Bab 4
Penulisan Struktur Kontrol

Tujuan
Setelah menyelesaikan bab ini, siswa diharapkan mampu :
- Mengidentifikasi penggunaan dan macam-macam tipe struktur kontrol
- Dapat membuat perintah IF
- Menggunakan ekspresi CASE
- Membuat dan mengidentifikasi macam-macam perintah perulangan
- Menggunakan table logika
- Kontrol blok menggunakan nested loops dan label

Tujuan Secara Umum
Pada bab ini, kita mempelajari kontrol kondisional pada blok PL/SQL dengan menggunakan perintah IF dan perulangan.

Mengontrol Alur Eksekusi PL/SQL
- Alur logika dari statemen yang dijalankan dapat kita ubah dengan menggunakan perintah kondisional IF dan struktur kontrol perulangan
- Perintah IF macamnya:
  - IF-THEN-END IF
  - IF-THEN-ELSE-END IF
  - IF-THEN-ELSIF-END IF

Menggunakan GOTO bagaimanapun tidak disarankan. Penggunaan GOTO digunakan berdasarkan penandaan atau pemberian label pada blok SQL seperti pada contoh berikut.

Contoh
```
BEGIN ... <<update_row>>
  BEGIN
    UPDATE employees ...
  END  update_row; ...
  GOTO update_row; ...
END;
```

Perintah IF
Cara penulisan :

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

Jika Pegawai dengan nama Gietz, diset Manager ID = 102.

```
IF UPPER(v_last_name) = 'GIETZ' THEN
  v_mgr := 102;
END IF;
```

Perintah IF Sederhana
Untuk contoh berikut ini, jika variable v_name = Vargas, maka :

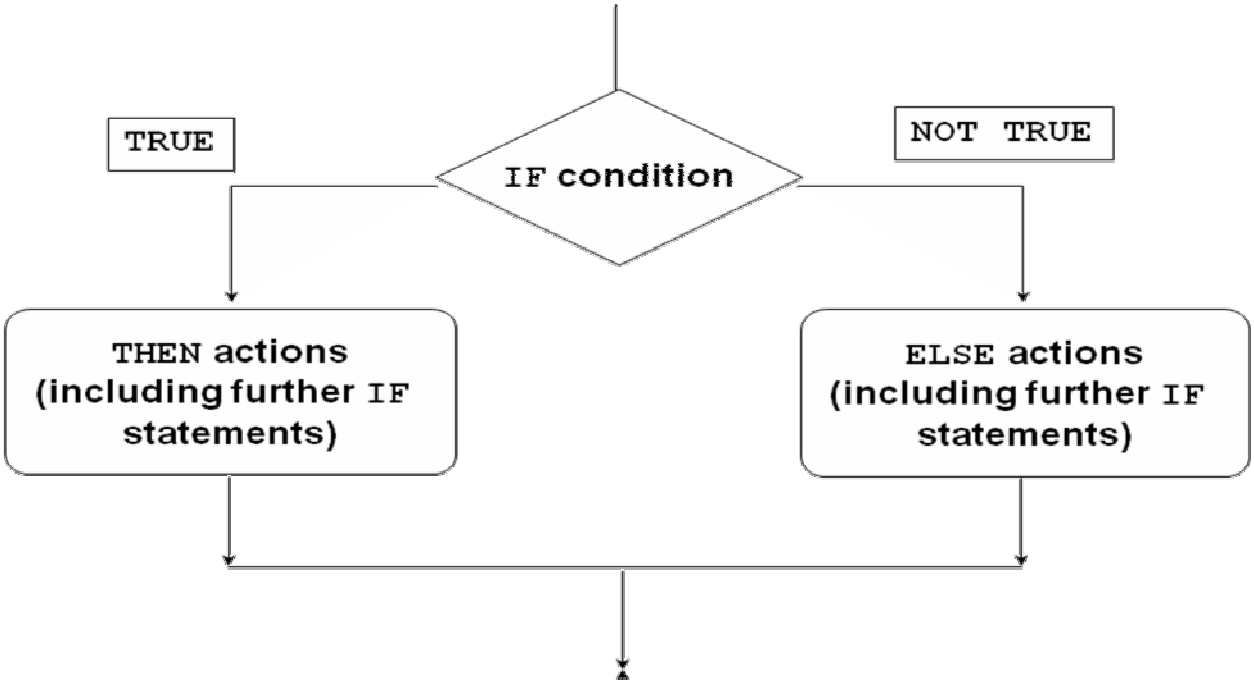
- Set job ID = SA_REP
- Set department number = 80

```
. . .
IF v_ename      = 'Vargas' THEN
   v_job        := 'SA_REP';
   v_deptno     := 80;
END IF;
. . .
```

