

Laboratorium Komputer

# Oracle Database Programming Practicum Module

Adrianus Wijaya

2014

Used For 3<sup>rd</sup> Diploma Database Programming Practicum.



STMIK STIKOM Surabaya

## Forewords

### 前書き

I know you all already passed this course in the regular class, so let's just review again what you all already studied and may be, you'll find something new in this practicum class.

May be, there are some of you who dislike this course because of something, may be the lecturer, difficulties, etc; but deal with it! you took this course by yourself and I know you want to pass this course, so once again, DEAL WITH IT!

May be some of you thinks "I don't need this course, because is useless for my future", well, there are no disadvantage when it comes to learning a new knowledge, even the smallest may be useful in the future, and after all, I DON'T CARE whether you like it or not, my job is to help you to pass, so DEAL WITH IT!

元気があるならなんでもできる！

"If you have the spirit, you can do anything!"

## Table Of Content

### コンテンツの表

<b>Forewords</b>	<b><i>i</i></b>
<b>Table Of Content</b>	<b><i>ii</i></b>
<b>Used Database Structure</b>	<b><i>vi</i></b>
<b>Before We Start</b>	<b>1</b>
<b>Stage 1: Introduction To PL/SQL</b>	<b>1</b>
<b>What's PL/SQL</b>	<b>2</b>
<b>PL/SQL Block Structure</b>	<b>3</b>
<b>PL/SQL Variables</b>	<b>4</b>
Type Data	5
Variable Declaration	7
<b>Control Structures</b>	<b>12</b>
Conditional Flow	12
Iteration Flow	17
<b>Nested PL/SQL Blocks</b>	<b>23</b>
<b>Practice</b>	<b>26</b>
<b>Stage 2: Cursor-ing a Lot of Data</b>	<b>27</b>
<b>About Cursors</b>	<b>28</b>
<b>Implicit Cursor</b>	<b>28</b>

---

<b>Explicit Cursor</b>	<b>29</b>
<b>Cursor And Record</b>	<b>35</b>
Record	35
Cursor and Record	39
<b>Cursor With Parameter</b>	<b>42</b>
<b>Practice</b>	<b>45</b>
<b>Stage 3: Procedure</b>	<b>46</b>
<b>Subprogram and Procedure</b>	<b>47</b>
Subprogram	47
Procedure	49
<b>Using Procedure</b>	<b>50</b>
Creating Procedure	50
Invoking Procedure	52
Removing Procedure	53
<b>Parametered Procedure</b>	<b>53</b>
Parameter	53
Using Parametered Procedure	54
<b>Practice</b>	<b>61</b>
<b>Stage 4: Stored Function</b>	<b>62</b>
<b>About Stored Function</b>	<b>63</b>
<b>Using Stored Function</b>	<b>65</b>
Creating Stored Function	65
Invoking Stored Function	67
Removing Stored Function	71
<b>Table Function</b>	<b>71</b>

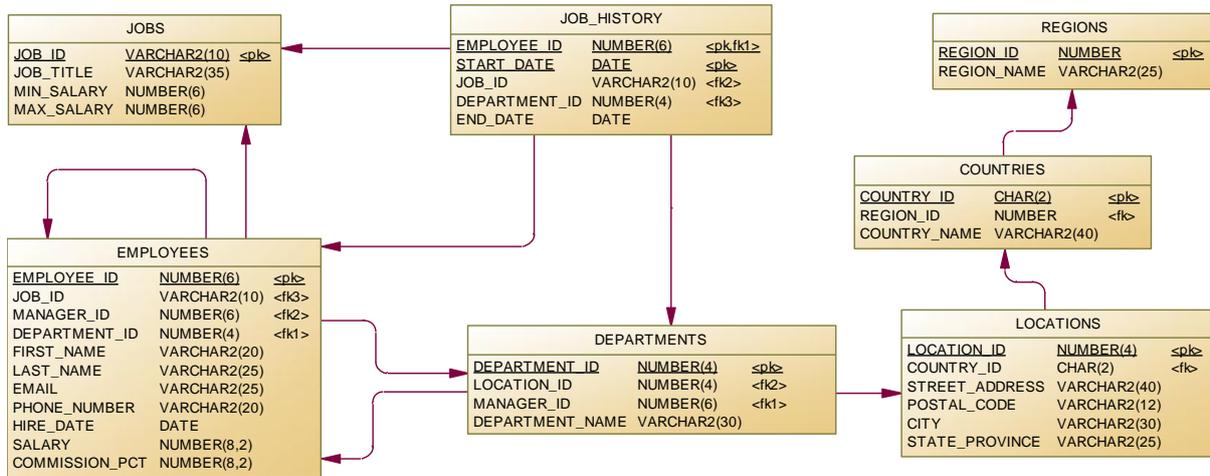
---

Non-Pipelined Table Function _____	72
Pipelined Table Function _____	75
<b>Practice _____</b>	<b>79</b>
<b><i>Stage 5: Oracle Developer (Single Block Form) _____</i></b>	<b><i>80</i></b>
<b>About Oracle Developer _____</b>	<b>81</b>
<b>Creating a Block _____</b>	<b>81</b>
Creating Block Based On Table _____	81
Creating Block Manually _____	92
<b>Simple Use Of The Form _____</b>	<b>97</b>
Alert _____	104
<b>Practice _____</b>	<b>109</b>
<b><i>Stage 6: Oracle Developer (Multiple Blocks and LOV)</i></b>	<b><i>110</i></b>
<hr/>	
<b>LOV (List Of Values) _____</b>	<b>111</b>
Creating LOV Through The Wizard _____	111
Using LOV in PL/SQL Code _____	124
<b>Working with Multiple-Block _____</b>	<b>126</b>
Form Design Example _____	127
Code Example _____	132
<b>Practice _____</b>	<b>136</b>
<b><i>Stage 7: Oracle Developer (Program Unit + Multiple Blocks II) _____</i></b>	<b><i>137</i></b>
<b>Using Program Unit _____</b>	<b>138</b>
<b>Manipulating Data Inside Detail Block _____</b>	<b>142</b>

---

<b>Practice</b>	<b>148</b>
<b>Stage 8: Report and Graphic</b>	<b>150</b>
<b>Creating a Report</b>	<b>151</b>
Creating A Parametered Report	162
Call a Report From a Form	169
<b>Creating A Graphic</b>	<b>171</b>
<b>Practice</b>	<b>176</b>
<b>Bibliography</b>	<b>177</b>
<b>Bonus Track</b>	<b>178</b>

## Used Database Structure



## Before We Start

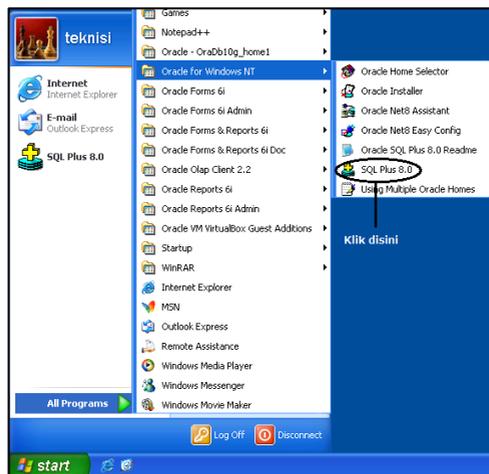
- ✓ *Log in* yang digunakan untuk menggunakan Oracle SQL Plus dan Oracle Form Developer 6i adalah sebagai berikut:

User Name: P[NIM Panjang]

Password: P[NIM Panjang]

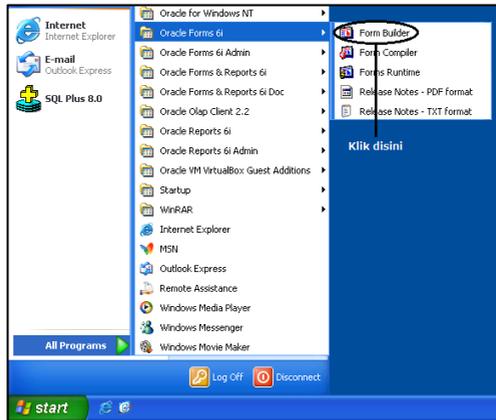
Host String: prakt

- ✓ Untuk mengakses Oracle SQL PLUS 8.0, klik pada [START Menu]-[All Programs]-[Oracle For Windows NT] – [SQL Plus 8.0].



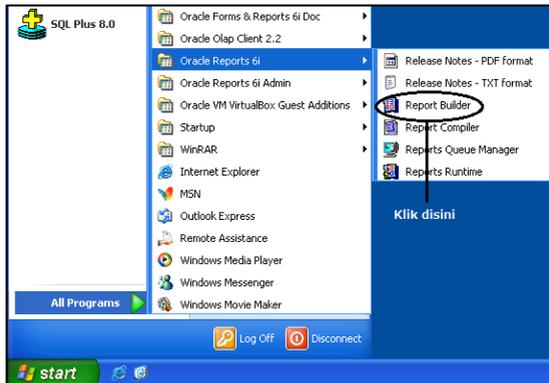
Gambar i. Akses SQL Plus

- ✓ Untuk mengakses *form builder*, klik pada [START Menu]-[All Programs]-[Oracle Forms 6i] – [Form Builder]



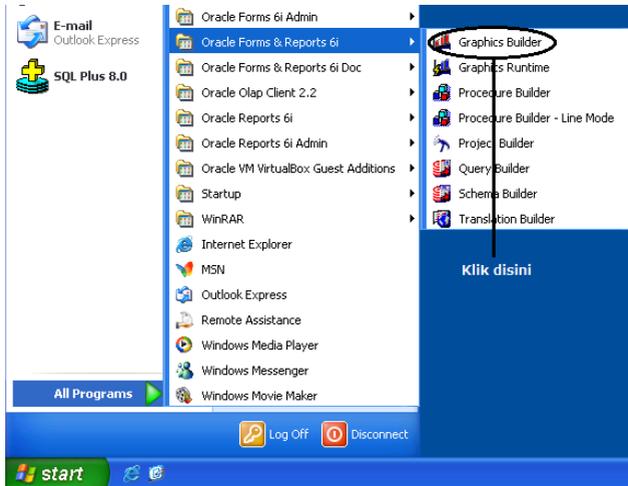
Gambar ii. Akses *Form Builder*

- ✓ Untuk mengakses *form builder*, klik pada [START Menu]-[All Programs]-[Oracle Reports 6i] – [Report Builder]



Gambar iii. Akses *Report Builder*

- ✓ Untuk mengakses *form builder*, klik pada [START Menu]-[All Programs]-[Oracle Forms & Reports 6i] – [Graphics Builder]



Gambar iv. Akses *Graphic Builder*

## Stage 1: Introduction To PL/SQL

### *PL/SQL の紹介*

“What’s Important is whether you’re brave enough to take a single step to start, or not.”

人

What you’ll Learn:

What’s PL/SQL

Benefits of PL/SQL

PL/SQL Block Structure

PL/SQL Variables

Control Structure

## What's PL/SQL



PL/SQL adalah singkatan dari *Procedural Language Extension* to SQL yang merupakan bahasa akses data standar milik Oracle untuk basis data relasional, yang mampu mengintegrasikan konstruksi prosedural dan SQL dengan lancar, dan menyediakan sebuah ekstensi pemrograman pada SQL.

PL/SQL mendefinisikan sebuah struktur *block* untuk menuliskan kode, sehingga Pemeliharaan (*maintaining*) dan *debugging* kode akan lebih mudah, dan aliran dan eksekusi dari sebuah *program unit* dapat lebih mudah dipahami. PL/SQL menyediakan konstruksi procedural seperti:

- ✓ Variabel, *constant*, dan tipe,
- ✓ Struktur kendali, seperti perulangan dan percabangan,
- ✓ *Program unit* yang dapat digunakan berulang-ulang.

Keuntungan dari PL/SQL antara lain:

- ✓ Pengembangan program secara termodulasi
- ✓ Integrasi dengan *Oracle Tools*,
- ✓ *Portability*
- ✓ *Exception Handling*

## PL/SQL Block Structure

Sebuah blok PL/SQL terdiri atas 3 bagian, yaitu:

1. *Declarative*

Bagian *declarative* dimulai dengan *keyword* `DECLARE` dan berakhir pada saat bagian *executable* dimulai. Bagian ini berisi deklarasi dari semua variabel, *constant*, *cursor*, dan *user-defined exception* yang direferensi pada bagian *executable* dan bagian *exception handling*. bagian ini bersifat opsional pada penulisan blok PL/SQL.

2. *Executable* (wajib)

Bagian *executable* dimulai dengan *keyword* `BEGIN` dan diakhiri dengan *keyword* `END`. Bagian *executable* dari sebuah blok PL/SQL dapat memuat sejumlah blok PL/SQL lain, perintah SQL untuk mengambil data dari database, dan perintah PL/SQL untuk memanipulasi data pada *block*.

3. *Exception Handling*

Bagian *exception handling* termasuk dalam bagian *executable* dan dimulai dengan *keyword* `EXCEPTION`. Bagian *exception handling* menspesifikasikan tindakan yang akan dilakukan ketika terjadi error dan kondisi

abnormal pada bagian eksekusi. Bagian ini juga bersifat opsional pada penulisan blok PL/SQL.

Susunan kode dari sebuah blok PL/SQL adalah sebagai berikut:

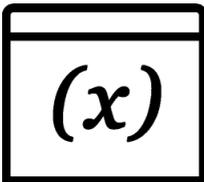
```
[DECLARE]

BEGIN
--statements

[EXCEPTION]

END;
```

## PL/SQL Variables



Variabel digunakan terutama untuk menyimpan data dan manipulasi nilai yang tersimpan di dalamnya. Variabel dapat menyimpan obyek PL/SQL apapun seperti variabel, *type* (tipe), *cursor*, dan *subprogram*.

pada saat dideklarasikan, variabel harus diberi nama (atau *identifier*). Untuk memberi nama variabel (atau menuliskan *identifier*), terdapat beberapa peraturan yang harus diikuti, yaitu:

- ✓ Harus dimulai dengan sebuah huruf.

- ✓ Dapat memuat huruf atau angka
- ✓ Dapat memuat karakter khusus, contohnya simbol dollar (\$) atau underscore (\_).
- ✓ Terbatas pada 30 karakter saja.
- ✓ Tidak boleh menggunakan *keyword* yang sudah digunakan pada Oracle.

### Tipe Data

Setiap *constant*, variabel, dan parameter memiliki sebuah tipe data (atau tipe), yang menjelaskan format penyimpanan, batasan, dan *range* nilai yang valid. Sebuah variabel, *constant*, atau parameter yang hanya bisa menampung sebuah nilai saja disebut variabel dengan tipe data *scalar*. Beberapa tipe data *scalar* adalah:

- ✓ CHAR (*[maximum\_length]*)

Jika *maximum\_length* tidak dituliskan/didefinisikan pada saat deklarasi variabel, maka panjang *default* adalah 1, dan sifat panjang karakter pada tipe data CHAR adalah tetap (jika *maximum\_length* didefinisikan 3 dan kolom diisi sebuah nilai dengan panjang 2, maka akan muncul spasi untuk mengisi sisa panjang yang tidak diisi.).

- ✓ `VARCHAR2 (maximum_length)`  
*Maximum\_length* harus didefinisikan pada saat deklarasi variabel dan sifat panjang karakter pada tipe data `VARCHAR2` adalah varian (jika *maximum\_length* didefinisikan 3 dan kolom diisi sebuah nilai dengan panjang 2, tidak akan terjadi apa-apa).
- ✓ `NUMBER [(precision, scale)]`  
*Precision* adalah jumlah digit yang dapat tampung oleh variabel, dengan *range* dari 1 sampai 38, sedangkan *scale* adalah jumlah dari digit sesudah koma (di belakang koma) dari panjang *precision* yang sudah dituliskan sebelumnya, dengan *range* dari -84 sampai 127 (jika *precision* didefinisikan 3 dan *scale* didefinisikan 2, maka variabel hanya bisa menerima nilai angka dengan 1 digit sebelum koma dan 2 digit sesudah koma). *Precision* dan *scale* bersifat opsional, sehingga tidak harus didefinisikan pada saat deklarasi variabel.
- ✓ `BOOLEAN`  
Hanya dapat berisi nilai `TRUE`, `FALSE`, atau `NULL`.

## ✓ DATE

Untuk menyimpan tanggal dan jam. *Range* dari nilai DATE antara 4712 sebelum masehi dan 9999 sesudah masehi.

### Variable Declaration

Susunan kode untuk mendeklarasikan sebuah variabel PL/SQL adalah sebagai berikut:

```
Identifier [CONSTANT] datatype [NOT NULL] [:=  
|DEFAULT expr];
```

Pada susunan kode ini:

<i>Syntax</i>	Penjelasan
<i>identifier</i>	Nama variabel
CONSTANT	Membuat nilai pada variabel tidak dapat dirubah. (nilai pada variabel harus langsung diinisiasi/diisi pada saat deklarasi)
<i>datatype</i>	Adalah tipe data dari variabel (bisa tipe data <i>scalar</i> , <i>composite</i> , <i>reference</i> , atau LOB)
Not NULL	Membuat variabel tidak bisa kosong/ dibuat bernilai NULL. (nilai pada variabel harus langsung diinisiasi/diisi pada saat deklarasi)
<i>Expr</i>	Adalah ekspresi PL/SQL yang dapat berupa ekspresi literal, variabel lain, atau ekspresi

<i>Syntax</i>	Penjelasan
	yang mengandung operator dan <i>function</i> .

Contoh deklarasi variabel:

```
DECLARE
  emp_hiredate    DATE;
  emp_deptno     NUMBER(2) NOT NULL := 10;
  location        VARCHAR2(13) := 'Atlanta';
  c_comm         CONSTANT NUMBER := 1400;
  . . .
```

Ketika sebuah variabel dideklarasikan dengan untuk menampung nilai sebuah kolom pada table, variabel tersebut harus dideklarasikan dengan tipe data dan *precision/maximum\_length* yang tepat. Untuk memudahkan hal ini, variabel dapat dideklarasikan dengan menggunakan atribut `%TYPE`. `%TYPE` membuat variabel dideklarasikan menurut variabel lain yang sudah dideklarasikan atau menurut kolom pada tabel. Susunan kode untuk mendeklarasikan variabel dengan atribut `%TYPE` adalah sebagai berikut:

```
Identifier          table.column_name%TYPE;
```

Contoh deklarasi variabel dengan menggunakan atribut %TYPE:

```
DECLARE
    emp_lname      employees.last_name%TYPE;
    balance        NUMBER(7,2);
    min_balance    balance%TYPE := 100;
    . . .
```

Contoh penggunaan variabel adalah sebagai berikut:

a. Memasukkan dan menampilkan nilai variabel.

```
DECLARE
    fname VARCHAR2(50);
BEGIN
    Fname := 'Adrianus Wijaya';
    DBMS_OUTPUT.PUT_LINE('My name is '||fname);
END;
```

b. Operasi perhitungan sederhana.

```
DECLARE
    val1 number(2) := 3;
    val2 number(2) := 30;
    result number(2);
BEGIN
    result := val1 + val2;
    DBMS_OUTPUT.PUT_LINE('The result is '||result);
END;
```

Catatan:

DBMS\_OUTPUT.PUT\_LINE adalah *keyword* yang digunakan mencetak/menuliskan nilai dalam variabel atau mencetak/menuliskan teks.

Selain menggunakan tanda “:=”, variabel juga dapat diisi melalui sebuah *query* SELECT untuk mengisi variabel dengan nilai dari sebuah kolom atau tabel. Susunan kode untuk mengisi variabel melalui *query* SELECT adalah sebagai berikut:

```
SELECT  select_list
INTO    {variable_name[, variable_name]. . .
        | record_name}
FROM    table
[WHERE  condition];
```

Pada susunan kode ini:

<i>Syntax</i>	Penjelasan
<i>select_list</i>	Adalah daftar dari kolom yang akan diambil nilainya dan dapat meliputi <i>row functions</i> , <i>group functions</i> , atau Ekspresi SQL.
<i>Variable_name</i>	Adalah variabel bertipe scalar yang akan menampung nilai yang diambil
<i>record_name</i>	Adalah obyek PL/SQL RECORD yang akan menampung nilai yang diambil
<i>table</i>	Nama dari tabel pada <i>database</i> yang diambil datanya.
<i>condition</i>	Bisa terdiri atas nama kolom, ekspresi, <i>constant</i> , dan operasi perbandingan yang meliputi variabel atau PL/SQL <i>constant</i> .

