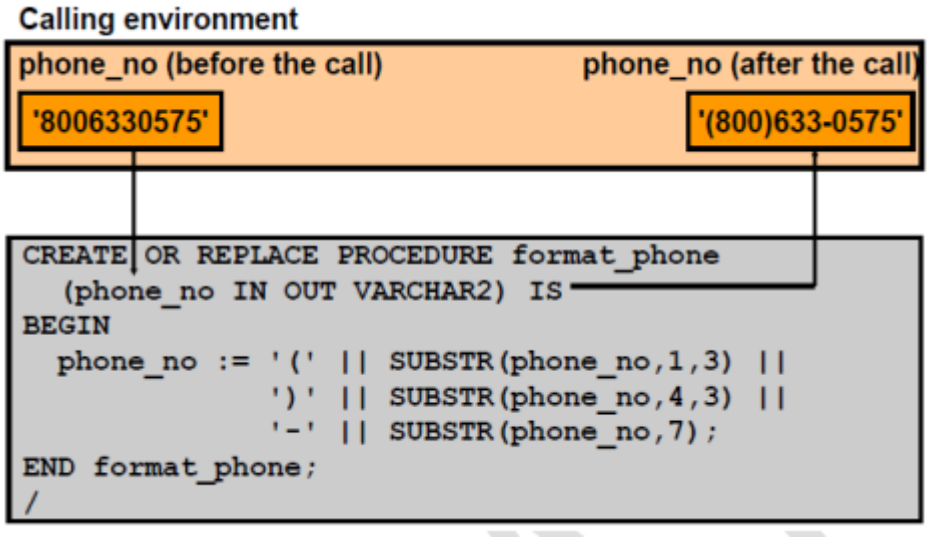


Contoh Stored Procedure Parameter OUT



```
SET SERVEROUTPUT ON
DECLARE
  emp_name employees.last_name%TYPE;
  emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(171, emp_name, emp_sal);
  DBMS_OUTPUT.PUT_LINE('Name: ' || emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || emp_sal);
END;
```

Contoh Stored Procedure Parameter IN OUT



Contoh Stored Procedure Parameter Passing

```
CREATE OR REPLACE PROCEDURE add_dept(
  name IN departments.department_name%TYPE,
  loc  IN departments.location_id%TYPE) IS
BEGIN
  INSERT INTO departments(department_id,
              department_name, location_id)
  VALUES (departments_seq.NEXTVAL, name, loc);
END add_dept;
/
```

Passing by position notation

```
EXECUTE add_dept ('TRAINING', 2500)
```

Passing by named notation

```
EXECUTE add_dept (loc=>2400, name=>'EDUCATION')
```

Removing stored procedure

```
DROP PROCEDURE procedure_name
```

Example:

```
DROP PROCEDURE raise_salary;
```

Contoh Stored Function

```
CREATE OR REPLACE FUNCTION get_sal
 (id employees.employee_id%TYPE) RETURN NUMBER IS
  sal employees.salary%TYPE := 0;
BEGIN
  SELECT salary
  INTO   sal
  FROM   employees
  WHERE  employee_id = id;
  RETURN sal;
END get_sal;
/
```

Function pada Oracle dapat digunakan untuk mengisi nilai suatu variable

```
DECLARE sal employees.salary%type;
BEGIN
  sal := get_sal(100); ...
END;
```

Eksekusi dari SQL*Plus sebagai berikut :

```
EXECUTE dbms_output.put_line(get_sal(100))
```

Eksekusi dari query SQL sebagai berikut :

```
SELECT job_id, get_sal(employee_id) FROM employees;
```

Removing Stored Functions

```
DROP FUNCTION function_name
```

```
DROP FUNCTION get_sal;
```

5. TRIGGER

Trigger adalah blok PL/SQL yang disimpan dalam database dan akan diaktivasi ketika kita melakukan statement-statement SQL (DELETE, UPDATE, dan INSERT) pada sebuah tabel. Aktivasi trigger didasarkan pada event yang terjadi di dalam tabel tersebut sehingga trigger dapat membantu dalam menjaga integritas dan konsistensi data. Implementasi trigger yang sering ditemui dalam dunia nyata adalah untuk mengeset dan mengubah nilai kolom dalam suatu tabel sehingga validasi nilai dari tabel tersebut akan terjaga. Adanya trigger dalam database akan meringankan kita dalam pembuatan aplikasi karena di dalam aplikasi yang kita buat, kita tidak perlu lagi untuk melakukan validasi data.