Perintah IF Campuran
Perintah IF dapat menggabungkan kondisi yang diperiksa dengan mengguakan operator logika seperti AND dan NOT.
Untuk contoh berikut ini, jika v_name=Vargas dan gajinya lebih dari 6500, maka :
Set department number = 60.

```
. . .
IF v_ename  = 'Vargas' AND salary > 6500 THEN
   v_deptno := 60;
END IF;
. . .
```

Alur Eksekusi IF-THEN-ELSE



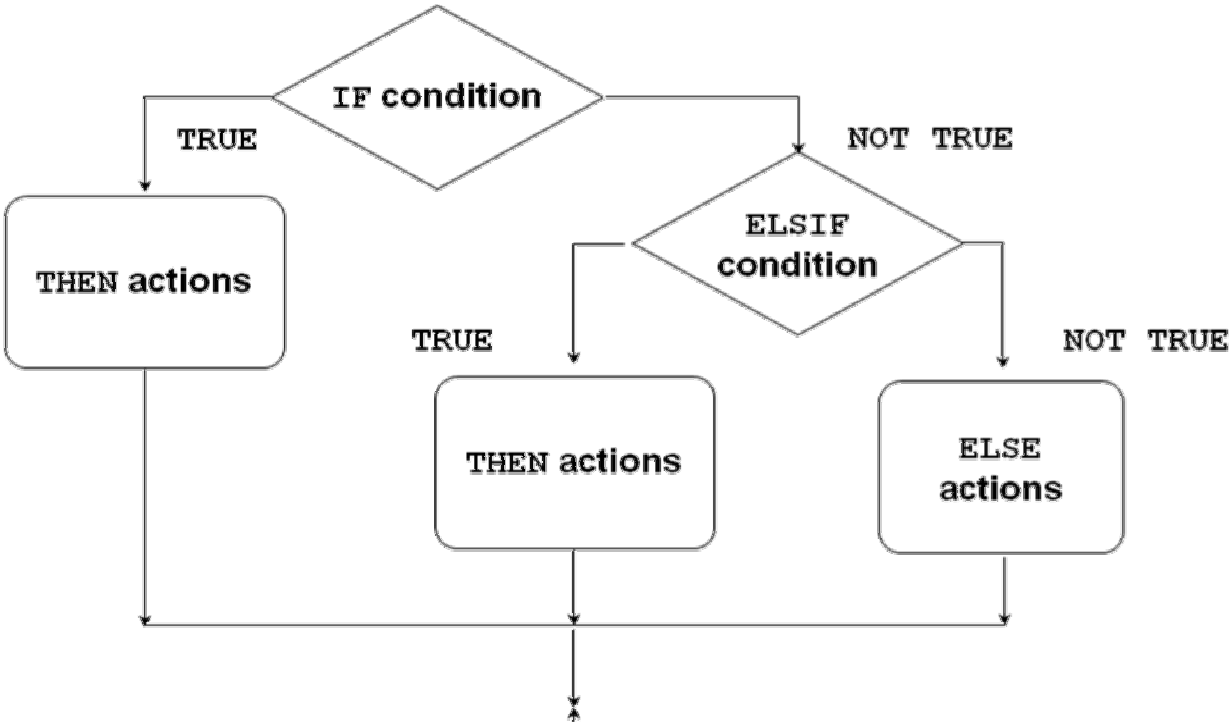
Perintah IF-THEN-ELSE
Untuk contoh berikut, set variable Boolean v_five_years = TRUE jika hire date lebih dari 5 tahun jika tidak variable diset FALSE.

```
DECLARE
   v_hire_date  DATE := '12-Dec-1990';
   v_five_years BOOLEAN;
BEGIN
. . .
IF MONTHS_BETWEEN(SYSDATE,v_hire_date)/12 > 5 THEN
   v_five_years := TRUE;
ELSE
   v_five_years := FALSE;
END IF;
...
```

Contoh Perintah IF-THEN-ELSE
Pada contoh diatas, fungsi MONTHS_BETWEEN digunakan untuk mencari selisih bulan antara tanggal sekarang dengan tanggal pegawai mulai bekerja. Karena perbedaannya dalam bulan, maka untuk mencari berapa tahun pegawai sudah bekerja hasilnya dibagi dengan 12.
Untuk contoh berikut. Periksa nilai pada variable v_ename. Jika nilainya = King, maka set variable v_job = AD_PRES. Tapi jika tidak, set variable v_job = ST_CLERK.

```
IF v_ename = 'King' THEN
  v_job := 'AD_PRES';
ELSE
  v_job := 'ST_CLERK';
END IF;
```