Yang perlu diperhatikan pada saat mengambil data dalam PL/SQL:

- ✓ Akhiri semua perintah SQL dengan “;”.
- ✓ Setiap nilai yang diambil harus dimasukkan ke dalam sebuah variabel dengan menggunakan klausa INTO.
- ✓ Klausa WHERE bersifat opsional, tetapi ketika klausa INTO digunakan, klausa WHERE harus digunakan agar perintah SELECT hanya mengembalikan 1 baris saja.

Contoh penggunaan query SELECT untuk mengisi nilai variabel adalah sebagai berikut:

- a. Mengambil nama lengkap (`first_name` dan `last_name`) dari karyawan dengan `employee_id` 100 dan dimasukkan ke dalam sebuah variabel.

```
DECLARE
    full_name varchar2(50);
BEGIN
    SELECT first_name||' '||last_name
    INTO full_name
    FROM EMPLOYEES
    WHERE employee_id = 100;
END;
```

- b. Ambil dan tampilkan nama departemen dan jumlah karyawan dari departemen dengan ID = 100.

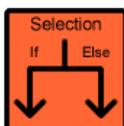
```
DECLARE
    dept_name departments.department_name%TYPE;
    nofEmp number(5,2);
BEGIN
    SELECT d.department_name, count(e.employee_id)
    INTO dept_name, nofEmp
    FROM employees e, departments d
    WHERE e.department_id = d.department_id AND
           d.department_id = 100
    GROUP BY d.department_name;
    DBMS_OUTPUT.PUT_LINE('Department ID: 100');
    DBMS_OUTPUT.PUT_LINE('Department Name: '|| dept_name);
    DBMS_OUTPUT.PUT_LINE('Number Of Employees: '||nofEmp);
EXCEPTION WHEN no_data_found THEN
    DBMS_OUTPUT.PUT_LINE('There is no department with ID 100');
END;
```

## Control Structures



Aliran logika dalam sebuah blok PL/SQL dapat dirubah dengan sejumlah struktur kontrol.

## Conditional Flow



Alir kondisional mengendalikan jalan sebuah aliran eksekusi program berdasarkan kondisi yang sudah didefinisikan. Terdapat 2 cara

untuk mengendalikan aliran eksekusi program, yaitu dengan menggunakan IF dan CASE.

➤ **Kalimat IF**

Kalimat IF mengijinkan PL/SQL untuk menjalankan perintah secara selektif berdasarkan kondisi yang didefinisikan. Susunan kode untuk alir kondisional menggunakan IF adalah sebagai berikut:

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

Pada susunan kode ini:

<i>Syntax</i>	Penjelasan
<i>condition</i>	Variabel BOOLEAN atau ekspresi yang mengembalikan nilai TRUE, FALSE, atau NULL.
THEN	Mengarahkan ke klausa yang menghubungkan ekspresi BOOLEAN dengan urutan kalimat PL/SQL yang mengikutinya.
<i>statements</i>	Bisa terdiri atas 1 atau lebih kalimat PL/SQL atau <i>query</i> SQL. Bisa juga berisi aliran kondisional lain. Proses yang terdapat pada klausa THEN hanya dijalankan jika kondisi pada klausa IF

<i>Syntax</i>	Penjelasan
	mengembalikan nilai TRUE.
ELSIF	<i>Keyword</i> opsional dan bisa dituliskan lebih dari 1, yang mengarahkan pada sebuah ekspresi BOOLEAN. (jika kondisi awal (IF awal) mengembalikan nilai FALSE atau NULL, maka proses akan diarahkan ke ekspresi BOOLEAN tambahan pada ELSIF)
ELSE	Mengarahkan ke klausa <i>default</i> yang akan dijalankan jika dan hanya jika kondisi-kondisi awal (pada IF atau ELSIF) tidak mengembalikan nilai TRUE
END IF	Menandakan akhir dari sebuah susunan kode IF dan diakhiri dengan “;”

➤ **Kalimat CASE**

Sebuah ekspresi CASE memilih sebuah hasil dan mengembalikannya berdasarkan 1 atau lebih alternatif (pilihan). Untuk mengembalikan sebuah hasil, ekspresi CASE menggunakan sebuah *selector*, sebuah ekspresi yang nilainya digunakan untuk mengembalikan 1 dari beberapa alternatif. *Selector* akan diikuti oleh 1 atau lebih klausa WHEN, yang akan diperiksa secara berurutan. Nilai pada *selector* akan menentukan hasil mana yang akan dikembalikan. Jika nilai pada *selector* sama dengan

nilai pada ekspresi klausa `WHEN`, maka klausa `WHEN` tersebut akan dijalankan dan hasilnya akan dikembalikan. Susunan kode untuk alir kondisional menggunakan `CASE` adalah sebagai berikut:

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

Ekspresi `CASE` juga dapat digunakan tanpa menggunakan sebuah *selector*, yang digunakan jika klausa `WHEN` berisi kondisi pencarian yang menghasilkan sebuah nilai `BOOLEAN`, bukan nilai dari tipe yang lain. Susunan kode untuk menuliskan `CASE` tanpa *selector* adalah sebagai berikut:

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

Aturan penanganan nilai `NULL` dalam alir kondisional

- ✓ Perbandingan sederhana yang melibatkan `NULL` akan menghasilkan `NULL`

- ✓ Menggunakan operator logika NOT pada NULL akan menghasilkan NULL
- ✓ Pada kontrol kondisional, jika sebuah kondisi menghasilkan NULL, maka urutan kalimat perintah yang mengikuti kondisi tersebut tidak akan dijalankan.

Contoh penggunaan alir kondisional (IF dan CASE) adalah sebagai berikut:

- a. Pengecekan umur seseorang, apakah terhitung dewasa atau tidak.

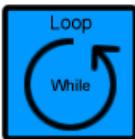
```
DECLARE
    umur number(3);
BEGIN
    umur := 18;
    IF umur < 18 THEN
        DBMS_OUTPUT.PUT_LINE('Umur Anak-anak');
    ELSIF umur >= 18 THEN
        DBMS_OUTPUT.PUT_LINE('Umur Dewasa');
    END IF;
END;
```

- b. Mengecek apakah karyawan dengan id = 100 sudah masuk masa bakti 16, 20, atau 24 tahun

```
DECLARE
  masaKerja NUMBER(3);
  empID employees.employee_id%TYPE;
BEGIN
  SELECT ROUND((sysdate-hire_date)/365)
  INTO masaKerja FROM employees
  WHERE employee_id = 100;
  CASE masaKerja
    WHEN 16 THEN
      DBMS_OUTPUT.PUT_LINE('Masa Bakti 16 tahun');
    WHEN 20 THEN
      DBMS_OUTPUT.PUT_LINE('Masa Bakti 20 tahun');
    WHEN 24 THEN
      DBMS_OUTPUT.PUT_LINE('Masa Bakti 24 tahun');
    ELSE DBMS_OUTPUT.PUT_LINE('Belum Masa Bakti');
  END CASE;
EXCEPTION WHEN no_data_found THEN
  DBMS_OUTPUT.PUT_LINE('Employee not found.');
```

```
END;
```

## Iteration Flow



Iterasi (perulangan) adalah alir program yang mengulangi sebuah/urutan proses berulang-ulang sampai kondisi berhenti tercapai.

Adalah sebuah keharusan untuk mendefinisikan kondisi berhenti dari sebuah perulangan, jika tidak, maka perulangan tidak akan pernah berhenti (*infinite*). Terdapat 3 tipe LOOP:

- *Basic* LOOP (*Infinite* LOOP)

Bentuk perulangan yang paling sederhana, yang melampirkan sebuah/urutan proses diantara *keyword* LOOP dan END LOOP. Sebuah basic LOOP menjalankan proses di dalamnya minimal 1 kali meskipun kondisi EXIT sudah dipenuhi pada saat memasuki LOOP. Tanpa kondisi EXIT, LOOP tidak akan berhenti (*Infinite*). Susunan kode untuk menuliskan Basic LOOP adalah sebagai berikut:

```
LOOP
  Statement1;
  .
  .
  .
  EXIT [WHEN condition];
END LOOP;
```

*Keyword* EXIT dapat digunakan untuk menghentikan LOOP. Setelah *keyword* END LOOP, proses akan dilanjutkan ke perintah, setelah proses LOOP. *Keyword* EXIT dapat dituliskan di dalam sebuah alir kondisional atau dapat berdiri sendiri perintah dalam LOOP. *Keyword* EXIT harus diletakkan di dalam LOOP. Klausula WHEN juga dapat ditambahkan setelah *keyword* EXIT untuk menambahkan kondisi penghentian LOOP. Sebuah *Basic* LOOP bisa memiliki banyak perintah/titik EXIT, tetapi

disarankan untuk menggunakan 1 perintah/titik EXIT saja.

➤ WHILE LOOP

WHILE LOOP digunakan untuk melakukan perulangan pada sebuah/urutan proses sampai kondisi yang mengendalikan tidak bernilai TRUE. susunan kode untuk menuliskan WHILE LOOP adalah sebagai berikut:

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

Dalam susunan kode ini:

<i>Syntax</i>	Penjelasan
<i>condition</i>	Sebuah ekspresi PL/SQL (bernilai TRUE, FALSE, atau NULL) atau variabel bertipe BOOLEAN
<i>statement</i>	Bisa terdiri atas 1 atau lebih proses PL/SQL atau <i>query</i> SQL.

Catatan pada WHILE LOOP:

- ✓ Jika variabel yang digunakan sebagai kondisi (dituliskan pada bagian *condition*) tidak dirubah pada proses perulangan (di dalam *statement*), maka

condition akan tetap bernilai TRUE dan perulangan tidak akan berhenti.

- ✓ Jika condition menghasilkan nilai NULL, maka perulangan tidak akan dijalankan dan akan diarahkan ke proses setelah perulangan.

➤ FOR LOOP

FOR LOOP digunakan untuk menyederhanakan uji coba untuk sejumlah iterasi. Susunan kode untuk menuliskan

FOR LOOP adalah sebagai berikut:

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

Pada susunan kode ini:

Syntax	Penjelasan
<i>counter</i>	Adalah sebuah variabel dengan tipe INTEGER yang harus dideklarasikan secara implisit, yang nilainya akan bertambah (atau berkurang jika <i>keyword</i> REVERSE digunakan) 1 pada setiap perulangan sampai batas atas atau batas bawah tercapai. Variabel yang dideklarasikan sebagai <i>counter</i> tidak boleh dideklarasikan di luar susunan kode FOR LOOP.

<i>Syntax</i>	Penjelasan
REVERSE	<i>Keyword</i> yang menyebabkan counter untuk berkurang pada setiap perulangan dari batas atas hingga batas bawah.
<i>Lower_bound</i>	Menentukan batas bawah dari jarak nilai counter
<i>Upper_bound</i>	Menentukan batas atas dari jarak nilai counter

Panduan dalam menggunakan LOOP:

- ✓ Gunakan basic LOOP jika proses di dalam perulangan harus dijalankan minimal 1 kali.
- ✓ Gunakan WHILE LOOP jika terdapat sebuah kondisi yang harus diperiksa pada setiap iterasi (perulangan).
- ✓ Gunakan FOR LOOP jika jumlah iterasi (perulangan) sudah diketahui.

Contoh penggunaan perulangan adalah sebagai berikut:

- a. Mencetak deret angka 1 sampai 10 dan tampilkan status genap dan ganjilnya.

```
BEGIN
  FOR i IN 1..10 LOOP
    IF MOD(i,2) = 0 THEN
      DBMS_OUTPUT.PUT_LINE(i|| ' Genap');
    ELSIF MOD(i,2) = 1 THEN
      DBMS_OUTPUT.PUT_LINE(i|| ' Ganjil');
    END IF;
  END LOOP;
END;
```

- b. Tampilkan nama lengkap, gaji, dan tipe gaji berdasarkan id karyawan yang dimasukkan.

```
DECLARE
  fname VARCHAR2(50);
  sal employees.salary%TYPE;
  salType CHAR(1);
  empID employees.employee_id%TYPE := 100;
BEGIN
  SELECT first_name||' '||last_name, salary
  INTO fname, sal
  FROM employees
  WHERE employee_id = empID;
  CASE
    WHEN sal BETWEEN 2000 AND 10000 THEN
      salType := 'C';
    WHEN sal BETWEEN 10100 AND 20000 THEN
      salType := 'B';
    WHEN sal BETWEEN 20100 AND 30000 THEN
      salType := 'A';
    ELSE salType := 'Unknown Type';
  END CASE;
  DBMS_OUTPUT.PUT_LINE('ID: '||empID);
  DBMS_OUTPUT.PUT_LINE('Full Name: '||fname);
  DBMS_OUTPUT.PUT_LINE('Salary: '||sal);
  DBMS_OUTPUT.PUT_LINE('Sal. Type: '||salType);
END;
```

## Nested PL/SQL Blocks

Sebuah blok PL/SQL dapat disarangkan pada bagian *executable* dari blok PL/SQL lain. Bagian blok PL/SQL yang dapat dimasuki oleh blok PL/SQL lain adalah bagian *Executable* (BEGIN . . . END) dan bagian *Exception*. Setiap variabel yang dideklarasikan akan mempunyai *scope* dan *visibility*.

- *Scope* dari sebuah variabel adalah bagian dari program dimana variable dideklarasikan dan dapat diakses
- *Visibility* dari sebuah variabel adalah bagian dari program dimana variabel bisa diakses tanpa menggunakan *qualifier*.

Qualifier adalah sebuah label yang diberikan pada sebuah blok PL/SQL. *Qualifier* dapat digunakan untuk mengakses variabel yang memiliki *scope* tetapi tidak memiliki *visible*.

Penulisan qualifier adalah sebagai berikut:

```
<<qualifier>>  
DECLARE  
...
```

Contoh penerapan Nested Block adalah sebagai berikut:

Mengecek apakah seorang karyawan bekerja pada departemen “Shipping” atau departemen “Marketing”, jika

benar, maka gaji ditampilkan dan dinaikkan 20%, jika tidak, maka gaji dinaikkan 10%.

```
<<empData>>
DECLARE
  empid employees.employee_id%TYPE :=&empID;
  fname VARCHAR2(50);
  sal employees.salary%TYPE;
  dname departments.department_name%TYPE;
BEGIN
  SELECT e.first_name||' '||e.last_name, e.salary,
  d.department_name
  INTO fname, sal, dname
  FROM employees e, departments d
  WHERE e.department_id = d.department_id AND
  e.employee_id = empid;
  <<salCalc>>
  DECLARE
    nsal employees.salary%TYPE;
  BEGIN
    IF empData.dname = 'Shipping' OR empData.dname
    = 'Marketing' THEN
      nsal := sal + (sal*0.2);
    ELSE
      nsal := sal + (sal*0.1);
    END IF;
    DBMS_OUTPUT.PUT_LINE('ID: '|| empData.empid);
    DBMS_OUTPUT.PUT_LINE('Name: '|| empData.fname);
    DBMS_OUTPUT.PUT_LINE('Dept Name: '||
empData.dname);
    DBMS_OUTPUT.PUT_LINE('Salary: '|| empData.sal);
    DBMS_OUTPUT.PUT_LINE('New Salary: '|| nsal);
  END;
END;
```

Catatan:

&empID adalah *substitute variable* yang akan meminta *user* untuk mengisi nilai untuk variabel tersebut, pada saat *script* dijalankan.

## Practice

Tampilkan Output Seperti Ini:

```
Employee ID : [employee_ID]
Full Name   : [first_name + last_name]
Salary      : [salary]
Level       : [*****]
```

Catatan:

- ✓ Tampilkan untuk karyawan dengan ID 111 sampai 130.
- ✓ Level diisi berdasarkan kelipatan per 1000 dari `salary`.  
(ex: jika `salary` 6000, maka jumlah bintang adalah 6, jika `salary` 15600, maka jumlah bintang adalah 16)

## Stage 2: Cursor-ing a Lot of Data

### たくさんのデータをカソリング

“Practice is about try and try more, not just reading and imagining things”

人

What you'll Learn:

About Cursor

Implicit Cursor

Explicit Cursor

Cursor and Record

Cursor With Parameter

## About *Cursors*

Sebuah *cursor* adalah sebuah petunjuk kepada private memory area yang dialokasikan oleh Oracle *server*. Setiap kalimat SQL yang dieksekusi oleh Oracle *server* memiliki sebuah *cursor* individu yang terkait dengan kalimat SQL tersebut. Terdapat 2 macam *cursor*, *cursor* implisit dan *cursor* eksplisit.

## *Implicit Cursor*

*Cursor* yang dideklarasikan dan dikelola oleh PL/SQL untuk semua *query* DML dan *query* `SELECT` pada PL/SQL. *Cursor* implisit memiliki atribut *cursor sql* yang dapat digunakan untuk memeriksa apa yang terjadi pada saat sebuah *cursor* implisit terakhir digunakan. Atribut-atribut ini hanya dapat digunakan dalam kalimat PL/SQL. Terdapat 3 atribut yang dapat digunakan untuk menguji hasil dari kalimat SQL:

➤ `SQL%FOUND`

Adalah sebuah atribut *cursor sql* bertipe `BOOLEAN` yang menghasilkan nilai `TRUE` jika kalimat SQL terbaru (paling terakhir dijalankan) mengembalikan minimal 1 baris data.

➤ `SQL%NOTFOUND`

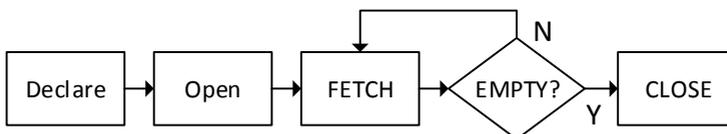
Adalah sebuah atribut *cursor sql* bertipe `BOOLEAN` yang menghasilkan nilai `TRUE` jika kalimat SQL terbaru (paling terakhir dijalankan) sama sekali tidak mengembalikan baris data.

➤ `SQL%ROWCOUNT`

Adalah sebuah atribut *cursor sql* bertipe `INTEGER` yang mengembalikan jumlah baris data yang terpengaruh dari kalimat SQL terbaru (paling terakhir dijalankan).

### *Explicit Cursor*

*Cursor* yang dideklarasikan dan dikelola oleh si programmer dan dimanipulasi melalui bagian `executable` pada blok PL/SQL, dan digunakan pada query SQL yang mengembalikan nilai lebih dari 1 baris data. Untuk mengendalikan/menggunakan sebuah *cursor* eksplisit, terdapat urutan proses yang harus dilakukan.



Gambar 2-1 Urutan Proses Pengendalian/Penggunaan *Cursor* Eksplisit

➤ DECLARE

*Cursor* harus dideklarasikan lebih dulu pada bagian *declarative* dari sebuah blok PL/SQL, dengan cara memberikan *identifier*/nama *cursor* dan menentukan query SQL yang terkait dengan *cursor* tersebut. Susunan kode untuk mendeklarasikan *cursor* adalah sebagai berikut:

```
CURSOR cursor_name IS  
select_statement;
```

Pada susunan kode ini:

Syntax	Penjelasan
<i>cursor_name</i>	Adalah <i>identifier</i> PL/SQL
<i>select_statement</i>	Adalah <i>query</i> SELECT tanpa klausa INTO

➤ OPEN

Kalimat OPEN mengeksekusi query yang terkait dengan *cursor*, mengidentifikasi *active set* (baris data yang akan dikembalikan), dan memposisikan pointer *cursor* ke baris pertama dari baris data yang akan dikembalikan. Kalimat OPEN dimasukkan pada bagian *executable* dari sebuah

blok PL/SQL. Susunan kode untuk kalimat `OPEN` adalah sebagai berikut:

```
OPEN cursor_name;
```

➤ `FETCH`

Kalimat `FETCH` mengembalikan baris data dari *cursor* per barisnya. Setelah tiap pengambilan, pointer dari *cursor* akan berpindah ke baris berikutnya dari *active set*. Kalimat `FETCH` dimasukkan pada bagian *executable* dari sebuah blok PL/SQL. Susunan kode untuk kalimat `FETCH` adalah sebagai berikut:

```
FETCH cursor_name  
INTO variable1, ..., variable || record_name;
```

Catatan:

- ✓ Jika menggunakan variabel pada klausa `INTO`, jumlah variabel yang dituliskan pada klausa `INTO` harus sama dengan jumlah kolom yang dihasilkan oleh perintah `SELECT` pada *cursor*.
- ✓ Jika menggunakan variabel pada klausa `INTO`, urutan kolom dan urutan penulisan variabel pada klausa `INTO` harus sesuai.