Tipe dari trigger adalah :

- Application trigger : diaktifkan pada saat terjadi event yang berhubungan dengan sebuah aplikasi
- Database trigger : diaktifkan pada saat terjadi event yang berhubungan dengan data (seperti operasi DML) atau event yang berhubungan dengan sistem (semisal logon atau shutdown) yang terjadi pada sebuah skema atau database.

PENGGUNAAN TRIGGER

Trigger dibuat sesuai dengan keperluan. Ada kalanya trigger perlu dibuat, dan kadangkala tidak perlu dibuat.

Trigger perlu dibuat pada saat :

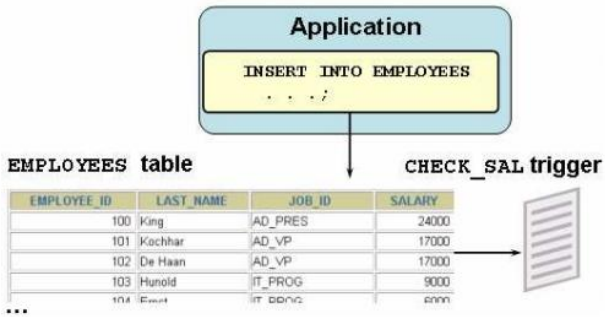
- membentuk sebuah aksi tertentu terhadap suatu event
- Memusatkan operasi global

Trigger tidak perlu dibuat, jika :

- Fungsionalitas yang diperlukan suatu ada pada Oracle server
- Duplikat atau sama dengan fungsi trigger yang lain.

Prosedur bisa dibuat dalam database, kemudian prosedur tersebut dipanggil pada trigger. Jika penggunaan trigger terlalu berlebihan, maka akan menyebabkan terjadi sifat ketidaktergantungan yang terlalu kompleks sehingga akan mempersulit pemeliharaan dari aplikasi yang besar.

Gambar berikut ini menunjukkan ilustrasi dari penggunaan trigger :



Pada gambar tersebut, database trigger CHECK_SAL memeriksa nilai gaji pada saat suatu aplikasi mencoba untuk memasukkan baris baru ke dalam table EMPLOYEES. Nilai yang terletak pada jangkauan diluar kategori pekerjaan akan diabaikan.

Sintak penulisan dari database trigger, berisi komponen berikut :

1. Trigger timing :
   a. Untuk tabel : BEFORE, AFTER
   b. Untuk view : INSTEAD OF
   c. Trigger event : INSERT, UPDATE atau DELETE
2. Nama tabel : yaitu nama tabel atau view yang berhubungan dengan trigger
3. Tipe trigger : Baris atau Pernyataan (statement)
4. klausa WHEN : untuk kondisi pembatasan
5. trigger body : bagian prosedur yang dituliskan pada trigger

KOMPONEN TRIGGER

Komponen dari sebuah trigger ada 6 (enam), yaitu : trigger timing, trigger event, nama tabel, tipe trigger, klausa WHEN, dan trigger body. Berikut ini penjelasan komponen dari trigger.

Trigger timing adalah waktu kapan trigger diaktifkan. Ada tiga macam trigger timing, yaitu :

- BEFORE : trigger dijalankan sebelum DML event pada tabel
- AFTER : trigger dijalankan setelah DML event pada tabel
- INSTEAD OF : trigger dijalankan pada sebuah view. Trigger event ada 3 kemungkinan : INSERT, UPDATE atau DELETE.

Pada saat trigger event UPDATE, kita dapat memasukkan daftar kolom untuk mengidentifikasi kolom mana yang berubah untuk mengaktifkan sebuah trigger (contoh : UPDATE OF salary ...). Jika tidak ditentukan, maka perubahannya akan berlaku untuk semua kolom pada semua baris.

Tipe trigger ada 2 macam, yaitu :

- Statement : trigger dijalankan sekali saja pada saat terjadi sebuah event. Statement trigger juga dijalankan sekali, meskipun tidak ada satupun baris yang dipengaruhi oleh event yang terjadi.
- Row : trigger dijalankan pada setiap baris yang dipengaruhi oleh terjadinya sebuah event. Row trigger tidak dijalankan jika event dari trigger tidak berpengaruh pada satu baris pun.