Alur Eksekusi Perintah IF-THEN-ELSIF



Alur Eksekusi Perintah IF-THEN-ELSIF

Jika kondisi pertama false atau null, maka klausa ELSIF akan memeriksa kondisi yang lain. Perintah IF dapat terdiri dari sejumlah klausa ELSIF; sedangkan klausa ELSE diletakkan pada bagian akhir. Perhatikan contoh berikut, untuk menentukan bonus yang diterima pegawai berdasarkan nomer departemennya :

```
IF v_deptno = 10 THEN
  v_bonus := 5000;
ELSIF v_deptno = 80 THEN
  v_bonus := 7500;
ELSE
  v_bonus := 2000;
END IF;
```

Perintah IF-THEN-ELSIF

Sedapat mungkin, gunakan klausa ELSIF daripada menggunakan perintah nested IF. Karena pengkodeannya lebih mudah dibaca dan dipahami.

Ekspresi CASE

- Ekspresi CASE menguji suatu variable selector diantara beberapa alternative ekspresi
- Jika salah satu dari ekspresi memberikan nilai TRUE, maka perintah CASE akan mengembalikan hasilnya.

```
CASE selector
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
  [ELSE resultN+1;]
END;
```

Contoh Ekspresi CASE
Pada contoh ini, ekspresi CASE digunakan untuk memeriksa nilai dari v_grade, dan memberikan hasilnya pada variable v_appraisal.

```
SET SERVEROUTPUT ON
DECLARE
   v_grade CHAR(1) := UPPER('&p_grade');
   v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal :=
      CASE v_grade
         WHEN 'A' THEN 'Excellent'
         WHEN 'B' THEN 'Very Good'
         WHEN 'C' THEN 'Good'
         ELSE 'No such grade'
      END;
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || '
                      Appraisal ' || v_appraisal);
END;
/
```

Perintah CASE merupakan alternative yang baik untuk menggantikan fungsi DECODE yang ada pada iSQL*PLUS.
Pada saat mengevaluasi suatu ekspresi, apapun yang berhubungan dengan nilai null akan memberikan hasil NULL pula. Contoh :
p := NULL;
q := NULL;
p || q hasilnya NULL.

Menangani Nilai Null
Pada saat bekerja dengan nilai null, anda harus menyadari aturan berikut :

- Pembandingan yang melibatkan nilai NULL hasilnya akan NULL pula
- Melibatkan operator logika dengan nilai NULL akan menghasilkan nilai NULL pula

- Pada perintah kondisional, jika suatu kondisi bernilai NULL, maka perintah setelahnya tidak akan dijalankan.

Menangani Nilai Null

Contoh berikut :

```
x := 5;
y := NULL;
...
IF x != y THEN  -- menghasilkan nilai NULL, bukan TRUE
   Perintah berikutnya;  -- tidakakan dijalankan
END IF;
```

Begitu juga pada contoh berikut :

```
a := NULL;
b := NULL;
...
IF a = b THEN  -- menghasilkan NULL, bukan TRUE
   Perintah berikutnya;  -- tidak akan dijalankan
END IF;
```

Tabel Logika

Berikut table untuk operator logika AND, OR dan NOT :

AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Kondisi Boolean

Apa nilai dari variable V_FLAG, untuk setiap kemungkinan nilai v_reorder_flag dan v_available_flag seperti yang ada pada table ?

```
v_flag := v_reorder_flag AND v_available_flag;
```

V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	?
TRUE	FALSE	?
NULL	TRUE	?
NULL	FALSE	?

Jawaban :

1. TRUE
2. FALSE
3. NULL
4. FALSE

Kontrol Iteratif: Perintah LOOP

- Perintah Loop (perulangan) mengerjakan kumpulan perintah secara berulang.
- Ada tiga macam loop :
  - Basic loop
    Perulangan tanpa kondisi
  - FOR loop
    Perulangan berdasarkan nilai variable counter (sudah diketahui berapa kali akan dikerjakan)
  - WHILE loop
    Perulangan berdasarkan suatu kondisi

Perintah EXIT digunakan untuk keluar dari Loop

Basic Loops

Cara Penulisan :

```
LOOP
  statement1;
  . . .
  EXIT [WHEN condition];
END LOOP;
```

```
condition     is a Boolean variable or
              expression (TRUE, FALSE, or NULL);
```

Basic Loops

Merupakan bentuk paling sederhana pada Loop. Baris di dalam tubuh perulangan akan dikerjakan berulang kali sampai ditemukan perintah EXIT yang dapat menghentikan loop.