Untuk mengambil semua baris data yang ada di *cursor*, dapat menggunakan perulangan.

➤ CLOSE

Kalimat `CLOSE` digunakan untuk menutup *cursor* yang sudah selesai digunakan (setelah menyelesaikan pemrosesan dari kalimat `FETCH`). Setelah ditutup, *cursor* dapat dibuka kembali dengan menggunakan kalimat `OPEN`. Kalimat `CLOSE` dimasukkan pada bagian *executable* dari sebuah blok PL/SQL. Susunan kode untuk kalimat `CLOSE` adalah sebagai berikut:

```
CLOSE cursor_name;
```

*Cursor* Eksplisit juga memiliki atribut *cursor*, yaitu:

➤ %ISOPEN

Adalah atribut *cursor* bertipe `BOOLEAN` yang akan bernilai `TRUE` jika *cursor* sedang dalam kondisi `OPEN`.

➤ %NOTFOUND

Adalah atribut *cursor* bertipe `BOOLEAN` yang akan bernilai `TRUE` jika `FETCH` terbaru tidak mengembalikan baris data.

➤ %FOUND

Adalah atribut *cursor* bertipe `BOOLEAN` yang akan bernilai `TRUE` jika `FETCH` terbaru mengembalikan baris data.

➤ %ROWCOUNT

Adalah atribut *cursor* bertipe NUMBER yang akan mengembalikan jumlah total baris data yang sudah dikembalikan oleh *cursor*.

Contoh Penggunaan *Cursor* dapat dilihat pada contoh berikut: tampilkan id departemen, nama departemen, rata2 gaji dan jumlah karyawan pada departemen tersebut.

```
DECLARE
  CURSOR cdept IS --DECLARE
  SELECT d.department_id, d.department_name,
         AVG(e.salary), COUNT(e.employee_id)
  FROM employees e, departments d
  WHERE d.department_id = e.department_id
  GROUP BY d.department_id, d.department_name;
  d_id departments.department_id%TYPE;
  d_name departments.department_name%TYPE;
  avg_sal employees.salary%TYPE;
  n_emp number(3);
BEGIN
  OPEN cdept; --OPEN
  FETCH cdept INTO d_id, d_name, avg_sal, n_emp;
  -- FETCH
  DBMS_OUTPUT.PUT_LINE('ID Department: ' ||
  d_id);
  DBMS_OUTPUT.PUT_LINE('Nama Department: ' ||
  d_name);
  DBMS_OUTPUT.PUT_LINE('Rata-rata Gaji: ' ||
  avg_sal);
  DBMS_OUTPUT.PUT_LINE('Jumlah Karyawan: ' ||
  n_emp);
  CLOSE cdept; --CLOSE
END;
```

Blok PL/SQL di atas hanya akan mengembalikan 1 data saja, sedangkan jika anda menginginkan untuk menampilkan semua data departemen yang ada, maka dapat ditambahkan sebuah fungsi LOOP pada saat melakukan FETCH. LOOP diatur agar berhenti pada saat tidak ada baris data yang dikembalikan oleh *cursor*.

```
DECLARE
  CURSOR cdept IS --DECLARE
  SELECT d.department_id, d.department_name,
         AVG(e.salary), COUNT(e.employee_id)
  FROM employees e, departments d
  WHERE d.department_id = e.department_id
  GROUP BY d.department_id, d.department_name;
  d_id departments.department_id%TYPE;
  d_name departments.department_name%TYPE;
  avg_sal employees.salary%TYPE;
  n_emp number(3);
BEGIN
  OPEN cdept; --OPEN
  LOOP
  FETCH cdept INTO d_id, d_name, avg_sal, n_emp; --FETCH
  EXIT WHEN cdept%NOTFOUND; --STOP CONDITION
  DBMS_OUTPUT.PUT_LINE('ID Department: ' || d_id);
  DBMS_OUTPUT.PUT_LINE('Nama Department: ' || d_name);
  DBMS_OUTPUT.PUT_LINE('Rata-rata Gaji: ' || avg_sal);
  DBMS_OUTPUT.PUT_LINE('Jumlah Karyawan: ' || n_emp);
  DBMS_OUTPUT.PUT_LINE(CHR(9));--empty space
  END LOOP;
  CLOSE cdept; --CLOSE
END
```

## Cursor And Record

### Record

*Record* pada PL/SQL adalah adalah sekelompok *data item* yang berhubungan, yang tersimpan dalam kolom-kolom, masing-masing dengan nama dan tipe datanya sendiri. Keterangan mengenai *record* pada PL/SQL:

- ✓ *Record* dapat didefinisikan dengan banyak kolom sesuai dengan kebutuhan.
- ✓ *Record* dapat diberikan nilai awal dan dapat didefinisikan/dideklarasikan sebagai `NOT NULL`.
- ✓ Kolom tanpa nilai awal akan diinisialisasi sebagai `NULL`.
- ✓ Pada saat mendefinisikan kolom, kata kunci `DEFAULT` dapat digunakan.
- ✓ *Record* dan *Record Type* dapat dideklarasikan pada bagian *declarative* pada blok PL/SQL, *subprogram*, dan *package*.
- ✓ *Record* dideklarasikan secara bersarang dan saling mereferensi. Sebuah *record* dapat menjadi bagian dari *record* lain.

Susunan kode untuk mendeklarasikan *record* adalah sebagai berikut:

```

TYPE type_name IS RECORD
  (field_declaration*[, field_declaration]...);

identifier type_name;

*field_declaration
Field_name
{field_type|variable%TYPE|table.column%TYPE
|table%ROWTYPE} [[NOT NULL] {:=|DEFAULT} expr]

```

Pada susunan kode ini:

Syntax	Penjelasan
<i>type_name</i>	Adalah nama dari <i>record type</i> (akan digunakan untuk mendeklarasikan <i>record</i> )
<i>field_name</i>	Adalah nama dari kolom di dalam <i>record</i> .
<i>field_type</i>	Adalah tipe data dari kolom pada <i>record</i> (bisa menggunakan semua tipe data PL/SQL kecuali REF CURSOR. Bisa juga menggunakan atribut %TYPE dan %ROWTYPE.
<i>expr</i>	Adalah <i>field_type</i> dari kolom, atau sebuah nilai awal.

Record dideklarasikan dengan urutan, mendeklarasikan dulu *type* dari *record* beserta kolom dari *record* tersebut, lalu dilanjutkan dengan mereferensi tipe tersebut dengan sebuah *identifier*. Contoh deklarasi record adalah sebagai berikut:

```
...
TYPE emp_record_type IS RECORD
  (last_name VARCHAR2(25),
   job_id     VARCHAR2(10),
   salary     NUMBER(8,2));
emp_record   emp_record_type;
...
```

Untuk mereferensi atau menginisialisasi sebuah *field*/kolom dari sebuah *record*, gunakan cara berikut.

```
record_name.field_name
```

Selain dengan mendefinisikan kolom-kolom pada record secara *manual*. Record juga dapat dideklarasikan melalui atribut %ROWTYPE. Atribut %ROWTYPE digunakan untuk mendeklarasikan sebuah *record* yang dapat menampung sebuah baris dari sebuah tabel atau *view*. Dengan menggunakan atribut %ROWTYPE, kolom/*field* pada *record* akan mengambil nama dan tipe data dari kolom, dari tabel atau *view* yang direferensi atau data yang diambil oleh sebuah *cursor*/variabel *cursor*. Susunan kode untuk deklarasi record dengan atribut %ROWTYPE adalah sebagai berikut:

```
DECLARE
  identifier reference%ROWTYPE;
...
```

Pada susunan kode ini:

<i>Syntax</i>	Penjelasan
<i>identifier</i>	Adalah nama dari <i>record</i>
<i>reference</i>	Adalah nama dari tabel, <i>view</i> , <i>cursor</i> , atau variabel <i>cursor</i> yang menjadi dasar dari <i>record</i> .

Contoh deklarasi *record* dengan menggunakan atribut `%ROWTYPE` adalah sebagai berikut:

```
DECLARE
  emp_record employees%ROWTYPE;
  ...
```

Keuntungan dari penggunaan atribut `%ROWTYPE` adalah:

- ✓ Tidak perlu mengetahui jumlah dan tipe data dari kolom pada tabel yang direferensi.
- ✓ Jika tipe data dari kolom pada tabel yang direferensi berubah, maka record juga akan berubah mengikuti tabel yang direferensi.
- ✓ Berguna ketika mengambil baris data dengan menggunakan *query* `SELECT`, sehingga tidak perlu mendeklarasikan variabel untuk menampung nilai dari setiap kolom.

### Cursor and Record

Telah dijelaskan bahwa *record* dapat dibuat dengan mereferensi sebuah *cursor* atau variabel *cursor* dengan menggunakan atribut `%ROWTYPE`. dengan cara ini, variabel *scalar* tidak perlu dideklarasikan per kolom, dan tidak memperhatikan urutan kolom pada *query* `SELECT` pada *cursor* dengan urutan penulisan variabel pada bagian `FETCH`, karena variabel *scalar* langsung digantikan oleh *record*.

Contoh penggunaan *cursor* yang diikuti dengan penggunaan *record* adalah sebagai berikut:

```
DECLARE
  CURSOR cdept IS --DECLARE
  SELECT d.department_id, d.department_name,
         AVG(e.salary) as avg_sal,
  COUNT(e.employee_id) as n_emp
  FROM employees e, departments d
  WHERE d.department_id = e.department_id
  GROUP BY d.department_id, d.department_name;
  dept_rec cdept%ROWTYPE; --RECORD DECLARATION
BEGIN
  OPEN cdept; --OPEN
  LOOP
  FETCH cdept INTO dept_rec; --FETCH
  EXIT WHEN cdept%NOTFOUND; --STOP CONDITION
  DBMS_OUTPUT.PUT_LINE('ID Department: ' ||
  dept_rec.department_id);
  DBMS_OUTPUT.PUT_LINE('Nama Department: ' ||
  dept_rec.department_name);
  DBMS_OUTPUT.PUT_LINE('Rata-rata Gaji: ' ||
  dept_rec.avg_sal);
  DBMS_OUTPUT.PUT_LINE('Jumlah Karyawan: ' ||
  dept_rec.n_emp);
  DBMS_OUTPUT.PUT_LINE(CHR(9));--empty space
  END LOOP;
  CLOSE cdept; --CLOSE
END;
```

Selain dengan menggunakan *basic* LOOP, record dapat digunakan untuk menampung data dari *cursor* dengan menggunakan FOR LOOP. Berikut adalah contohnya:

```
DECLARE
  CURSOR cdept IS --DECLARE
    SELECT d.department_id, d.department_name,
           AVG(e.salary) as avg_sal,
    COUNT(e.employee_id) as n_emp
    FROM employees e, departments d
    WHERE d.department_id = e.department_id
    GROUP BY d.department_id, d.department_name;
BEGIN
  FOR dept_rec IN cdept LOOP --record is
    declared implicitly
    DBMS_OUTPUT.PUT_LINE('ID Department: ' ||
    dept_rec.department_id);
    DBMS_OUTPUT.PUT_LINE('Nama Department: ' ||
    dept_rec.department_name);
    DBMS_OUTPUT.PUT_LINE('Rata-rata Gaji: ' ||
    dept_rec.avg_sal);
    DBMS_OUTPUT.PUT_LINE('Jumlah Karyawan: ' ||
    dept_rec.n_emp);
    DBMS_OUTPUT.PUT_LINE(CHR(9));--empty space
  END LOOP;
END;
```

Sebagai catatan:

- ✓ *Record* dideklarasikan dan **HARUS** dideklarasikan secara implisit.
- ✓ *Cursor* dapat juga tidak dideklarasikan, tetapi digantikan dengan *subquery*.

Contoh For LOOP dengan record dan *subquery* adalah sebagai berikut:

```
BEGIN
  FOR dept_rec IN (SELECT d.department_id,
    d.department_name, AVG(e.salary) as avg_sal,
    COUNT(e.employee_id) as n_emp
  FROM employees e, departments d
  WHERE d.department_id = e.department_id
  GROUP BY d.department_id, d.department_name) LOOP
  --Subquery is used
  DBMS_OUTPUT.PUT_LINE('ID Department: ' ||
    dept_rec.department_id);
  DBMS_OUTPUT.PUT_LINE('Nama Department: ' ||
    dept_rec.department_name);
  DBMS_OUTPUT.PUT_LINE('Rata-rata Gaji: ' ||
    dept_rec.avg_sal);
  DBMS_OUTPUT.PUT_LINE('Jumlah Karyawan: ' ||
    dept_rec.n_emp);
  DBMS_OUTPUT.PUT_LINE(CHR(9));--empty space
  END LOOP;
END;
```

## Cursor With Parameter

Sebuah *cursor* dapat dideklarasikan dengan menambahkan parameter, sehingga *cursor* dapat dibuka dan ditutup beberapa kali dalam sebuah *block*, tetapi mengembalikan kumpulan baris data yang berbeda. Untuk setiap eksekusi, *cursor* yang sebelumnya ditutup lalu kemudian dibuka kembali dengan parameter yang baru. Contoh penggunaan parameter pada *cursor* adalah sebagai berikut:

```
DECLARE
  CURSOR cdept (min_num_emp NUMBER) IS --DECLARE
  SELECT d.department_id, d.department_name,
         AVG(e.salary) as avg_sal,
  COUNT(e.employee_id) as n_emp
  FROM employees e, departments d
  WHERE d.department_id = e.department_id
  GROUP BY d.department_id, d.department_name
  HAVING COUNT(e.employee_id) >= min_num_emp;
  dept_rec cdept%ROWTYPE; --RECORD DECLARATION
BEGIN
  OPEN cdept(10); --OPEN
  LOOP
  FETCH cdept INTO dept_rec; --FETCH
  EXIT WHEN cdept%NOTFOUND; --STOP CONDITION
  DBMS_OUTPUT.PUT_LINE('ID Department: ' ||
  dept_rec.department_id);
  DBMS_OUTPUT.PUT_LINE('Nama Department: ' ||
  dept_rec.department_name);
  DBMS_OUTPUT.PUT_LINE('Rata-rata Gaji: ' ||
  dept_rec.avg_sal);
  DBMS_OUTPUT.PUT_LINE('Jumlah Karyawan: ' ||
  dept_rec.n_emp);
  DBMS_OUTPUT.PUT_LINE(CHR(9));--empty space
  END LOOP;
  CLOSE cdept; --CLOSE
END;
```

Pada contoh di atas, dicari data departemen yang jumlah karyawannya di atas atau sama dengan nilai *parameter* `min_num_emp`. Lalu pada saat melakukan `OPEN cursor`, *parameter* `min_num_emp` diisi dengan nilai 10, sehingga data departemen yang ditampilkan adalah daftar departemen yang memiliki jumlah pegawai diatas atau sama

dengan 10. Berikut adalah contoh dengan menggunakan FOR LOOP:

```
DECLARE
  CURSOR cdept (min_num_emp NUMBER, min_avg_sal
NUMBER) IS --DECLARE
  SELECT d.department_id, d.department_name,
         AVG(e.salary) as avg_sal,
  COUNT(e.employee_id) as n_emp
  FROM employees e, departments d
  WHERE d.department_id = e.department_id
  GROUP BY d.department_id, d.department_name
  HAVING COUNT(e.employee_id) >= min_num_emp AND
  avg(e.salary) >= min_avg_sal;
BEGIN
  FOR dept_rec IN cdept (10,5000) LOOP
    EXIT WHEN cdept%NOTFOUND; --STOP CONDITION
    DBMS_OUTPUT.PUT_LINE('ID Department: ' ||
dept_rec.department_id);
    DBMS_OUTPUT.PUT_LINE('Nama Department: ' ||
dept_rec.department_name);
    DBMS_OUTPUT.PUT_LINE('Rata-rata Gaji: ' ||
dept_rec.avg_sal);
    DBMS_OUTPUT.PUT_LINE('Jumlah Karyawan: ' ||
dept_rec.n_emp);
    DBMS_OUTPUT.PUT_LINE(CHR(9));--empty space
  END LOOP;
END;
```

Pada contoh di atas, terdapat 2 parameter yang digunakan, yang dimasukkan ke dalam *cursor* sebagai penentu batas minimal jumlah karyawan (*min\_num\_emp*) dan batas minimal rata-rata gaji (*min\_avg\_sal*) yang akan menyaring data departemen yang akan ditampilkan oleh *cursor*.

## Practice

Buatlah sebuah blok PL/SQL yang menerapkan cursor berparameter untuk menampilkan status karyawan dari sebuah departemen dengan ketentuan sebagai berikut:

- ✓ Jika `manager_id` dari karyawan adalah 101 atau 124 dan `salary` karyawan kurang dari 5000, maka status dari karyawan tersebut adalah 'Due for a raise', sedangkan jika tidak maka, status karyawan tersebut adalah 'Not due for a raise'
- ✓ Tampilan Output adalah sebagai berikut:

```
Selected Department ID: [department_id]
Department Name: [department_name]
Employee's Status
[Full Name] [Not/Due For a Raise]
.
.
.
```

## Stage 3: *Procedure*

プロシージャ

“So, how’s your study? Getting better? Or worse? What’s Next?”

人

What you’ll Learn:

Subporgram and *Procedure*

Using *Procedure*

Parametered *Procedure*

## *Subprogram and Procedure*

### *Subprogram*

Pada 2 stage yang sudah dibahas sebelumnya, blok PL/SQL yang dibahas masih berbentuk *anonymous block* (*block* tanpa nama). Karena blok PL/SQL bersifat *anonymous*, *block* tersebut tidak bisa digunakan ulang atau disimpan untuk keperluan di lain waktu.

*Procedure* (dan *function*) adalah blok PL/SQL yang memiliki nama, atau dikenal juga sebagai *subprogram*. *Subprogram* di-*compile* dan disimpan dalam database. *Subprogram* tidak hanya dapat dideklarasikan pada level *schema* saja, tetapi juga di dalam blok PL/SQL lain. Sebuah *subprogram*. Sebuah *subprogram* terdiri bagian-bagian berikut:

➤ *Bagian Declarative*

Sebuah *subprogram* dapat memiliki bagian *declarative* secara opsional, tetapi tidak dimulai dengan keyword `DECLARE` seperti pada *anonymous block*. Bagian *declarative* ditulis setelah keyword `IS` atau `AS` pada saat mendeklarasikan *subprogram*.

➤ *Bagian Executable*

*Bagian Executable* adalah bagian yang harus ada pada sebuah *subprogram*, yang berisi implementasi dari logika bisnis. Dengan melihat pada bagian ini, dapat diketahui tujuan dibuatnya *subprogram*. Bagian ini dimulai dengan keyword `BEGIN` dan diakhiri dengan keyword `END`.

➤ *Bagian Exception*

*Bagian Exception* adalah bagian opsional yang menspesifikasikan tindakan yang akan dilakukan ketika terjadi *error* dan kondisi abnormal pada bagian eksekusi, sama seperti pada *anonymous block*.

Perbedaan dari *anonymous block* dan *subprogram* adalah sebagai berikut:

<i>Anonymous block</i>	<i>Subprograms</i>
Blok PL/SQL tanpa nama	Blok PL/SQL dengan nama
di- <i>compile</i> setiap waktu	di- <i>compile</i> satu kali
Tidak tersimpan pada <i>database</i>	Tersimpan pada <i>database</i>
Tidak dapat dipanggil oleh aplikasi lain.	Memiliki nama, sehingga dapat dipanggil oleh aplikasi lain
Tidak mengembalikan nilai	<i>Subprogram</i> bisa mengembalikan nilai ( <i>Function</i> )
Tidak dapat menerima	Dapat menerima

---

<i>Anonymous block</i>	<i>Subprograms</i>
<i>parameter</i>	parameter

Terdapat beberapa keuntungan dari *subprogram*, yaitu:

- ✓ Kemudahan pemeliharaan
- ✓ Peningkatan keamanan dan keutuhan/integritas data
- ✓ Peningkatan performa
- ✓ Peningkatan kejelasan kode

### *Procedure*

*Procedure* adalah tipe dari *subprogram* yang menjalankan sebuah proses/tindakan, yang dapat disimpan dalam database sebagai obyek schema. *Procedure* memiliki bagian header, bagian declarative, bagian executable, dan bagian *exception handling* (opsional). *Procedure* mendukung reusability dan maintainability, yaitu *procedure*, setelah divalidasi, dapat digunakan pada banyak aplikasi, dan jika kebutuhan berubah, maka cukup *procedure* tersebut yang harus dirubah. Sebuah *procedure* dipanggil dengan menggunakan nama *procedure* pada bagian executable dari blok PL/SQL lain, atau dengan menggunakan perintah EXEC [nama *procedure*].