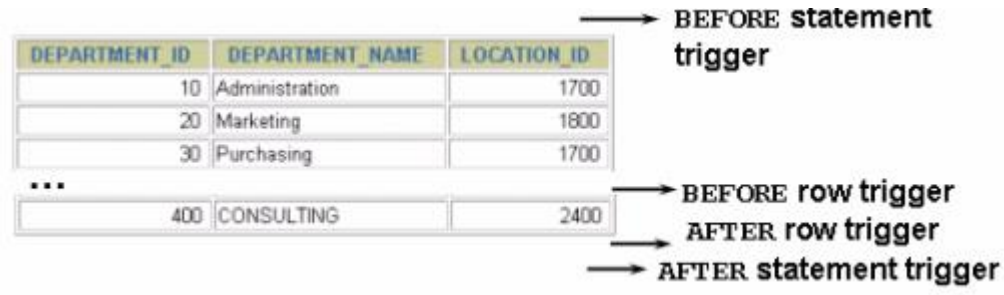
Trigger body mendefinisikan tindakan yang perlu dikerjakan pada saat terjadinya event yang mengakibatkan sebuah trigger menjadi aktif.

CONTOH PEMBUATAN TRIGGER

Contoh berikut ini akan mengaktifkan sebuah trigger pada saat sebuah baris tunggal dimanipulasi pada tabel : Misal diberikan perintah DML untuk menyisipkan baris baru ke dalam tabel sebagai berikut :

```
INSERT INTO departments (department_id, department_name, location_id)
VALUES (400, 'CONSULTING', 2400);
```

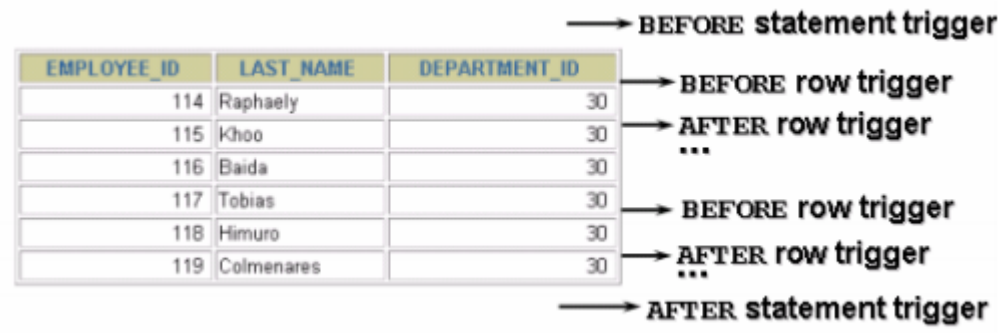
Ilustrasi dari trigger timing untuk event tersebut adalah sebagai berikut :



Jika DML statement berlaku untuk lebih dari satu baris yang ada pada tabel (multiple row), semisal :

```
UPDATE employees
  SET salary = salary * 1.1
  WHERE department_id = 30;
```

Maka ilustrasi dari trigger timing untuk event tersebut adalah sebagai berikut :



DML STATEMENT TRIGGER

Berikut ini sintak atau cara penulisan untuk pembuatan DML Statement trigger :

```
CREATE OR REPLACE TRIGGER nama_trigger
        (BEFORE | AFTER) statement ON nama_tabel
        FOR EACH ROW
DECLARE
        --- berisi deklarasi variable
BEGIN
        --- berisi statemen statement yang akan di eksekusi
END;
/
```

Berikut contoh pembuatan DML Statement trigger :

```
CREATE OR REPLACE TRIGGER secure_emp
 BEFORE INSERT ON employees
 BEGIN
  IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
    (TO_CHAR(SYSDATE,'HH24:MI') NOT BETWEEN '08:00' AND '18:00')
   THEN RAISE_APPLICATION_ERROR (-20500,'Penyisipan data pada table
EMPLOYEES hanya diperbolehkan selama jam kerja');
  END IF;
END;
/
```

Contoh trigger diatas akan membatasi penyisipan baris baru ke dalam table EMPOYEES diperbolehkan hanya pada jam kerja mulai hari Senin sampai Jum'at. Jika user menyisipkan baris baru diluar ketentuan tersebut, missal pada hari SAbtu maka akan tampil pesan kesalahan. Perintah berikut ini akan menguji trigger SECURE_EMP dengan memberikan perintah SQL berikut ini pada jam diluar jam kerja, sebagai berikut :

```
INSERT INTO employees (employee_id, last_name,first_name, email, hire_date,
job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,'IT_PROG', 4500, 60);
```