Perintah EXIT

Bentuk perintah EXIT pada basic Loop adalah EXIT WHEN *kondisi;*
Jika kondisi yang disyaratkan terpenuhi, maka akan keluar dari perulangan.

Basic Loops

Contoh penggunaan Basic Loop :

```
DECLARE
  v_country_id    locations.country_id%TYPE := 'CA';
  v_location_id   locations.location_id%TYPE;
  v_counter       NUMBER(2) := 1;
  v_city          locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_location_id FROM locations
  WHERE country_id = v_country_id;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + v_counter),v_city, v_country_id);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```

Loop diatas akan menambahkan tiga baris (record) baru ke dalam table locations. Dengan nilai location_id = nilai maksimum dari location_id yang ada di table ditambah dengan 1. Jadi missal nilai maksimum location_id=100, maka akan ditambahakan :

101 Montreal CA
102 Montreal CA
103 Montreal CA

Catatan

Selain digunakan pada Basic Loop, kita juga dapat menggunakan perintah EXIT pada tipe Loop yang lain seperti WHILE Loop dan FOR loop.

WHILE Loops
Cara Penulisan :

```
WHILE condition LOOP
   statement1;
   statement2;
   . . .
END LOOP;
```

Gunakan perintah WHILE loop untuk mengulang sekumpulan perintah selama kondisi yang ditentukan bernilai TRUE.

WHILE Loops
Berikut contoh penggunaan WHILE Loop :

```
DECLARE
  v_country_id      locations.country_id%TYPE := 'CA';
  v_location_id     locations.location_id%TYPE;
  v_city            locations.city%TYPE := 'Montreal';
  v_counter         NUMBER  := 1;
BEGIN
  SELECT MAX(location_id) INTO v_location_id FROM locations
  WHERE country_id = v_country_id;
  WHILE v_counter <= 3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + v_counter), v_city, v_country_id);
    v_counter := v_counter + 1;
  END LOOP;
END;
/
```

Program diatas, hasilnya sama dengan program sebelumnya yang menggunakan Basic Loop.

FOR Loops
Cara Penulisan :

```
FOR counter IN [REVERSE]
    lower_bound..upper_bound LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- FOR loop digunakan jika berapa kali pengulangan telah diketahui
- Variable counter pada FOR loop dideklarasikan secara implicit (tidak perlu dideklarasikan)
- Nilai dari variable counter otomatis akan di-increment setiap kembali ke awal perulangan

- 'lower_bound .. upper_bound' menentukan range dari variable counter

Perintah REVERSE tidak berlaku dalam hal ini. Jika lower bound lebih besar nilainya dari upper bound, maka perulangan akan dijalankan sekali. Jika nilai lower bound sama dengan upper bound, maka perulangan juga dijalankan sekali, contohnya :
FOR i IN 3..3 LOOP *statement1;* END LOOP;

FOR Loops
Program berikut ini hasilnya sama dengan program sebelumnya yang menggunakan Basic Loop dan WHILE Loop.

```
DECLARE
  v_country_id    locations.country_id%TYPE := 'CA';
  v_location_id   locations.location_id%TYPE;
  v_city          locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_location_id
    FROM locations
    WHERE country_id = v_country_id;
  FOR i IN 1..3 LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + i), v_city, v_country_id );
  END LOOP;
END;
/
```

FOR Loops
Petujuk :

- counter hanya digunakan untuk menandai perulangan pada loop dan tidak didefinisikan diluar loop
- Tidak menggunakan variable counter sebagai target dari suatu penandaan (assignment). Misal : counter := v_blablabla;
- Pada saat menulis FOR loop, lower dan upper bounds tidak perlu harus numeric, bis juga ekspresi yang nantinya akan dikonversi ke nilai numeric.