## Using Procedure

### Creating Procedure

Berikut adalah susunan kode untuk membuat *procedure*:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter1 [mode] datatype1,
  parameter2 [mode] datatype2, ...
  parameter [mode] datatypeN)] IS|AS
[local_variable_declarations; ...]
BEGIN
  --actions;
END [procedure_name];
```

Catatan dalam pembuatan *procedure*:

- ✓ Opsi REPLACE menunjukkan bahwa, jika *procedure* yang akan dibuat sudah ada (terdapat *procedure* dengan nama yang sama persis dengan nama *procedure* yang akan dibuat), maka *procedure* tersebut akan di-drop (dihapus) dan digantikan dengan *procedure* baru yang dibuat.
- ✓ [local\_variable\_declarations; ...] sama dengan bagian declarative pada *anonymous block*
- ✓ Actions sama dengan bagian *executable* pada *anonymous block*. Dimulai dengan keyword BEGIN dan diakhiri dengan keyword END atau END [nama *procedure*]
- ✓ Parameter1 menggambarkan nama dari sebuah parameter.

- ✓ [mode] menentukan bagaimana parameter akan digunakan.
- ✓ datatype1 menentukan tipe data dari parameter (dituliskan tanpa *precision/maximum length*).

Contoh pembuatan *procedure* adalah sebagai berikut:

- a. membuat *procedure* bernama “test” untuk menampilkan pesan “hello world”

```
CREATE OR REPLACE PROCEDURE test IS
BEGIN
  DBMS_OUTPUT.PUT_LINE ('hello world');
END;
```

- b. membuat *procedure* bernama “executive\_data” untuk menampilkan `employee_id`, nama lengkap karyawan, dan `hire_date` dari departemen “Executive”.

```
CREATE OR REPLACE PROCEDURE executive_data
AS
CURSOR cExcData IS SELECT e.employee_id,
e.first_name||' '||e.last_name as full_name,
e.hire_date
FROM employees e JOIN departments d ON
e.department_id = d.department_id
WHERE d.department_name = 'Executive';
BEGIN
  FOR excRec IN cExcData LOOP
    DBMS_OUTPUT.PUT_LINE(excRec.employee_id||C
HR(9)||excREC.full_name||CHR(9)||excREC.hire
_date);
  _END LOOP;
END;
```

Untuk contoh pembuatan *procedure* dengan parameter, akan dijelaskan pada bagian *parametered procedure*.

### Invoking Procedure

sebuah *procedure* yang sudah tersimpan di dalam *database* dapat dijalankan dengan dua cara, yaitu:

- ✓ Dengan menggunakan sebuah *anonymous block*
- ✓ Dengan menggunakan *procedure* atau *subprogram* PL/SQL lain.

Contoh menjalankan sebuah *procedure* pada *anonymous block* adalah sebagai berikut:

```
BEGIN
    executive_data ;
END;
```

*Anonymous block* di atas menjalankan *procedure* *executive\_data* (contoh sebelumnya). Contoh menjalankan sebuah *procedure* pada *subprogram* lain, adalah sebagai berikut:

```
CREATE OR REPLACE PROCEDURE
print_executive_data IS
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Employee
ID' || chr(9) || 'Full Name' || chr(9) || 'Hire
Date');
    executive_data;
END;
```

Pada contoh di atas, *procedure* `executive_data` dijalankan pada *procedure* lain yang bernama `print_executive_data`. Untuk cara memanggil *procedure* yang memiliki parameter, akan dijelaskan pada bagian parametered *procedure*.

### Removing Procedure

Ketika sebuah *procedure* sudah tidak dibutuhkan, maka *procedure* tersebut dapat dihapus, dengan perintah

```
DROP PROCEDURE procedure_name
```

*Procedure\_name* adalah nama dari *procedure* yang akan dihapus. Contoh:

```
DROP PROCEDURE print_executive_data
```

### Parametered Procedure

#### Parameter

Parameter digunakan untuk memindahkan nilai kepada/ dari pemanggil *subprogram* kepada/dari *subprogram*. Parameter dideklarasikan pada bagian *header* dari sebuah *subprogram*, setelah penulisan nama *subprogram* dan sebelum bagian *declarative* dari *subprogram*.

Parameter dapat digunakan seperti variabel lokal (variabel yang dideklarasikan pada bagian *declarative* di sebuah blok PL/SQL) pada sebuah *subprogram*, tetapi parameter sangat bergantung pada *mode* yang didefinisikan pada parameter tersebut. Terdapat tiga macam *mode* yang dapat didefinisikan pada sebuah parameter.

✓ IN

Parameter dengan mode IN hanya akan memindahkan nilai dari pemanggil *subprogram* ke dalam *subprogram*.

✓ OUT

Parameter dengan mode OUT hanya akan memindahkan nilai dari *subprogram* ke pemanggil *subprogram*.

✓ IN OUT

Parameter dengan mode IN OUT akan memindahkan nilai dari pemanggil *subprogram* ke dalam *subprogram*, dan memungkinkan untuk memindahkan nilai yang berbeda dari *subprogram* ke pemanggil *subprogram*, dengan menggunakan satu parameter yang sama.

### Using Parametered Procedure

a. Using IN Parameter

Contoh *procedure* dengan menggunakan parameter IN adalah sebagai berikut: buatlah sebuah *procedure* untuk menampilkan `employee_id`, `last_name`, dan `salary` berdasarkan `department_id`. `department_id` dapat dirubah sesuai kebutuhan.

```
CREATE OR REPLACE PROCEDURE dept_employee
(DeptID IN VARCHAR2) AS
  CURSOR cDeptEmp IS SELECT employee_id,
last_name, salary FROM employees WHERE
department_id = DeptID;
BEGIN
  FOR empRec IN cDeptEmp LOOP
    DBMS_OUTPUT.PUT_LINE(empRec.employee_id||C
HR(9)||empRec.last_name||CHR(9)||empRec.sala
ry);
  END LOOP;
END;
```

Pada contoh di atas, *procedure* `dept_employee` menerima nilai dengan parameter `DeptID` untuk menentukan, `department_id` berapa yang akan ditampilkan daftar karyawannya. Untuk menjalankan *procedure* di atas, dapat menggunakan *anonymous block* sebagai berikut:

```
BEGIN
  dept_employee (80);
END;
```

*Anonymous block* di atas akan memasukkan nilai 80 ke parameter DeptID pada *procedure* dept\_employee, yang membuat *procedure* menampilkan employee\_id, last\_name, dan salary dari karyawan dengan department\_id 80 saja.

Sebuah parameter IN dapat juga diberikan sebuah nilai *default*. Jika sebuah parameter IN pada *subprogram* diberikan sebuah nilai *default*, maka pada saat *subprogram* dipanggil, parameter yang memiliki nilai *default* tidak harus diberi nilai. Contohnya:

```
CREATE OR REPLACE PROCEDURE dept_employee
(DeptID IN VARCHAR2 :=100) AS
  CURSOR cDeptEmp IS SELECT employee_id,
last_name, salary FROM employees WHERE
department_id = DeptID;
BEGIN
  FOR empRec IN cDeptEmp LOOP
    DBMS_OUTPUT.PUT_LINE(empRec.employee_id||C
HR(9)||empRec.last_name||CHR(9)||empRec.sala
ry);
  END LOOP;
END;
```

Pada *procedure* di atas, parameter DeptID diberi sebuah nilai *default* 100, sehingga pada saat dipanggil, jika parameter DeptID tidak diberi nilai, parameter DeptID akan bernilai 100. Cara pemanggilannya pun dapat berubah seperti berikut:

```
BEGIN
  dept_employee;
END;
```

Pada *anonymous block* di atas, *procedure* `dept_employee` dapat dipanggil tanpa memasukkan nilai ke dalam parameter `DeptID`, karena sudah memiliki nilai *default* 100. Selain menggunakan `:=`, penulisan nilai *default* dapat juga menggunakan *keyword* `DEFAULT`, seperti contoh berikut.

```
CREATE OR REPLACE PROCEDURE dept_employee
(DeptID IN VARCHAR2 DEFAULT 100) AS
  CURSOR cDeptEmp IS SELECT employee_id,
last_name, salary FROM employees WHERE
department_id = DeptID;
BEGIN
  FOR empRec IN cDeptEmp LOOP
    DBMS_OUTPUT.PUT_LINE(empRec.employee_id||C
HR(9)||empRec.last_name||CHR(9)||empRec.sala
ry);
  END LOOP;
END;
```

Sebagai catatan, parameter yang memiliki nilai *default*, masih dapat diisi dengan nilai lain pada saat *subprogram* dari parameter tersebut dipanggil.

b. Using `OUT` Parameter

Contoh *procedure* dengan menggunakan parameter `OUT` adalah sebagai berikut: tambahkan fungsi pada

*procedure* `dept_employee` untuk menampilkan jumlah karyawan pada `department_id` yang dimasukkan.

```
CREATE OR REPLACE PROCEDURE dept_employee
(DeptID IN VARCHAR2, nmbOfEmp OUT Number) AS
  CURSOR cDeptEmp IS SELECT employee_id,
  last_name, salary FROM employees WHERE
  department_id = DeptID;
BEGIN
  nmbOfEmp := 0;
  FOR empRec IN cDeptEmp LOOP
    DBMS_OUTPUT.PUT_LINE(empRec.employee_id||CHR(9)||empRec.last_name||CHR(9)||empRec.salary);
    nmbOfEmp := nmbOfEmp + 1;
  END LOOP;
END;
```

Pada *procedure* ini, terdapat penambahan parameter `nmbOfEmp` untuk mengembalikan jumlah karyawan dari department. Nilai parameter tersebut bertambah 1 setiap kali terjadi pengambilan data oleh *record* `empRec` pada *cursor*. Untuk menjalankan *procedure* di atas, dapat menggunakan *anonymous block* sebagai berikut:

```
DECLARE
  empCount NUMBER(3);
BEGIN
  dept_employee(30, empCount);
  DBMS_OUTPUT.PUT_LINE('Number of Employee
in this Department: ' || empCount);
END;
```

*Anonymous block* di atas akan memasukkan nilai 30 ke dalam parameter DeptID dari *procedure* dan menerima nilai yang dikembalikan oleh parameter nmbOfEmp dari *procedure* pada variabel empCount.

c. Using IN OUT Parameter

Contoh *procedure* dengan parameter IN OUT adalah sebagai berikut: buatlah sebuah *procedure* untuk mem-format tampilan email yang dimasukkan, berdasarkan department\_id yang dimasukkan. Format yang ditampilkan adalah “[email yang dimasukkan] @[3 Digit depan department\_name].comp”

```
CREATE OR REPLACE PROCEDURE format_email (email
IN OUT VARCHAR2, deptID IN VARCHAR2) AS
    v_deptName departments.department_name%TYPE;
BEGIN
    SELECT department_name INTO v_deptName FROM
departments WHERE department_id = deptID;
    email :=
email||'@'||SUBSTR(v_deptname,1,3)||'.comp';
END;
```

Pada contoh di atas, *procedure* format\_email menerima nilai dengan parameter email untuk menerima email yang akan di-format, sekaligus menerima nilai department\_id dari parameter deptID untuk mengambil nama departemen yang akan

digunakan untuk mem-*format* email. Parameter `email` juga akan digunakan untuk mengembalikan email hasil *format*. Untuk menjalankan *procedure* ini, dapat menggunakan *anonymous block* sebagai berikut:

```
DECLARE
  v_deptID employees.department_id%TYPE;
  v_mail VARCHAR2(30);
BEGIN
  SELECT department_id,email INTO v_deptID,
  v_mail FROM employees WHERE employee_ID = 100;
  format_email(v_mail, v_deptID);
  DBMS_OUTPUT.PUT_LINE(v_mail);
END;
```

Pada *anonymous block* di atas, akan memasukkan email dan `department_id` dari karyawan dengan `employee_id` 100 ke dalam parameter `email` dan `deptID` melalui variabel `v_mail` dan `v_deptID`, dan kemudian akan menampilkan email hasil *format* dengan menggunakan variabel `v_mail` yang sama.

## Practice

Buatlah sebuah *procedure* dengan nama `new_job` yang berfungsi untuk memasukkan jabatan (*job*) baru ke dalam tabel `job`. Dengan ketentuan sebagai berikut:

- `Job_ID` = 2 Digit Huruf Depan `job_title` + '\_' + 3 Digit Huruf Belakang `job_title` + '\_' + (jumlah baris pada tabel `jobs` + 1)
- `Job_title` = ditentukan pada saat *procedure* dipanggil (gunakan parameter)
- `Min_salary` = ditentukan pada saat *procedure* dipanggil (gunakan parameter)
- `Max_salary` = `min_salary` x 2.

## Stage 4: Stored Function

### ストレッドファンクション

“Trouble is like a dual edge sword, whether to make you stronger or weaker, will depend to how is you response to it”

人

What you'll Learn:

About Stored Function

Using Function

Multiple Value Return

## About Stored Function

Sebuah *function* adalah sebuah *subprogram* atau blok PL/SQL yang memiliki nama, yang dapat menerima parameter, dapat dipanggil, dan **HARUS** mengembalikan sebuah nilai. Seperti *procedure*, *function* juga dapat disimpan di database sebagai obyek *schema* untuk penggunaan berulang-ulang. *Function* juga mendukung reusability dan maintainability, sama seperti *procedure*. Terdapat beberapa keuntungan yang dapat diambil dengan menerapkan *function* (yang dibuat sendiri oleh *user*):

- ✓ Dapat memperluas SQL pada saat kebutuhan terlalu kompleks, terlalu ambigu, dan tidak bisa diselesaikan dengan SQL biasa.
- ✓ Dapat meningkatkan efisiensi ketika digabungkan dengan klausa WHERE untuk menyaring data, dibandingkan dengan menyaring data lewat aplikasi.
- ✓ Dapat memanipulasi nilai data.
- ✓ Meningkatkan independensi data dengan memproses analisa data yang kompleks di dalam *server* Oracle, daripada mengambil data ke dalam aplikasi.

Meskipun terlihat sama, terdapat beberapa perbedaan antara *procedure* dan *functions*.

<i>Procedure</i>	<i>Function</i>
Dieksekusi sebagai sebuah kalimat PL/SQL	Dipanggil sebagai bagian dari sebuah ekspresi
Tidak memuat klausa RETURN pada bagian <i>header</i>	Harus memuat klausa RETURN pada <i>header</i>
Dapat mengembalikan nilai (jika diperlukan) melalui parameter keluaran	Harus mengembalikan sebuah nilai
Dapat memuat sebuah kalimat RETURN tanpa sebuah nilai	Harus memuat minimal 1 kalimat RETURN

Procedure dibuat untuk menyimpan serangkaian aktivitas atau tindakan untuk penggunaan ke depan. Sebuah procedure dapat memuat parameter yang banyak atau tidak sama sekali, tetapi procedure tidak harus mengembalikan nilai. Sebuah procedure dapat memanggil function jika diperlukan.

Function dibuat ketika dibutuhkan perhitungan sebuah nilai yang harus dikembalikan ke pemanggil subprogram (function). Sebuah function dapat memuat parameter yang banyak atau tidak sama sekali. Biasanya sebuah function hanya mengembalikan satu nilai saja yang dikembalikan melalui kalimat RETURN. *Function* yang memuat parameter bertipe OUT atau IN OUT tidak dapat digunakan oleh kalimat SQL.

## Using Stored Function

### Creating Stored Function

Berikut adalah susunan kode untuk membuat *stored function*.

```
CREATE (OR REPLACE) FUNCTION function_name
[(parameter1 [mode1] datatype1, ...)]
RETURN datatype IS|AS
[local_variable_declarations; ...]
BEGIN
--actions;
RETURN expression;
END [function_name];
```

Catatan pembuatan *stored function*:

- ✓ Keyword OR REPLACE berfungsi sama dengan pada procedure, yaitu jika terdapat *function* yang sama dengan *function* yang akan dibuat, maka *function* tersebut akan dihapus dan akan digantikan oleh *function* yang dibuat.
- ✓ Datatype pada klausa RETURN tidak boleh mencantumkan spesifikasi ukuran (*precision/maximum\_length*).
- ✓ [*local\_variable\_declarations*;...] sama dengan bagian declarative pada *anonymous block*
- ✓ Actions sama dengan bagian *executable* pada *anonymous block*. Dimulai dengan keyword BEGIN dan

diakhiri dengan *keyword* END atau END [nama *function*]

- ✓ Harus terdapat minimal 1 kalimat RETURN pada bagian *executable* dari *function*.

Contoh Pembuatan *stored function* adalah sebagai berikut:

- a. Buat sebuah *function* untuk menentukan *grade* dari gaji karyawan berdasarkan *salary* karyawan.

```
CREATE OR REPLACE FUNCTION getGrade
(sal IN employees.salary%TYPE)
RETURN CHAR IS
grade CHAR(1);
BEGIN
IF sal BETWEEN 5000 AND 10000 THEN
grade := 'C';
ELSIF sal BETWEEN 10100 AND 15000 THEN
grade := 'B';
ELSIF sal >= 15100 THEN
grade := 'A';
ELSE
grade := 'U';
END IF;
RETURN grade;
END;
```

- b. Buat sebuah *function* untuk menghasilkan nama lengkap dari karyawan berdasarkan *employee\_id*.

```
CREATE OR REPLACE FUNCTION getFullName(empID IN
employees.employee_id%TYPE)
RETURN VARCHAR2 IS
fname VARCHAR2(20);
BEGIN
SELECT first_name||' '||last_name INTO fname
FROM employees
WHERE employee_id = empID;
RETURN fname;
END;
```

### Invoking Stored Function

Stored function yang sudah disimpan dalam database dapat dipanggil dengan beberapa cara:

- a. Sebagai bagian dari sebuah ekspresi PL/SQL.

Variabel *host* dan variabel lokal (pada sebuah anonymous block) dapat digunakan untuk menangkap nilai yang dikembalikan *stored function* yang dipanggil. Contoh pemanggilan stored function menggunakan variabel *host* adalah sebagai berikut.

```
VARIABLE salgrade VARCHAR2(2);
EXECUTE :salgrade := getGrade(10000);
EXECUTE DBMS_OUTPUT.PUT_LINE(:salgrade);
```

Pada susunan kode di atas, sebuah variabel *host* (variabel tidak memiliki hubungan dengan anonymous block apapun) bernama *salgrade*, lalu variabel tersebut diisi dengan hasil dari *function* *getGrade* yang diberi nilai

10000 pada parameternya. Setelah diisi, nilai pada variabel tersebut ditampilkan. Sedangkan contoh pemanggilan *function* pada sebuah *anonymous block* menggunakan variabel lokal adalah sebagai berikut:

```
DECLARE
vid employees.employee_id%TYPE;
vsal employees.salary%TYPE;
vgrade CHAR(1);
BEGIN
vid := &empID;
SELECT salary INTO vsal FROM employees
WHERE employee_id = vid;
vgrade := getGrade(vsal);
DBMS_OUTPUT.PUT_LINE('Grade salary dari
karyawan dengan ID ' || vid ||' adalah ' ||
vgrade);
END;
```

Pada saat *anonymous block* di atas dijalankan, *user* akan diminta memasukkan *employee\_id* untuk mengisi variabel *vid*. Setelah *vid* diisi, *vid* digunakan untuk mencari *salary* yang akan dicek *grade*-nya (*salary* dimasukkan ke variabel *vsal*). Setelah *vsal* diisi, *vgrade* diisi dengan menjalankan *function* *getGrade* yang parameternya diisi berdasarkan variabel *vsal*. Setelah *grade* ditemukan, *employee\_id* dan *grade* ditampilkan.

b. Sebagai parameter dari *subprogram* lain

Nilai yang dikembalikan oleh sebuah stored function dapat digunakan sebagai nilai untuk parameter dari subprogram lain (bisa *procedure* atau *stored function*). Berikut contoh penggunaan *function* sebagai parameter pada *subprogram* lain.

```
CREATE OR REPLACE FUNCTION getSalary(empID
IN VARCHAR2) RETURN NUMBER AS
v_sal employees.salary%TYPE;
BEGIN
SELECT salary INTO v_sal FROM employees
WHERE employee_id = empID;
RETURN v_sal;
END;
```

*Function* di atas digunakan untuk mengambil gaji karyawan berdasarkan *employee\_id* karyawan. *Function* ini akan digunakan untuk menampilkan grade gaji dari karyawan pada *function* *getGrade* yang sudah dibuat sebelumnya.