Perintah tersebut akan memberikan pesan kesalahan :

```
INSERT INTO employees (employee_id, last_name, first_name, email,
       *
ERROR at line 1:
ORA-20500: You may insert into EMPLOYEES table only during business hours.
ORA-06512: at "PLSQL.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'PLSQL.SECURE_EMP'
```

MENGKOMBINASIKAN EVENT PADA TRIGGER

Beberapa event pada trigger bisa dikombinasikan dalam sebuah trigger dengan menggunakan predikat kondisional INSERTING, UPDATING dan DELETING. Berikut ini akan dibuat trigger yang menggunakan predikat kondisional INSERTING, UPDATING dan DELETING untuk membatasi manipulasi data pada tabel EMPLOYEES hanya diperbolehkan pada setiap jam kerja mulai hari Senin sampai Jum'at.

```
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 business hours.');
  ELSIF  INSERTING THEN
    RAISE_APPLICATION_ERROR (-20500,'You may insert into
           EMPLOYEES table only during business hours.');
  ELSIF   UPDATING ('SALARY') THEN
    RAISE_APPLICATION_ERROR (-20503,'You may update
                  SALARY only during business hours.');
  ELSE
    RAISE_APPLICATION_ERROR (-20504,'You may update
        EMPLOYEES table only during normal hours.');
  END IF;
 END IF;
END;
```

ROW TRIGGER

Contoh berikut ini akan dibuat row trigger dengan timing BEFORE untuk membatasi operasi DML pada table EMPLOYEES hanya diperbolehkan untuk pegawai yang memiliki kode pekerjaan 'AD_PRES' dan 'AD_VP' serta memiliki gaji kurang dari 15000.

```
CREATE OR REPLACE TRIGGER restrict_salary
 BEFORE  INSERT OR UPDATE OF salary ON employees
  FOR EACH ROW
  BEGIN
   IF  NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
      AND :NEW.salary > 15000
   THEN
    RAISE_APPLICATION_ERROR (-20202,'Employee
                                        cannot earn this amount');
  END IF;
END;
/
```

Jika kita mencoba memberikan perintah SQL sebagai berikut, maka akan ditampilkan pesan kesalahan :

```
UPDATE employees
SET salary = 15500
WHERE last_name = 'Russell';
```

MENGGUNAKAN OLD DAN NEW QUALIFIERS

Pada Row Trigger, nilai dari kolom sebelum dan sesudah perubahan data dapat dirujuk dengan menggunakan OLD dan NEW qualifier. OLD dan NEW hanya digunakan pada Row Trigger. OLD dan NEW menggunakan prefiks (:) untuk pernyataan dalam perintah SQL. Jika qualifier ini terlibat dalam pembatasan kondisi pada klausa WHEN, maka tidak digunakan prefiks (:).

Row triggers akan menurunkan unjuk kerja jika banyak dilakukan update pada table yang cukup besar.

Contoh Trigger berikut ini menggunakan OLD dan NEW qualifier pada Row Trigger :

```
CREATE OR REPLACE TRIGGER audit_emp_values
 AFTER DELETE OR INSERT OR UPDATE ON employees
 FOR EACH ROW
BEGIN
 INSERT INTO audit_emp_table (user_name, timestamp,
   id, old_last_name, new_last_name, old_title,
   new_title, old_salary, new_salary)
 VALUES (USER, SYSDATE, :OLD.employee_id,
    :OLD.last_name, :NEW.last_name, :OLD.job_id,
    :NEW.job_id, :OLD.salary, :NEW.salary );
END;
/
```

Untuk memeriksa hasil dari pembuatan trigger diatas, diberikan perintah SQL sebagai berikut

```
INSERT INTO employees
        (employee_id, last_name, job_id, salary, ...)
 VALUES (999, 'Temp emp', 'SA_REP', 1000, ...);

UPDATE employees
 SET salary = 2000, last_name = 'Smith'
 WHERE employee_id = 999;
```

```
1 row created.
1 row updated.
```

Hasil dari perintah SQL tersebut adalah akan disimpan record perubahan pada table AUDIT_EMP_TABLE sebagai hasil dari operasi Trigger :

SELECT user_name, timestamp, ... FROM audit_emp_table

USER_NAME	TIMESTAMP	ID	OLD_LAST_N	NEW_LAST_N	OLD_TITLE	NEW_TITLE	OLD_SALARY	NEW_SALARY
PLSQL	28-SEP-01			Temp emp		SA_REP		1000
PLSQL	28-SEP-01	999	Temp emp	Smith	SA_REP	SA_REP	1000	2000