Example

```
DECLARE
  v_lower    NUMBER := 1;
  v_upper    NUMBER := 100;
BEGIN
  FOR i IN v_lower..v_upper LOOP
  ...
```

```
  END LOOP;
END;
```

Petunjuk Umum Penggunaan Loop

- Gunakan basic loop jika perintah dalam Loop harus dijalankan sedikitnya sekali.
- Gunakan WHILE loop jika setiap kali perulangan ada kondisi yang harus dievaluasi nilainya pada setiap iterasi.
- Gunakan FOR loop jika jumlah iterasi diketahui

Nested Loopsda n Labelspada Basic Loop

- Loop dapat di-nested untuk beberapa level
- Gunakan level untuk membedakan blok dan loop
- Keluar dari outer loop dapat dilakukan dengan perintah EXIT yang merefer ke label.

Nested Loops and Labels

Berikut penggunaan Nested Loop dan label :

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
```

Nested Loopsda n Labelspada Basic Loop

Pada contoh diatas, ada dua loop. Yang diidentifikasi sebagai <<Outer_Loop>> dan <<Inner_Loop>>. identifiers atau nama label yang menandai awal Loop ditempatkan sebelum kata LOOP dengan diapit oleh delimiters (<<*label*>>). Nama label dicantumkan setelah perintah END LOOP untuk memperjelas alur program.

Catatan

Selain digunakan pada loop, label juga dapat digunakan pada nested blocks , untuk menghindari ambiguitas dari variable.

Ringkasan

Pada bab ini telah dipelajari : Perubahan alur logika dengan menggunakan Struktur Kontrol

- Kondisional (perintah IF)
- Ekspresi CASE

- Loops:
  - Basic loop
  - FOR loop
  - WHILE loop
- Perintah EXIT

Topik Latihan Bab 4

Topik latihan bab 4 meliputi :

- Pembuatan program dengan perintah kondisional menggunakan IF
- Pembuatan program perulangan menggunakan struktur Loop

Latihan 4

1. Jalankan skrip lab04_1.sql untuk membuat table MESSAGES. Tulis blok PL/SQL untuk menyisipkan bilangan ke dalam table MESSAGES.
   a. Masukkan bilangan 1 sampai 10, kecuali 6 dan 8.
   b. Jalankan Commit sebelum akhir dari block.
   c. Tampilkan data pada table MESSAGES untuk melihat hasilnya.

2. Buat blok PL/SQL untuk menghitung nilai komisi pegawai berdasarkan gajinya.
   a. Gunakan perintah DEFINE untuk menampung nomer pegawai (employee ID). Pass the value to the PL/SQL block through a *i*SQL*Plus substitution variable.
      DEFINE p_empno = 100
   b. Jika gaji pegawai kurang dari $5,000, nilai bonus yang diterima pegawai = 10% dari gaji
   c. Jika gaji pegawai antara $5,000 dan $10,000, maka bonus yang diterima pegawai 15% dari gaji.
   d. Jika gaji pegawai melebihi $10,000, maka bonus yang diterima pegawai 20% dari gaji.
   e. Jika gaji pegawai NULL, bonus yang diterima = 0.

   Catatan: Sertakan SET VERIFY OFF pada program yang anda buat

3. Buat table EMP yang merupakan replica (kopi) dari table EMPLOYEES. Anda bisa melakukannya dengan menjalankan skrip lab04_3.sql. Tambahkan kolom baru yaitu STARS, bertipe VARCHAR2 dengan panjang 50 pada table EMP untuk menyimpan nilai asterisk (*).

4. Buat blok PL/SQL yang memberikan hadiah pada pegawai dengan menampilkan tanda asterisk pada kolom/ field STARS untuk setiap $1000 dari gaji yang dimiliki oleh pegawai. Simpan program anda pada file dengan nama p4q4.sql dengan mengklik tombol Save Script.
   a. Gunakan perintah DEFINE untuk mendefinisikan table yang menyimpan nomer pegawai (employee ID). Lewatkan nilai dari blok PL/SQL melalui variable substitusi *i*SQL*Plus.
      DEFINE p_empno=104
   b. Inisialisasi variable v_asterisk = NULL.
   c. Tambahkan asterisk ke string untuk setiap $1000 dari jumlah gaji pegawai. Misal, jika pegawai punya gaji $8000, maka jumlah asterisk-nya = 8. Jika pegawai punya gaji $12500, maka jumlah asterisk-nya = 13.

d. Update field / kolom STARS untuk menampung jumlah asterisk sesuai jumlah gaji pegawai
e. Berikan perintah Commit.
f. Jalankan program untuk nilai berikut :
   DEFINE p_empno=174
   DEFINE p_empno=176
g. Tampilkan baris data dari table EMP untuk memeriksa apakah program sudah dijalankan dengan benar.

Catatan: Sertakan SET VERIFY OFF pada program yang anda buat