```
DECLARE
vid employees.employee_id%TYPE;
BEGIN
vid := 100;
DBMS_OUTPUT.PUT_LINE('Employee ID ' || vid);
DBMS_OUTPUT.PUT_LINE('Nama Lengkap: ' ||
getFullName(vid));
DBMS_OUTPUT.PUT_LINE('Gaji/Grade: ' ||
getSalary(vid) || '/' ||
getGrade(getSalary(vid)));
END;
```

Pada *anonymous block* di atas, akan ditampilkan `employee_id`, nama lengkap, gaji, dan *grade* dari gaji milik karyawan dengan id 100. Untuk mengambil nama lengkap, digunakan function `getFullName` yang sudah dibuat sebelumnya. Untuk mengambil gaji, digunakan function `getSalary` yang sudah dibuat sebelumnya, sedangkan untuk mengambil *grade*, digunakan function `getGrade` yang sudah dibuat sebelumnya, dengan parameter yang diisi dengan hasil dari function `getSalary`.

- c. Sebagai bagian dari sebuah kalimat SQL.

Sebuah *function* dapat digunakan juga sebagai *single-row function* pada sebuah kalimat SQL. Berikut adalah contoh penggunaan *function* pada sebuah kalimat SQL.

```
SELECT employee_id, getFullName(employee_id)
"full_name", salary , getGrade(salary)
from employees where department_id = 50;
```

Pada kalimat SQL di atas, terdapat perintah `SELECT` yang memanggil function `getFullName` untuk mengambil nama lengkap, function `getGrade` untuk mengambil *grade* dari gaji karyawan pada seluruh karyawan dengan `department_id` 50.

## Removing Stored Function

Seperti pada *procedure*, Ketika sebuah *function* sudah tidak dibutuhkan, maka *stored function* tersebut dapat dihapus, dengan perintah

```
DROP FUNCTION function_name;
```

*Function\_name* adalah nama dari *function* yang akan dihapus. Contoh:

```
DROP FUNCTION getFullName;
```

## Table Function

*Table function* adalah *function* yang menghasilkan sekumpulan baris yang dapat di-*query* seperti pada tabel *database* fisik. *Table function* dapat digunakan seperti nama pada sebuah tabel *database* pada klausa `FROM` dari sebuah *query* (dengan menambahkan *keyword* `TABLE`).

Eksekusi dari *table function* dapat dilakukan secara paralel, dan baris data yang dikembalikan dapat dialirkan langsung ke proses berikutnya tanpa perlu perantara antar proses. Pada baris dari koleksi dikembalikan oleh *table function*, juga dapat dilakukan *pipelining* yaitu, secara iteratif,

mengembalikan baris data yang dihasilkan, tanpa perlu dikumpulkan terlebih dulu sebelum dikembalikan.

*Streaming* (pengaliran langsung), *pipelining*, dan eksekusi secara paralel dapat meningkatkan kinerja:

- ✓ Dengan mengaktifkan *multiple threading*, sehingga eksekusi *table function* dapat dilakukan secara paralel (bersamaan).
- ✓ Dengan menghilangkan perantara antar proses
- ✓ Dengan meningkatkan waktu respon dari *query*. Dengan *table function* yang tidak menerapkan *pipelining*, seluruh baris koleksi yang akan dikembalikan oleh *table function* harus dibangun terlebih dulu dan dikembalikan ke *server*, sebelum *query* (yang menggunakan *table function*) dapat mengembalikan sebuah baris hasil. *Pipelining* memungkinkan baris data dikembalikan secara iteratif pada saat baris data diproduksi. Hal ini dapat mengurangi penggunaan *memory* yang digunakan oleh *table function*.

### Non-Pipelined Table Function

Table Function yang tidak menerapkan *pipelining* mengembalikan hasil bertipe koleksi dan dapat di-*query* seperti sebuah tabel dengan memanggil *table function* pada

klausula FROM dari sebuah query dan ditambahkan keyword FROM.

Berikut adalah langkah-langkah (sekaligus contoh) dalam membuat *table function*.

1. membuat OBJECT TYPE dari koleksi yang menjadi dasar kolom yang akan dikembalikan oleh *table function*.

```
CREATE OR REPLACE TYPE testRow IS OBJECT  
(col1 NUMBER);
```

Pada susunan kode di atas, dibuat sebuah OBJECT TYPE koleksi dengan nama testRow yang memiliki sebuah kolom bernama col1 dengan tipe data NUMBER.

2. Setelah membuat OBJECT TYPE, dibuat juga TABLE TYPE yang digunakan untuk menampung nilai yang akan dikembalikan oleh *table function*.

```
CREATE OR REPLACE TYPE testTableType IS  
TABLE OF testRow;
```

Pada susunan kode di atas, dibuat sebuah TABLE TYPE bernama testTableType berdasarkan OBJECT TYPE testRow yang sudah dibuat sebelumnya. testTableType akan menjadi tipe data berbentuk tabel yang memiliki kolom yang sesuai dengan testRow.

3. Setelah membuat TABLE TYPE, dibuat sebuah *function* yang mengembalikan tipe data berdasarkan TABLE TYPE yang sudah dibuat.

```
CREATE OR REPLACE FUNCTION getEmployeeID
RETURN testTableType IS
CURSOR cEmp IS SELECT employee_id FROM
employees;
testTable testTableType := testTableType();
BEGIN
FOR empRec IN cEmp LOOP
testTable.EXTEND;
testTable(testTable.LAST) :=
testRow(empRec.employee_id);
END LOOP;
RETURN testTable;
END;
```

Pada susunan kode di atas, `testTableType` digunakan pada *keyword* `RETURN` di bagian *header function* untuk menyatakan bahwa `testTableType` adalah tipe yang akan dikembalikan oleh *function* `getEmployeeID`. Pada *function* ini juga dideklarasikan sebuah variabel `testTable` yang bertipe data sesuai dengan TABLE TYPE yang dituliskan pada *keyword* `RETURN`, dan diinisiasi dengan TABLE TYPE yang sama (diperlukan untuk memasukkan baris data yang diambil dari *cursor* `cEmp`). Lalu dengan memanfaatkan `RECORD` dan mekanisme perulangan, memindahkan hasil dari

*cursor* pada RECORD ke variabel `testTable`. Langkah pemindahannya adalah:

- ✓ Variabel `testTable` diberi sebuah space kosong untuk diisi nilai dari *cursor* (dengan perintah `EXTEND`).
- ✓ Nilai terakhir dari variabel `testTable` (dengan kata kunci `LAST`) diisi dengan nilai dari RECORD dengan memanfaatkan `OBJECT TYPE testRow` yang sudah dibuat untuk menyesuaikan kolom-kolom yang diisi dengan nilai dari RECORD

Setelah memindah semua nilai, variabel `testTable` dikembalikan dengan perintah `RETURN`.

### Pipelined Table Function

Pipelining pada *table function* memungkinkan untuk mengembalikan baris data dengan lebih cepat dan mengurangi *memory* yang diperlukan dalam eksekusi *table function*. Sebuah *table function* yang menggunakan pipelining, dapat mengembalikan koleksi hasil *table function* sebagai *subset* yang dapat diambil berdasarkan permintaan. Untuk langkah-langkah (sekaligus contoh) adalah sebagai berikut:

1. Dibuat dulu OBJECT TYPE untuk mendefinisikan kolom dari koleksi yang akan dikembalikan.

```
CREATE OR REPLACE TYPE deptDataRow IS OBJECT
(deptID NUMBER,
 deptName VARCHAR2(50),
 manID NUMBER,
 manName VARCHAR2(50),
 address VARCHAR2(100),
 city VARCHAR2(50),
 numEmp NUMBER);
```

Pada susunan kode di atas, seperti pada pembuatan OBJECT TYPE pada *non-pipelined table function*, dibuat sebuah OBJECT TYPE dengan nama deptDataRow dengan kolom-kolom seperti pada susunan kode di atas.

2. Setelah OBJECT TYPE dibuat, dibuat TABLE TYPE sebagai tipe data tabel yang akan digunakan sebagai tipe data yang akan dikembalikan oleh *table function*.

```
CREATE OR REPLACE TYPE deptDataTableType IS
TABLE OF deptDataRow;
```

Pada susunan kode di atas, dibuat sebuah TABLE TYPE bernama deptDataTableType berdasarkan OBJECT TYPE deptDataRow yang sudah dibuat sebelumnya. deptDataTableType akan menjadi tipe data berbentuk tabel yang memiliki kolom yang sesuai dengan deptDataRow.

3. Setelah TABLE TYPE dan OBJECT TYPE, dibuat sebuah *function* yang mengembalikan tipe data berdasarkan TABLE TYPE yang sudah dibuat.