PENGGUNAAN KLAUSA WHEN PADA TRIGGE

Untuk membatasi operasi trigger hanya pada baris yang memenuhi kondisi tertentu, maka digunakan klausa WHEN. Berikut ini akan dibuat trigger pada tabel EMPLOYEES yang menghitung komisi yang diterima oleh seorang pegawai pada saat sebuah baris ditambahkan ke dalam tabel EMPLOYEES, atau pada saat dilakukan modifikasi pada gaji pegawai.

```
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;
/
```

Pada klausa WHEN, penggunaan OLD dan NEW qualifier tidak dengan prefiks (:). Untuk menggunakan NEW qualifier, gunakan BEFORE Row Trigger, jika timing BEFORE pada trigger diatas diganti dengan AFTER, maka akan didapat pesan kesalahan :

```
CREATE OR REPLACE TRIGGER derive_commission_pct*
ERROR at line 1:
ORA-04084: cannot change NEW values for this trigger type
```

PERINTAH UMUM

Berikut ini perintah-perintah umum yang digunakan pada trigger. Untuk mengaktifkan atau menonaktifkan database trigger, digunakan perintah :

ALTER TRIGGER trigger_name DISABLE | ENABLE

Untuk mengaktifkan atau menonaktifkan semua trigger yang berlaku untuk sebuah tabel, digunakan perintah :

ALTER TABLE table_name DISABLE | ENABLE ALL

Untuk melakukan kompilasi ulang sebuah trigger, digunakan perintah :

ALTER TRIGGER trigger_name COMPILE

Untuk menghapus trigger dari database, digunakan perintah :

DROP TRIGGER trigger_name

Catatan : Semua trigger yang berlaku pada sebuah tabel akan dihapus pada saat tabel tersebut dihapus dari database.

B. TUGAS PRAKTIKUM

1. Buatlah stored procedure tanpa parameter. Stored procedure ini memiliki dua buah variabel yaitu “x” dan “y”. inisialisasi isi kedua variabel tersebut dengan nilai tertentu kemudian tukar nilai “x” dan “y”. sebagai contoh nilai awal x=8 dan y=5. Diakhir proses nilai akan tertukar sehingga nilai x=5 dan y=8!
2. Buatlah stored procedure tanpa parameter untuk menampilkan department_name dan jumlah total pegawai yang ada pada setiap department!
3. Buatlah prosedur untuk menampilkan nama pegawai (FIRST_NAME , LAST_NAME) dan gaji (SALARY) dimana gaji dari pegawai tersebut lebih besar dari :BILANGAN_INPUT (BIND VARIABLE) dari tabel pegawai (EMPLOYEES) (*hint : gunakan kursor eksplisit)
4. Buatlah prosedur dengan parameter masukan EMPLOYEE_ID untuk menampilkan nama pegawai (FIRST_NAME) beserta total salary-nya! (total salary dihitung dari salary ditambah dengan persentase komisi/commission_pct dari salary) Jika commission_pct kosong maka total salary adalah tetap salary itu sendiri.
   Input : 176
   Output : total salary = 8600+(0.2*8600)=10320
   Input : 181
   Output : total salary = 3100

5. Buatlah stored function untuk menghitung pajak dari gaji yang diterima pegawai yaitu sebesar 2% dengan nilai masukan EMPLOYEE_ID. Kemuadian buat Anonymous block yang memanfaatkan function tersebut untuk menampilkan nama, id dan gaji bersih setelah dkurangi pajak dari seluruh pegawai! {sesuai dengan database HR}
6. Buatlah statement trigger yang bernama TRG_RESTRICT_DEL_EMPLOYEE untuk mencegah penghapusan data-data employees pada hari kerja (senin s/d jumat) antara jam 09.00 s/d 17.00!
7. Buatlah sebuah tabel bernama REGION_HISTORY yang memiliki 6 kolom yaitu OLD_REGION_ID, OLD_REGION_NAME, NEW_REGION_ID, NEW_REGION_NAME, CHANGE_TIME, DESCRIPTION.

Kemudian modifikasi trigger TRG_DML_REGIONS_WARNING sehingga ketika terjadi operasi DML akan tercatat nilai lama, nilai baru, waktu perubahan, dan deskripsinya (insert, update, delete)!