```
CREATE OR REPLACE FUNCTION getDeptData
RETURN deptDataTableType PIPELINED IS
CURSOR cDeptData IS SELECT d.department_id,
d.department_name, d.manager_id,
e.first_name||' '||e.last_name as
full_name , l.street_address, l.city
FROM employees e JOIN departments d ON
e.employee_id = d.manager_id JOIN locations
l ON d.location_id = l.location_id;
nEmp NUMBER;
BEGIN
FOR cDeptRec IN cDeptData LOOP
SELECT COUNT(employee_id) INTO nEMP FROM
employees WHERE department_id =
cDeptRec.department_id;
PIPE ROW(deptDataRow(cDeptRec.department_id,
cDeptRec.department_name,
cDeptRec.manager_id, cDeptRec.full_name,
cDeptRec.street_address, cDeptRec.city,
nEmp));
END LOOP;
RETURN;
END;
```

Terdapat beberapa perbedaan pada *function* yang dibuat dengan *pipelining* dan pada *function* yang dibuat tanpa *pipelining*:

- Terdapat *keyword* PIPELINED pada bagian *header* dari *function* setelah tipe pada *keyword* RETURN.

- ✓ Terdapat perintah `PIPE ROW` yang digunakan untuk mengembalikan baris data yang dihasilkan *cursor*. Baris data yang dikembalikan dimasukkan dulu ke `OBJECT TYPE deptDataRow` yang sudah dibuat untuk menyesuaikan kolom-kolom yang diisi dengan nilai dari `RECORD`.
- ✓ Perintah `RETURN` pada bagian akhir *function* dituliskan tanpa ada variabel atau nilai yang dikembalikan, karena sudah dikembalikan menggunakan perintah `PIPE ROW`.

## Practice

Buat sebuah function untuk yang digunakan untuk menghitung gaji tahunan dengan rumusan:

$$(\text{salary} * 12) + (\text{commission\_pct} * \text{salary} * 12)$$

Terdapat 1 parameter pada Function ini untuk menerima `employee_id`. Berdasarkan `employee_id` yang diterima, dicari `salary` dan `commission_pct`, dan kemudian hitung gaji tahunan menggunakan rumus yang ada. Jika `employee_id` tidak memiliki `commission_pct`, maka anggap `commission_pct` adalah 0.

## Stage 5: Oracle Developer

### (Single Block Form)

オラクルディベロッパーに紹介します  
「シングルブロックフォーム」

“Almost there, keep it up”

人

What you'll learn:

About Oracle Form Developer

Create a Block

Simple Use of The Form

## About Oracle Developer

*Oracle form developer* merupakan sebuah *tool design* visual yang dirancang khusus berpasangan dengan database Oracle. *Tool* ini digunakan untuk membangun sebuah aplikasi *form* yang terdiri atas sebuah atau beberapa *form* yang saling berhubungan. Untuk pertemuan ini, akan digunakan *software form builder* dari *Oracle form developer* versi 6i.

## Creating a Block

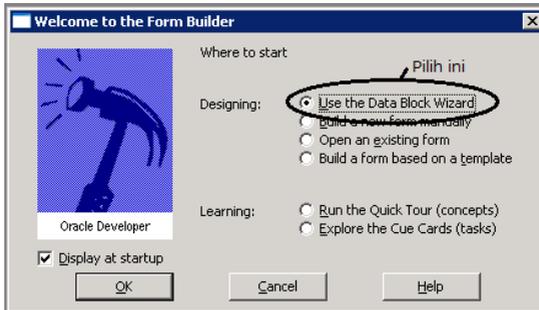
Sebuah *block data (data block)* adalah sebuah kontainer yang berisi sekelompok obyek, seperti *text items, list, buttons*, dan lain-lain. *Block data* tidak direpresentasikan secara fisik, hanya obyek-obyek yang terdapat di dalamnya yang akan terlihat dalam sebuah aplikasi/*form*.

Pada modul ini akan dijelaskan 2 cara untuk membuat *block data* beserta tampilannya, yaitu membuat secara manual atau membuat berdasarkan tabel.

## Creating Block Based On Table

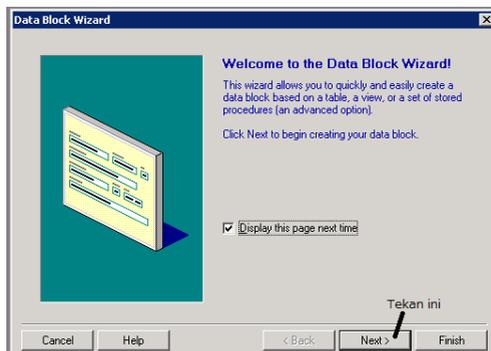
Berikut adalah penjelasan untuk membuat sebuah *block* berdasarkan sebuah tabel.

1. Pada saat *software* pertama kali dijalankan, akan muncul sebuah *dialog box* untuk menentukan apakah apa yang akan dilakukan selanjutnya. Pada poin ini, pilih [Use the Data Block Wizard], lalu tekan tombol [OK].



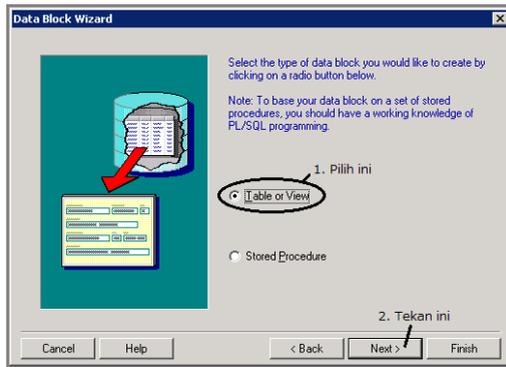
Gambar 5-1 *Dialog Box Welcome* pada *Form Builder*

2. Akan muncul *dialog box data block wizard*, pada poin ini cukup tekan tombol [Next].



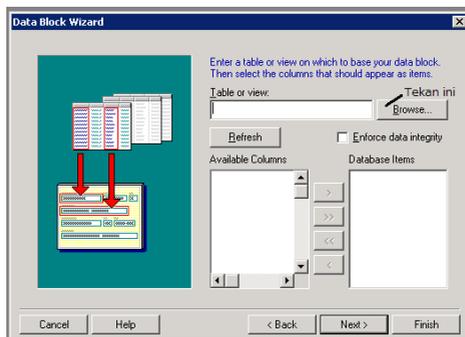
Gambar 5-2 Tampilan Awal *Dialog Box Data Block Wizard*

3. Pada poin ini, pilih [Table or View], tekan tombol [Next].



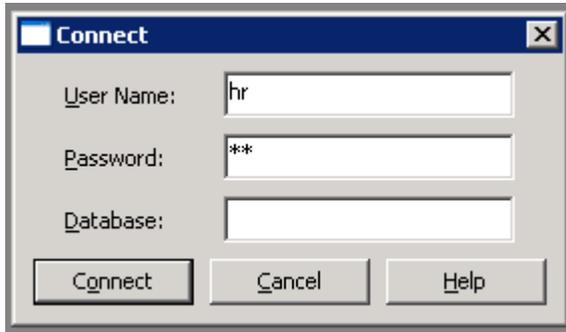
Gambar 5-3 Data Block Wizard - Memilih Tipe Data Block

4. Pada poin ini, *user* akan diminta untuk menentukan tabel apa yang akan digunakan sebagai dasar *block* data.
  - a. Tekan tombol [Browse] untuk memilih tabel apa yang akan digunakan sebagai dasar *block* data.



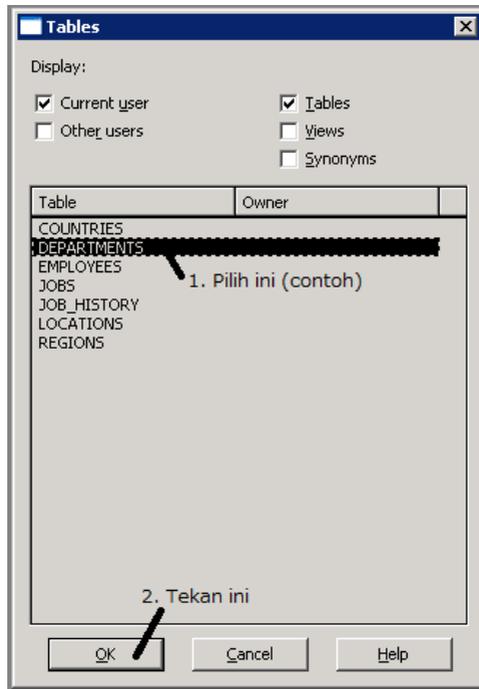
Gambar 5-4 Data Block Wizard - Memilih Tabel yang digunakan

- b. Jika *user*, belum melakukan koneksi ke Oracle database, akan muncul sebuah *dialog box* untuk melakukan koneksi ke Oracle database.



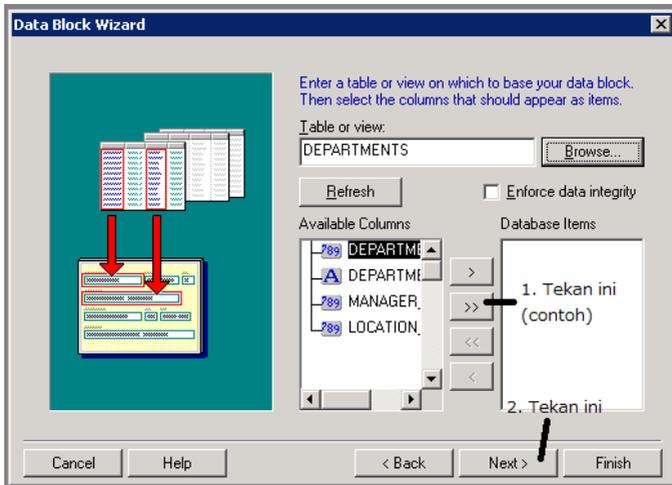
Gambar 5-5 *dialog box* untuk melakukan koneksi ke Oracle

- c. Setelah melakukan koneksi (atau jika sudah terkoneksi) ke Oracle, akan muncul *dialog box* yang menampilkan daftar tabel yang dapat diakses oleh *user* pada database tersebut. Sebagai contoh, akan digunakan tabel `departments`. Setelah memilih tabel, tekan tombol [OK].



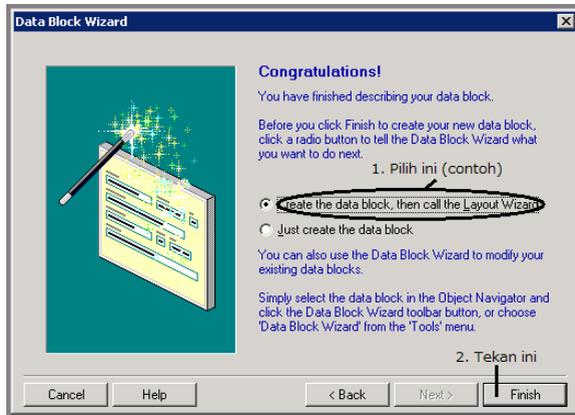
Gambar 5-6 Dialog Box Daftar Tabel untuk Data Block

- d. Setelah memilih tabel, *user* memilih kolom apa saja yang akan digunakan pada *block* data yang dibuat. Sebagai contoh, ambil semua kolom pada tabel `departments` dengan menekan tombol [`>>`], lalu tekan tombol [`Next`].



Gambar 5-7 Data Block Wizard - Memilih Tabel yang digunakan (Memilih Kolom)

5. Pada poin ini, proses pembuatan *block* data sudah selesai, dan *user* diminta untuk memilih akan melanjutkan ke *layout wizard* untuk mendesain tampilan secara otomatis, atau berhenti pada pembuatan data *block* saja (tampilan didesain secara manual). Sebagai contoh, proses akan dilanjutkan ke desain tampilan menggunakan *layout wizard* dengan memilih [Create the data block, then call the layout wizard], lalu tekan tombol [Finish].



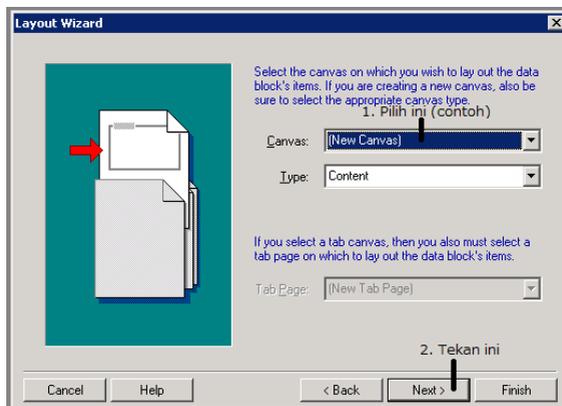
Gambar 5-8 Dialog Box Data Block Wizard - Finish

6. Akan muncul *dialog box layout wizard*, pada poin ini cukup tekan tombol [Next]



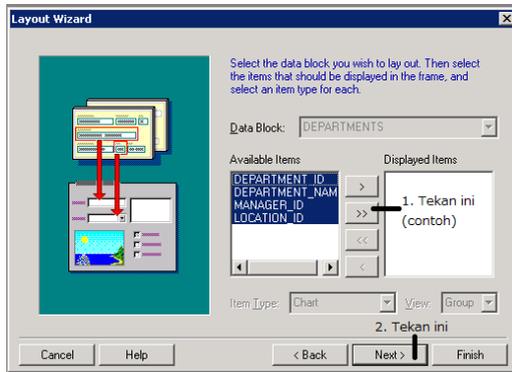
Gambar 5-9 Tampilan Awal Dialog Box Layout Wizard

7. Pada poin ini, *user* akan diminta untuk memilih *canvas* yang akan digunakan. *Canvas* adalah sebuah permukaan dari sebuah *window/form*, tempat dimana diletakkan semua obyek yang akan ditampilkan pada sebuah *window/form*. Karena sampai poin ini belum ada *canvas* yang dibuat, maka pilih [New Canvas] pada pilihan *canvas*, lalu tekan [Next].



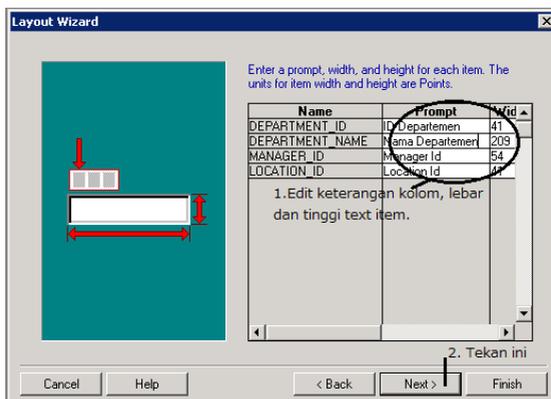
Gambar 5-10 Dialog Box Layout Wizard - Memilih Canvas

8. Pada poin ini, *user* diminta untuk memilih kolom apa saja dari tabel yang dipilih pada saat membuat *block* data, yang akan ditampilkan pada *canvas*. Sebagai contoh, tampilkan semua kolom dari tabel `departments` yang sudah dipilih pada *block* data dengan menggunakan tombol [`>>`], lalu tekan tombol [Next].



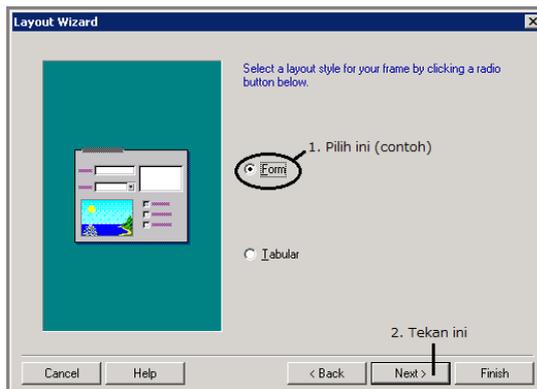
Gambar 5-11 Dialog Box Layout Wizard - Memilih Kolom yang ditampilkan

9. Pada poin ini, *user* diminta untuk menentukan keterangan yang mengikuti *textitem* dari setiap kolom yang sudah dipilih pada poin sebelumnya, sekaligus mengatur lebar dan tinggi dari masing-masing *textitem*.



Gambar 5-12 Dialog Box Layout Wizard - Setting Text Item

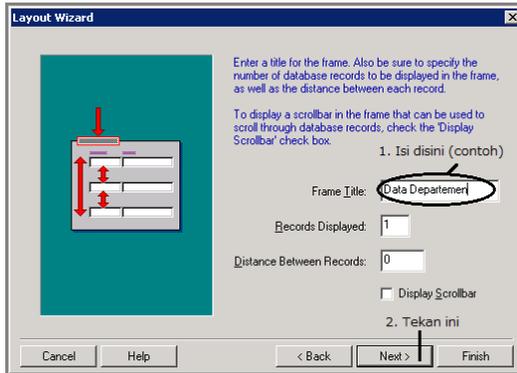
10. Pada poin ini, *user* diminta untuk menentukan *layout style* yang akan menentukan bentuk tampilan pada *canvas*. Terdapat 2 pilihan, yaitu [*Form*] dan [*Tabular*]. Pilihan *form* akan membuat tampilan pada canvas layaknya sebuah formulir. Sedangkan pilihan [*tabular*] akan membuat tampilan pada canvas layaknya sebuah tabel yang memiliki baris dan kolom. Untuk contoh, pilih [*Form*], lalu tekan [*Next*]



Gambar 5-13 Dialog Box Layout Wizard - Memilih Layout

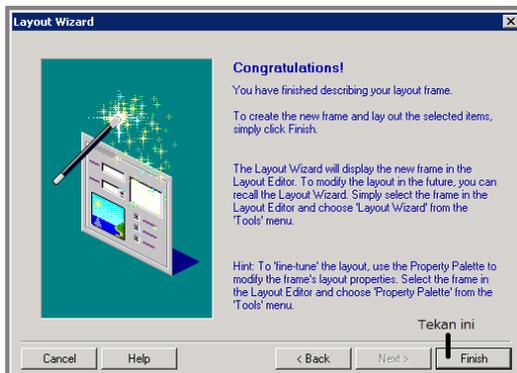
11. Pada poin ini, *user* diminta untuk menentukan frame title yang akan digunakan untuk menamai layout yang sudah dibuat, menentukan jumlah data yang ditampilkan, jarak spasi per data yang ditampilkan, dan apakah *scroll bar*

perlu ditampilkan. Untuk contoh, beri frame title “data departemen”, lalu tekan [Next].



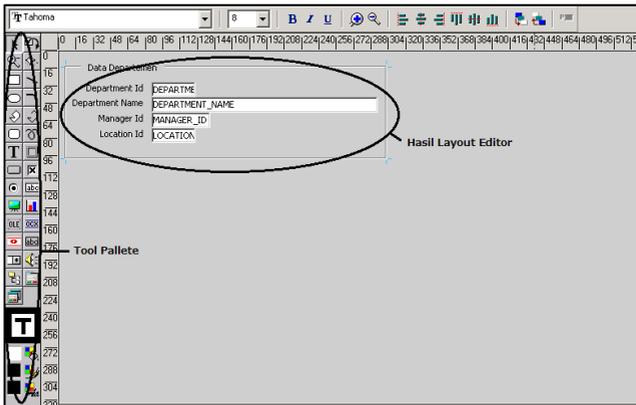
Gambar 5-14 Dialog Box Layout Wizard - Mengisi Frame Title

12. Pada poin ini, setting layout telah selesai, dan user cukup menekan tombol [Finish] untuk melihat hasil layout yang telah dibuat.



Gambar 5-15 Dialog Box Layout Wizard - Finish

13. *Layout* yang sudah dibuat akan ditampilkan pada *layout editor*. Pada *layout editor* ini juga dapat dilakukan pengaturan tampilan ulang atau penambahan obyek pada *layout* yang sudah dibuat dengan memanfaatkan obyek-obyek yang ada pada *tool pallete* di *layout editor*.

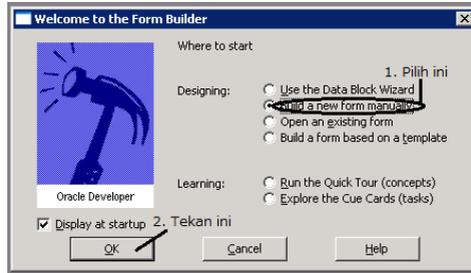


Gambar 5-16 *Layout Editor* dan *Tool Pallete*

## Creating Block Manually

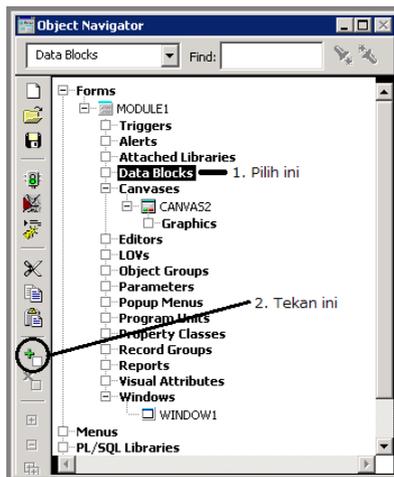
Berikut adalah penjelasan untuk membuat sebuah *block* data beserta tampilan secara manual.

1. Pada saat *software* pertama kali dijalankan, akan muncul sebuah *dialog box* untuk menentukan apakah apa yang akan dilakukan selanjutnya. Pada poin ini, pilih [Build new form manually], lalu tekan tombol [OK].



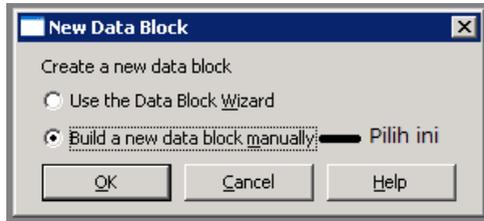
Gambar 5-17 *Dialog Box Welcome* pada *Form Builder - Build Manual*

2. Akan terbuka object navigator yang akan menampilkan sebuah struktur obyek pada sebuah modul yang sedang terbuka. Pada *object navigator* ini, pilih bagian [Data Blocks] pada modul yang terbuka (biasanya bernama MODULE1), lalu tekan tombol [Create].



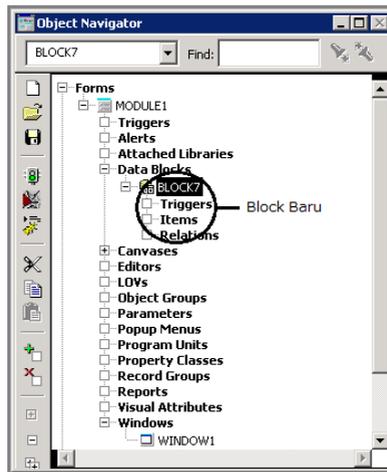
Gambar 5-18 *Object Navigator*

3. Akan muncul *dialog box new form dialog*, disini user menentukan apakah ingin membuat *block* baru secara manual atau dengan menggunakan *data block wizard*. Pilih [Build a new data block manually], lalu tekan [OK].



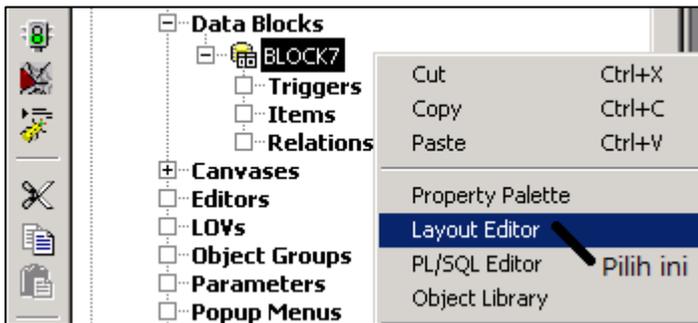
Gambar 5-19 *Dialog Box New Data Block*

4. Pada poin, telah terdapat sebuah data *block* baru pada bagian [Data Blocks] di Object Navigator.



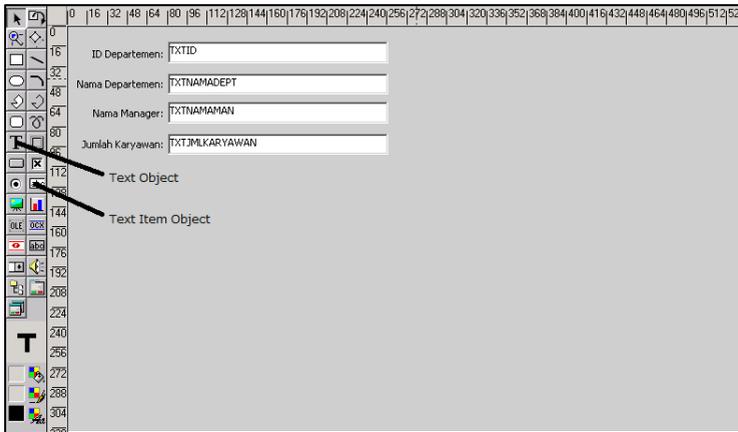
Gambar 5-20 *Block Data Baru*

5. *Data block* baru yang sudah dibuat belum memiliki obyek apapun, untuk menambahkan obyek baru, *user* dapat memanfaatkan *layout editor* untuk mendesain konten dari *data block*. Untuk mengakses *layout editor*, *right click* pada nama *block* data yang sudah dibuat, lalu pilih [Layout Editor].



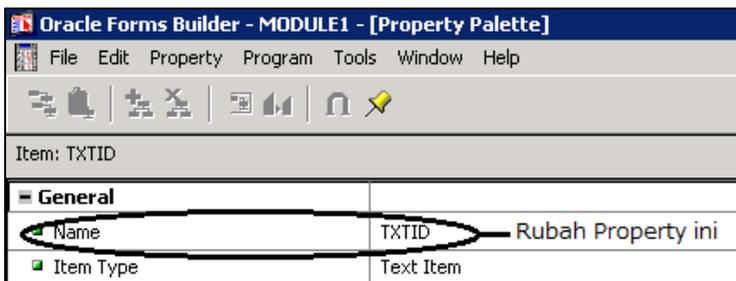
Gambar 5-21 Akses *Layout Editor* via *Object Navigator*

6. Pada *layout editor*, *user* dapat memasukkan obyek-obyek dari *tool pallete* untuk ke dalam *data block* yang sudah dibuat. Sebagai contoh, buat desain seperti pada gambar 5-22 dengan memanfaatkan obyek *text item* dan obyek *text*.



Gambar 5-22 Contoh Desain Form Manual

Untuk merubah nama dari text item, lakukan *double click* pada *text item* yang akan dirubah untuk membuka *property pallette*, lalu rubah nilai pada *property* [name].

Gambar 5-23 Merubah Nama dari *Property Palette*

## Simple Use Of The *Form*

Berikut akan dijelaskan contoh sederhana untuk melakukan pemrograman pada form yang sudah dibuat dengan menggunakan *block* data yang sudah dibuat pada pembuatan *block* secara manual sebelumnya. Kondisi nya adalah:

### 1. *User*

- ✓ Memasukkan id departemen pada `TXID`, lalu menekan ENTER.

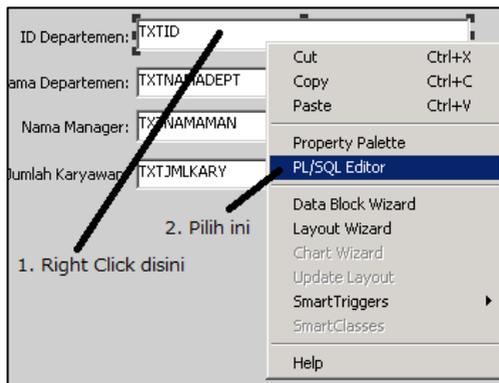
### 2. *Sistem*

- ✓ Melakukan cek apakah id departemen memang ada pada tabel departemen atau tidak. Jika tidak ada, keluarkan pesan bahwa id departemen yang dimasukkan tidak ada pada tabel `department`.
- ✓ Jika id departemen ada, maka tampilkan nama departemen pada `TXNAMADEPT`.
- ✓ Setelah menampilkan nama departemen, tampilkan juga nama lengkap manager yang memimpin departemen tersebut pada `TXNAMAMAN`. Jika departemen belum memiliki manager, maka tampilkan “-” pada *text item* tersebut.

- ✓ Setelah nama manager, tampilkan jumlah karyawan pada departemen tersebut pada `TXTJMLKARY`.

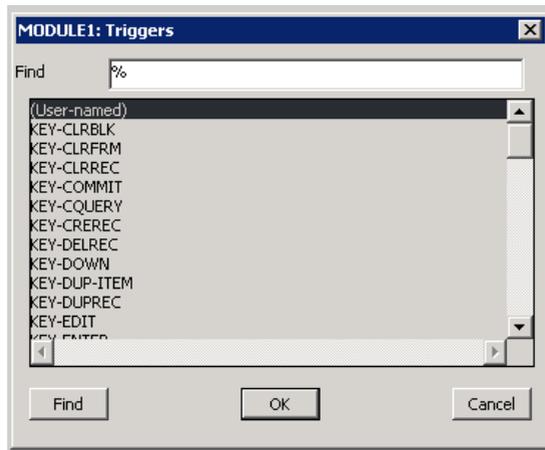
Langkah-langkah pengerjaan:

1. Untuk menerima ENTER, akan digunakan *trigger* bernama `KEY-NEXT-ITEM` pada `TXTID`. *Trigger* merupakan PL/SQL block yang ditulis untuk menampilkan tugas-tugas pada saat sebuah *event* khusus terjadi di dalam sebuah aplikasi. Sebuah trigger harus disimpan ke dalam obyek yang spesifik di dalam sebuah form. Pada kasus ini trigger `KEY-NEXT-ITEM` akan diletakkan pada *text item* `TXTID`. Untuk melakukannya:
  - a. *Right click* pada `TXTID`, lalu pilih [PL/SQL Editor].

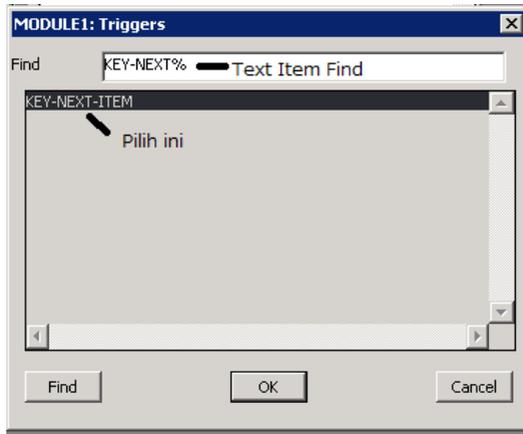


Gambar 5-24 Bagaimana Membuka PL/SQL Editor

- b. Akan muncul sebuah dialog box yang menampilkan daftar semua trigger yang bisa digunakan. Pada dialog box ini, *user* dapat melakukan pencarian nama sebuah trigger dengan memanfaatkan *text item Find*. Konsep pencarian yang digunakan pada *dialog box* ini, sama dengan pencarian dengan menggunakan *wild card (%)*. Pada dialog box ini pilih Trigger KEY-NEXT-ITEM, lalu tekan [OK].

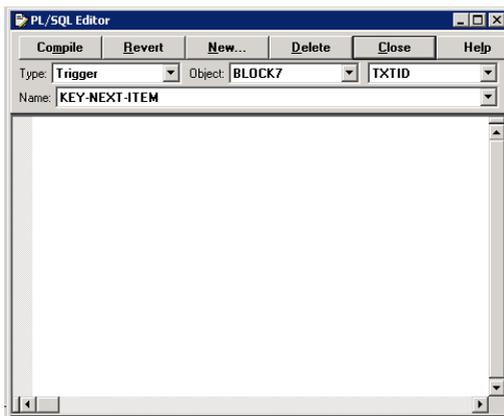


Gambar 5-25 Dialog Box Daftar Trigger pada Module



Gambar 5-26 Hasil Pencarian pada *Dialog Box* Daftar Trigger

- c. Setelah menentukan *trigger*, *software* akan membuka PL/SQL Editor untuk memasukkan blok PL/SQL/sesuai dengan trigger yang sudah ditentukan.

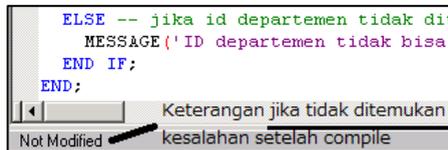


Gambar 5-27 Tampilan PL/SQL Editor

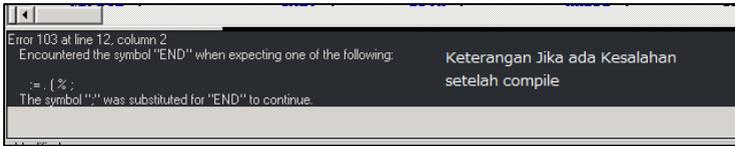
2. Lakukan pengkodean blok PL/SQL pada *PL/SQL Editor* untuk setiap proses yang sudah didefinisikan. Setelah melakukan pengkodean, tekan tombol [Compile] pada bagian atas *PL/SQL Editor*, untuk melihat apakah ada kesalahan dalam pengkodean atau tidak. Jika tidak ada kesalahan, maka pada bagian kiri bawah dari *PL/SQL Editor* akan muncul kata-kata “Not Modified”, sedangkan jika terjadi kesalahan, maka akan muncul daftar kesalahan yang terbaca pada saat compile.



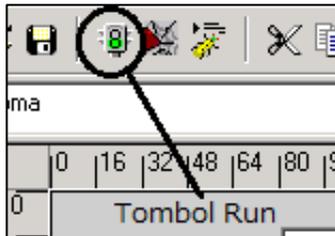
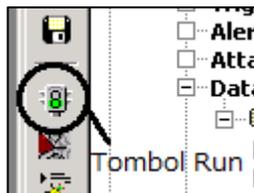
Gambar 5-28 Tombol *Compile* pada *PL/SQL Editor*



Gambar 5-29 Tidak Ada Kesalahan Setelah *Compile*

Gambar 5-30 Daftar Kesalahan Setelah *Compile*

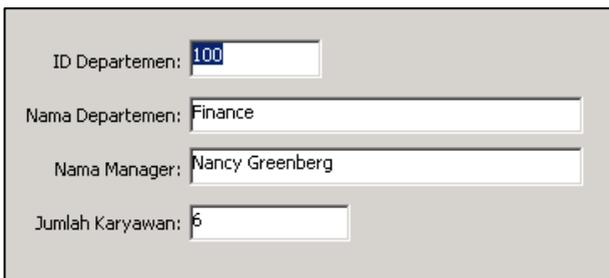
setelah melakukan selesai melakukan *compile* dan tidak ada *error* pada kode, maka lakukan *running* untuk melihat jalannya *form* yang sudah dibuat. Untuk menjalankan *form* (*running form*), gunakan tombol [run] pada *layout editor* atau pada *object navigator*.

Gambar 5-31 Tombol *Run* pada *Layout Editor*Gambar 5-32 Tombol *Run* pada *Object Navigator*

Berikut adalah salah satu susunan kode untuk memenuhi kebutuhan-kebutuhan dari *form*.

```
DECLARE
deptname VARCHAR2(30); --untuk menampung nama department
find NUMBER; --untuk pengecekan apakah department_id ada atau tidak.
manid NUMBER; --untuk menampung manager ID
manname VARCHAR(50); --> untuk menampung nama (lengkap) manager
jmlkary NUMBER; --> untuk menampung jumlah karyawan
BEGIN
--cek apakah id ada atau tidak pada tabel departments
SELECT COUNT(department_id) INTO find FROM departments WHERE department_ID = :TXTID;
IF find > 0 THEN --jika id departemen ditemukan, ambil nama department dan manager id
SELECT department_name, NVL(manager_id, NULL) INTO deptname, manid
FROM departments WHERE department_id = :TXTID;
:TXTNAMEDEPT := deptname; --tampilkan nama department pada text item
--cek apakah departemen punya manager atau tidak
IF manid IS NULL THEN
manname := '-'; --jika tidak ada, maka nama ganti '-'
ELSE
--jika ada, ambil nama dari tabel employees berdasarkan manid (sebagai employee_id)
SELECT first_name||' '||last_name INTO manname FROM employees WHERE employee_id = manid;
END IF;
:TXTNAMAMAN := manname; --tampilkan nama ke text item
--ambil jumlah karyawan dari departemen yang ditampilkan.
SELECT COUNT(employee_id) INTO jmlkary FROM employees WHERE department_id = :TXTID;
:TXTJMLKARYAWAN := jmlkary; --tampilkan pada text item
ELSE -- jika id departemen tidak ditemukan.
MESSAGE('ID departemen tidak bisa ditemukan');
END IF;
END;
```

Untuk menguji coba susunan kode di atas, apakah sudah memenuhi kebutuhan dari *form*, *user* dapat memasukkan nomor 100 dan 25 pada id departemen, lalu tekan ENTER. Jika *user* memasukkan 100 dan menekan ENTER, maka seharusnya tampilan dari *form* menjadi sebagai berikut:



ID Departemen:	<input type="text" value="100"/>
Nama Departemen:	<input type="text" value="Finance"/>
Nama Manager:	<input type="text" value="Nancy Greenberg"/>
Jumlah Karyawan:	<input type="text" value="6"/>

Gambar 5-33 Hasil Uji Coba *Form* Contoh

Jika *user* memasukkan angka 25 pada id departemen, maka akan keluar pesan “ID departemen tidak bisa ditemukan” pada bagian kiri bawah dari *window form*.

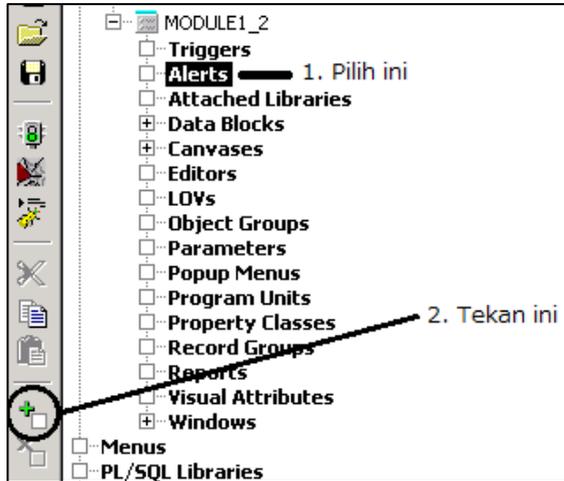


Gambar 5-34 Hasil Uji Coba Gagal *Form* Contoh

## Alert

Alert adalah sebuah *window* yang menampilkan pesan kepada *user* pada beberapa kondisi aplikasi. Alert juga adalah sebuah obyek yang tersimpan dalam sebuah module, sehingga sebelum digunakan, sebuah alert harus dibuat terlebih dahulu. Sebagai contoh, akan dibuat sebuah alert untuk menggantikan pesan “ID departemen tidak bisa ditemukan” pada *form* yang sudah dibuat sebelumnya. Langkah-langkahnya adalah:

1. Pada object navigator, pilih bagian [Alerts], lalu tekan tombol [Create].



Gambar 5-35 Bagaimana Membuat Alert

2. Akan muncul *alert* baru pada bagian [Alerts], lalu lakukan *double-click* pada alert tersebut untuk membuka *property pallette* dari *alert*.



Gambar 5-36 Alert Baru

3. Pada property palette dari alert, lakukan setting sebagai berikut:

- [Title] : “Peringatan”
- [Message]: “ID departemen tidak ditemukan”
- [Alert Style]: “Stop”
- [Button 1 Label]: “OK”
- [Button 2 Label]: kosong

Functional	
Title	Informasi
Message	ID departemen tidak ditemukan
Alert Style	Stop
Button 1 Label	OK
Button 2 Label	

Gambar 5-37 Setting *Property Palette* pada *Alert*

Setelah selesai lakukan perubahan pada susunan kode PL/SQL pada *trigger* KEY-NEXT-ITEM di text item TXTID. Untuk memanggil *alert*, harus ada sebuah variabel bertipe NUMBER yang digunakan untuk menampilkan *alert* sekaligus menampung nilai yang dikembalikan oleh *alert* tersebut. Lalu *alert* dipanggil dengan perintah `SHOW_ALERT([nama alert])`.

```

DECLARE
deptname VARCHAR2(30); --untuk menampung nama department
find NUMBER; --untuk pengecekan apakah department_id ada atau tidak.
manid NUMBER; --untuk menampung manager ID
manname VARCHAR(50); --> untuk menampung nama (lengkap) manager
jmlkary NUMBER; --> untuk menampung jumlah karyawan
v_al NUMBER; --> untuk menampilkan alert
BEGIN
--cek apakah id ada atau tidak pada tabel departments
SELECT COUNT(department_id) INTO find FROM departments WHERE department_id = :TXTID;
IF find > 0 THEN --jika id departemen ditemukan, ambil nama department dan manager id
SELECT department_name, NVL(manager_id, NULL) INTO deptname, manid
FROM departments WHERE department_id = :TXTID;
:TXINAMADEPT := deptname; --tampilkan nama department pada text item
--cek apakah departemen pny manager atau tidak
IF manid IS NULL THEN
manname := '-'; --jika tidak ada, maka nama ganti '-'
ELSE
--jika ada, ambil nama dari tabel employees berdasarkan manid (sebagai employee_id)
SELECT first_name||' '||last_name INTO manname FROM employees WHERE employee_id = manid;
END IF;
:TXINAMAMAN := manname; --tampilkan nama ke text item
--ambil jumlah karyawan dari departemen yang ditampilkan.
SELECT COUNT(employee_id) INTO jmlkary FROM employees WHERE department_id = :TXTID;
:TXIJMLKARYAWAN := jmlkary; --tampilkan pada text item
ELSE --jika id departemen tidak ditemukan.
--MESSAGE('ID departemen tidak bisa ditemukan');
v_al := SHOW_ALERT('ALERT21'); --memanggil alert
--MESSAGE(v_al);
END IF;
END;

```

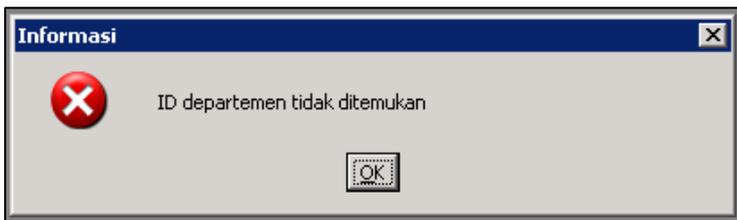
Bagian yang diberi garis adalah bagian yang ditambah/diubah. Terdapat juga satu model susunan kode lain yang lebih sederhana.

```

DECLARE
deptname VARCHAR2(30); --untuk menampung nama department
manid NUMBER; --untuk menampung manager ID
manname VARCHAR(50); --> untuk menampung nama (lengkap) manager
jmlkary NUMBER; --> untuk menampung jumlah karyawan
v_al NUMBER; --> untuk menampilkan alert
BEGIN
--ambil nama departemen dan manager_id dari id departemen yang dimasukkan.
SELECT department_name, NVL(manager_id, NULL) INTO deptname, manid
FROM departments WHERE department_id = :TXTID;
:TXINAMADEPT := deptname; --tampilkan nama department pada text item
--cek apakah departemen pny manager atau tidak
IF manid IS NULL THEN
manname := '-'; --jika tidak ada, maka nama ganti '-'
ELSE
--jika ada, ambil nama dari tabel employees berdasarkan manid (sebagai employee_id)
SELECT first_name||' '||last_name INTO manname FROM employees WHERE employee_id = manid;
END IF;
:TXINAMAMAN := manname; --tampilkan nama ke text item
--ambil jumlah karyawan dari departemen yang ditampilkan.
SELECT COUNT(employee_id) INTO jmlkary FROM employees WHERE department_id = :TXTID;
:TXIJMLKARYAWAN := jmlkary; --tampilkan pada text item
EXCEPTION --jika terjadi kesalahan
WHEN no_data_found THEN
v_al := SHOW_ALERT('ALERT21'); --memanggil alert
END;

```

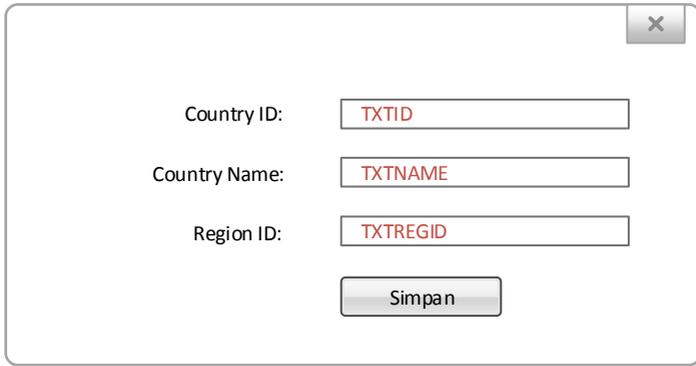
Pada susunan kode ini, dengan memanfaatkan bagian *exception handling* (keyword `EXCEPTION`) yang diberi kondisi `no_data_found`, *alert* akan muncul jika perintah `SELECT` yang dijalankan, tidak mengembalikan data apapun (tidak mengembalikan data bukan berarti `NULL`). Untuk menguji coba, silahkan masukkan "25" pada id departemen.



Gambar 5-38 Hasil Uji Coba *Alert*

## Practice

Buatlah sebuah *form* untuk memanipulasi (menambahkan data baru dan mengubah data yang ada) data negara (`countries`). Tampilan dari *form* adalah sebagai berikut:



Country ID:

Country Name:

Region ID:

Jalannya form adalah sebagai berikut:

1. *User* memasukkan data negara lalu menekan tombol simpan.
2. Jika terdapat `country_id` yang kembar dengan yang dimasukkan *user*, maka lakukan update terhadap data, berdasarkan `country_id`.
3. Jika tidak terdapat `country_id` kembar, maka lakukan insert data negara baru.

## Stage 6: Oracle Developer (Multiple Blocks and LOV)

オラクルディベロッパーII  
「マルチプルブロックとLOV」

“You're not weak, are you? You're strong, you're just not ready to accept it.”

人

What you'll learn:

LOV

Multiple Block

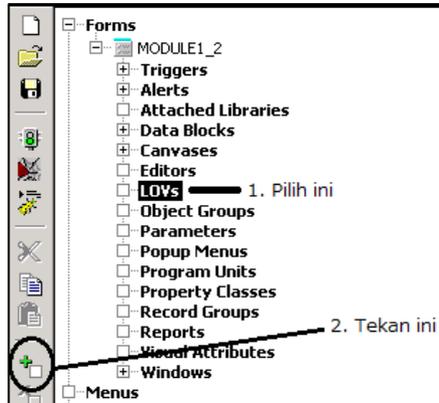
## LOV (List Of Values)

LOV merupakan sebuah kumpulan field di dalam sebuah tabel atau beberapa tabel yang dikelompokkan menjadi sebuah kesatuan. LOV menggunakan sebuah *record group* sebagai sumber data dan LOV dapat dibentuk secara manual dan secara otomatis melalui *wizard*. Jika sebuah LOV dibuat secara manual, maka *record group* yang akan digunakan oleh LOV tersebut harus dibuat terlebih dulu, sedangkan jika sebuah LOV dibuat melalui *wizard*, maka user dapat menentukan apakah akan membuat record group baru atau menggunakan record group yang sudah ada. Sebuah LOV digunakan untuk menampilkan daftar nilai yang nilainya dapat dipilih dan dikembalikan ke *form* yang memanggil.

### Creating LOV Through The Wizard

Berikut adalah langkah-langkah membuat LOV melalui *wizard*, dengan memanfaatkan form yang sudah dibuat pada *stage* 5.

1. Pada *object navigator*, pilih bagian [LOVs] lalu tekan tombol [Create].



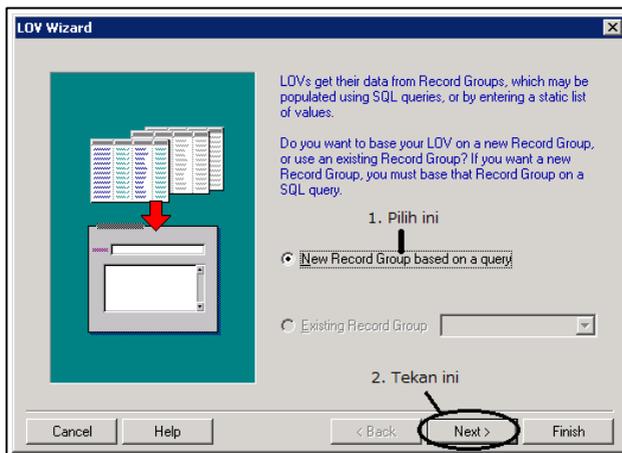
Gambar 6-1 Bagaimana Membuat LOV Baru

2. Akan muncul *dialog box new list-of-values* yang digunakan untuk menentukan apakah LOV akan dibuat dengan menggunakan *wizard* atau akan dibuat secara manual. Pada poin ini, pilih [Use The LOV Wizard].



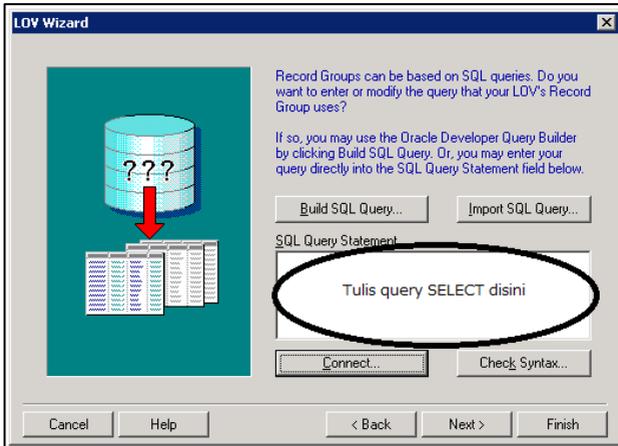
Gambar 6-2 Dialog Box New List-of-Values

3. Akan terbuka *dialog box LOV Wizard*, pada poin ini, user diminta untuk memilih record group yang akan digunakan pada LOV, apakah membuat *record group* baru atau menggunakan *record group* yang sudah ada. Pilih [New Record Group based on a query] lalu tekan tombol [Next ].



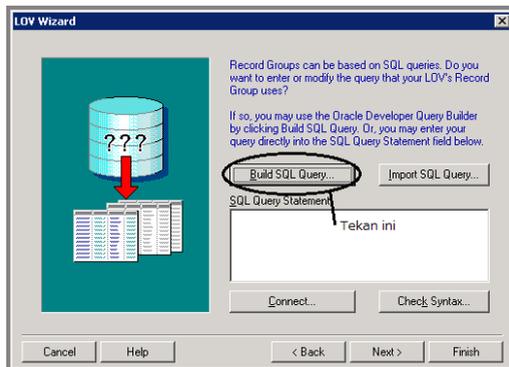
Gambar 6-3 LOV Wizard – Memilih Record Group

4. Pada poin ini user diminta untuk memasukkan *query SELECT* yang akan digunakan sebagai dasar mengambil data dari *database* (pembuatan *record group*).
  - a. User dapat memasukkan langsung *query SELECT* pada *text box* yang sudah ada, lalu menekan tombol [Next].



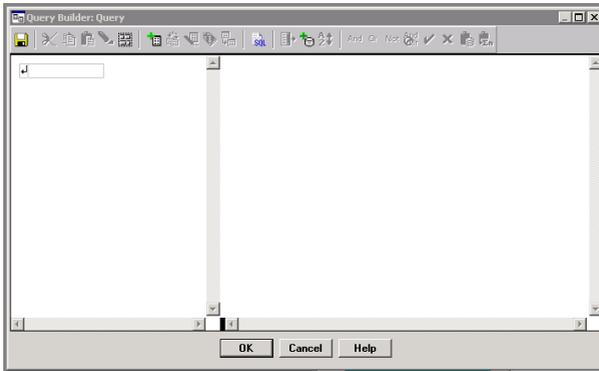
Gambar 6-4 LOV Wizard - Text Box SELECT Query

- b. Selain, memasukkan langsung query SELECT, user juga dapat membangun sendiri query SELECT dengan memanfaatkan tombol [Build SQL Query].

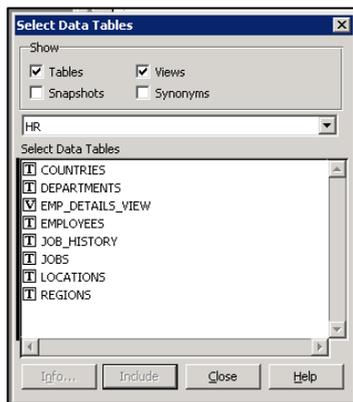


Gambar 6-5 LOV Wizard - Tombol Build SQL Query

Setelah menekan tombol [Build SQL Query], akan muncul sebuah window *query builder* dan 1 buah *dialog box* yang menampilkan seluruh tabel dan view yang dapat diakses oleh *user* pada sebuah *server*.

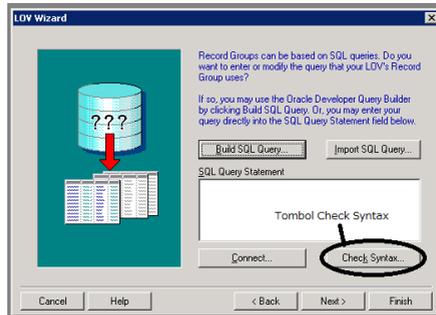


Gambar 6-6 Window Query Builder



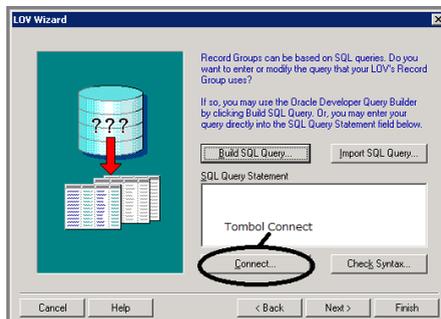
Gambar 6-7 Dialog Box Select Data Tables

- c. Setelah membangun atau menuliskan *query* SELECT, user dapat menekan tombol [Check Syntax] untuk memeriksa apakah query yang sudah dimasukkan, memiliki kesalahan pengkodean atau tidak.



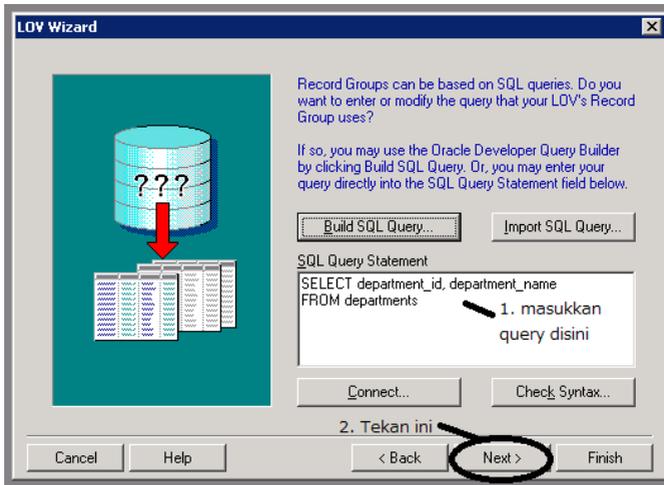
Gambar 6-8 LOV Wizard - Tombol Check Syntax

- d. Tombol [Connect] digunakan untuk membuka *dialog box* untuk melakukan koneksi ke Oracle database seperti pada Gambar 5-5.



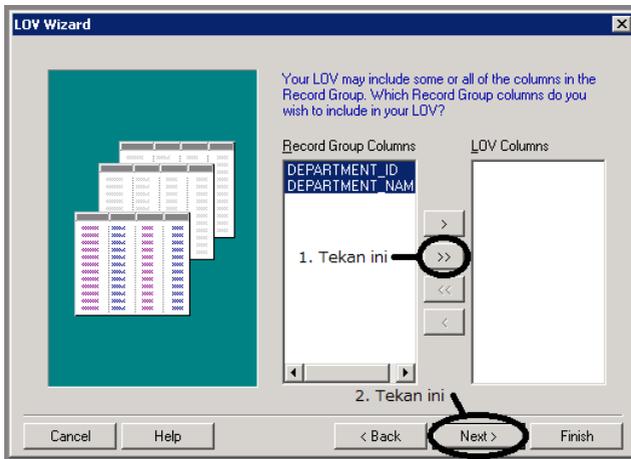
Gambar 6-9 LOV Wizard - Tombol Connect

- e. Masukkan *query* `SELECT` untuk mengambil daftar id departemen dan nama departemen dari seluruh departemen yang ada, sebagai dasar record group, dan tekan tombol [Next].



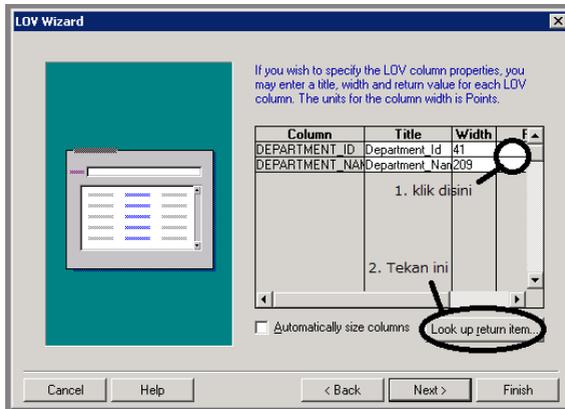
Gambar 6-10 Pembuatan Record Group

5. Pada poin ini user diminta untuk memilih kolom apa saja pada *record group* yang akan ditampilkan pada LOV. Pilih semua kolom dengan menekan tombol [`>>`], lalu tekan tombol [Next].



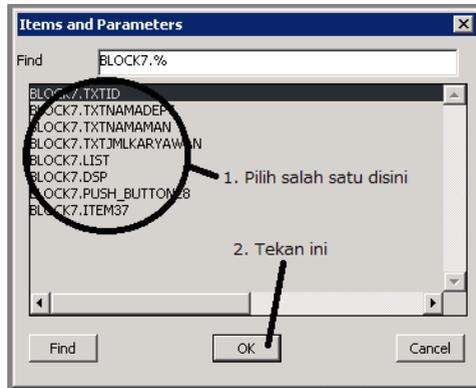
Gambar 6-11 LOV Wizard - Memilih Kolom LOV

6. Pada poin user diminta untuk menentukan item yang digunakan untuk menampung nilai yang dikembalikan oleh kolom pada LOV.
  - a. Untuk menentukan item yang akan digunakan oleh setiap kolom pada daftar kolom, pertama-tama pilih dulu kolom yang akan dikembalikan nilainya dengan mengarahkan cursor ke kolom [Return Value] dari baris kolom, lalu tekan tombol [Look up return item].



Gambar 6-12 Bagaimana Menentukan *Item* untuk *Return Value*

- b. Setelah tombol [Look up return item] ditekan, maka akan muncul sebuah dialog box *items and parameters* yang memunculkan semua daftar item dan parameter pada sebuah *module* yang bisa digunakan untuk menerima nilai yang akan dikembalikan oleh LOV. Disini user dapat memilih item yang apa yang akan digunakan untuk menerima nilai yang dikembalikan, lalu tekan tombol [Ok]. Jika daftar item yang ditampilkan terlalu banyak, user dapat menggunakan fungsi *find* yang tersedia. Fungsi *find* yang digunakan pada *dialog box* ini, sama dengan pencarian yang menggunakan *wild card* (%)

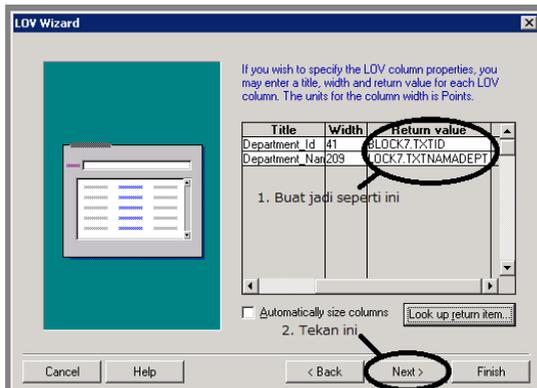


Gambar 6-13 Dialog Box Items and Parameters

c. Pada poin ini, buat *return value* sebagai berikut:

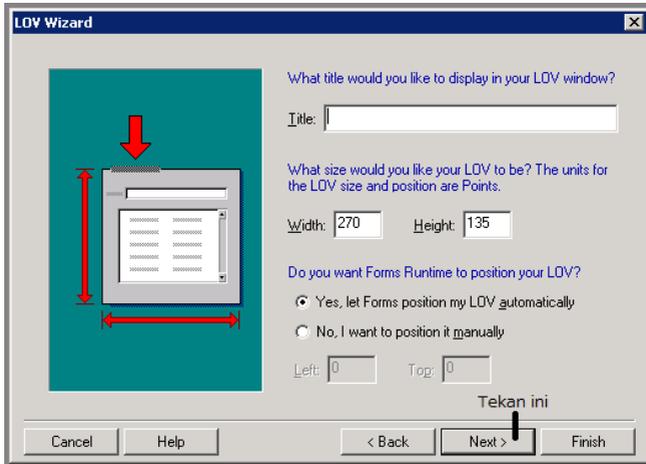
- Department\_ID = TXTID
- Department\_NAME =- TXTNAMADEPT

Lalu tekan tombol [Next].



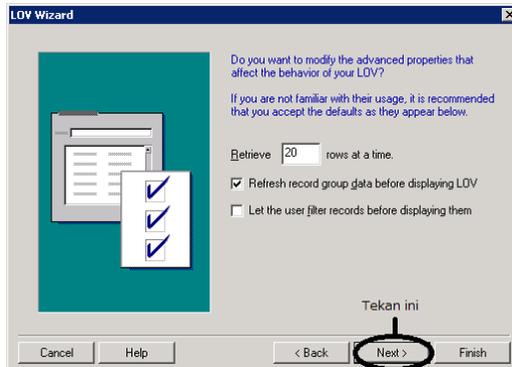
Gambar 6-14 Contoh Setting Return Item

- Pada poin ini, user diminta untuk mengatur judul dari LOV, ukuran window LOV, dan posisi LOV pada saat dipanggil. Pada poin ini, langsung tekan tombol [Next].



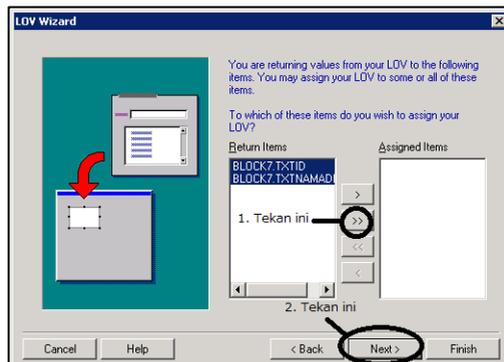
Gambar 6-15 LOV Wizard - Pengaturan Judul, Ukuran Window, dan Posisi LOV

- Pada poin ini, user diminta untuk memasukkan jumlah baris data yang diambil oleh LOV pada saat dipanggil, status apakah record group perlu melakukan *refresh* sebelum LOV ditampilkan, dan apakah user harus melakukan pencarian lebih dulu sebelum data ditampilkan. pada poin ini, langsung tekan tombol [Next].



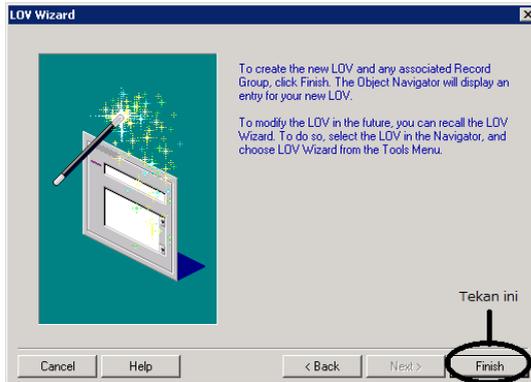
Gambar 6-16 LOV Wizard - Pengaturan Jumlah Data, Refresh Record Group, dan Pencarian Oleh User

9. Pada poin ini user diminta untuk menentukan kembali, mana item yang akan ditampilkan berdasarkan LOV. Pada poin ini, pilih semua item dengan menekan tombol [>>], lalu tekan tombol [Next].



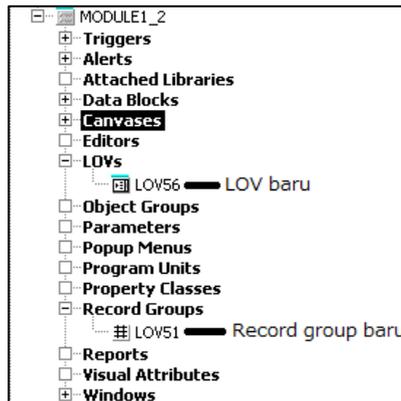
Gambar 6-17 LOV Wizard - Item Assignment

10. Pada poin ini, user cukup menekan tombol [Finish].



Gambar 6-18 LOV Wizard – Finish

Akan muncul sebuah LOV baru dan sebuah *record group* baru pada bagian [LOVs] dan [Record Groups] di *object navigator*.

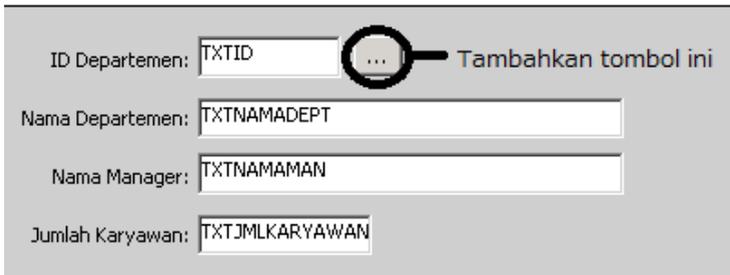


Gambar 6-19 LOV dan *Record Group* Hasil Penggunaan LOV Wizard

## Using LOV in PL/SQL Code

Berikut adalah contoh penggunaan LOV dengan memanfaatkan form yang sudah dibuat pada *stage* 5.

1. Tambahkan sebuah tombol [...] pada form yang sudah dibuat pada *stage* 5.



The image shows a form with four text input fields and a button. The fields are labeled 'ID Departemen', 'Nama Departemen', 'Nama Manager', and 'Jumlah Karyawan'. The button is labeled 'Tambahkan tombol ini' and has a magnifying glass icon over it.

Gambar 6-20 Tombol (Button) Tambah

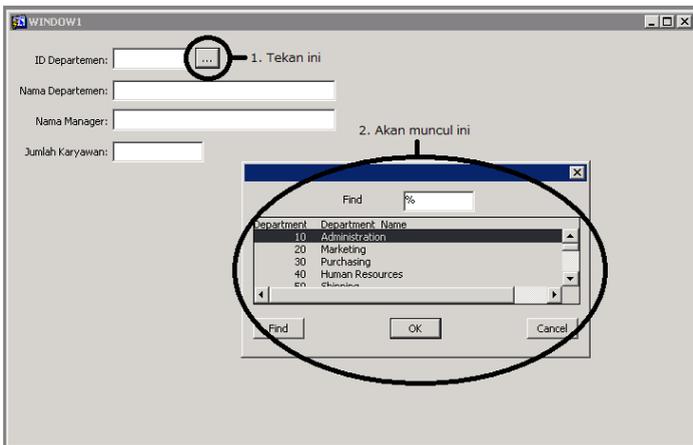
2. Tambahkan *trigger* WHEN-BUTTON-PRESSED pada tombol yang sudah ditambahkan. Lalu masukkan kode di bawah ini

```
DECLARE
    v_lov BOOLEAN;
BEGIN
    v_lov := SHOW_LOV('LOV56');
END;
```

Pada susunan kode ini, untuk memanggil LOV dibutuhkan sebuah variabel bertipe BOOLEAN untuk menampung

apakah LOV yang dipanggil telah mengembalikan nilai atau belum.

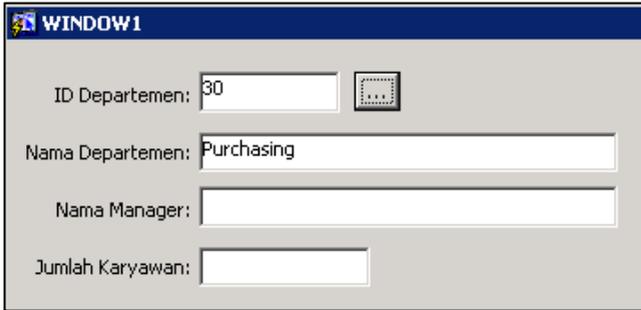
3. Jalankan Form, dan tekan tombol yang sudah dibuat. Akan muncul sebuah *dialog box* yang menampilkan daftar id departemen dan nama departemen dari semua departemen yang tersimpan di tabel DEPARTMENTS.



Gambar 6-21 Hasil Pemanggilan LOV

4. Untuk memilih data yang akan dikembalikan, user cukup melakukan *double-click* pada baris data yang akan dikembalikan nilainya, atau juga dapat memilih baris data, lalu menekan tombol [OK]. Jika data yang ditampilkan terlalu banyak, user dapat memanfaatkan fungsi pencarian yang sudah disediakan pada LOV. Pencarian

yang digunakan pada *dialog box* LOV ini sama dengan pencarian dengan menggunakan *wild card* (%).



The image shows a screenshot of a dialog box titled "WINDOW1". It contains four input fields with labels: "ID Departemen:" with the value "30", "Nama Departemen:" with the value "Purchasing", "Nama Manager:" which is empty, and "Jumlah Karyawan:" which is empty. There is a small icon to the right of the "ID Departemen:" field.

Gambar 6-22 Hasil Pemilihan Data dari LOV

Catatan tentang penggunaan LOV:

Pada saat sebuah LOV dipanggil, user tidak akan bisa menggunakan *form* atau *window* yang memanggil LOV, sampai LOV tersebut ditutup, dengan kata lain, proses dari *form* atau *window* tersebut akan berhenti sampai LOV ditutup.

## Working with Multiple-Block

Relasi *master detail* adalah koneksi antara dua buah *block* yang merefleksikan hubungan *primary* dan *foreign key* antara tabel yang berasosiasi dengan *block* tersebut. *Block master* adalah *block* yang berbasis pada tabel

yang memiliki *primary key* dan *block detail* berbasis pada tabel dengan *foreign key*.

### Form Design Example

Contoh penerapan *master detail* atau beberapa *block* dalam satu *form* adalah sebagai berikut:

1. Pertama-tama dibuat dulu sebuah form dengan tampilan sebagai berikut:

A screenshot of a form design window. At the top left, there is a label 'Department Name:' followed by a text box containing the text 'LSTDEPTNAME'. The text box has a small downward-pointing arrow on its right side, indicating it is a dropdown menu. The rest of the form area is empty and greyed out.

Gambar 6-23 Tahap Awal Multiple Block

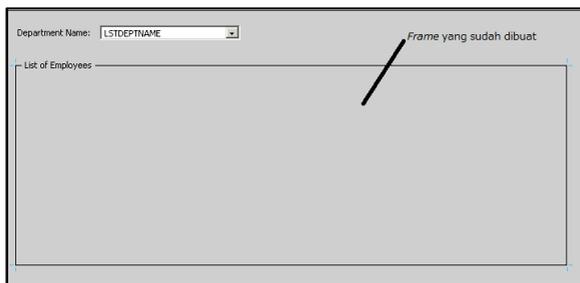
Komponen yang digunakan adalah:

- a. Text
- b. List item, dengan property sebagai berikut:
  - ✓ [Name] = LSTDEPTNAME
  - ✓ [List Style] = Combo Box

2. Setelah membuat form di atas (beserta *block*-nya), dibuat sebuah *block* baru (disarankan secara manual).
3. Setelah membuat *block* baru, buka *layout editor*, lalu tambahkan *frame* dari *tool pallete*, lalu atur property *frame* menjadi seperti berikut:
  - [Layout Data Block] : isi dengan *block* kedua yang sudah dibuat
  - [Layout Style] : Tabular
  - [Number of Record Displayed] : 10
  - [Background Color] : gray20
  - [Frame Title]: List Of Employees
  - [Show Scroll Bar]: Yes

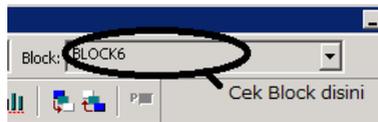


Gambar 6-24 Icon Frame pada Tool Pallette



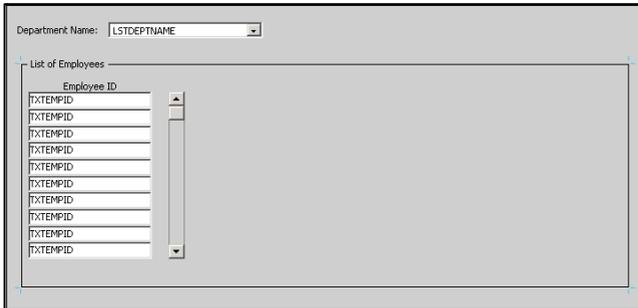
Gambar 6-25 Frame

4. Setelah membuat *frame*, tambahkan sebuah *text item* ke dalam *frame*, dengan catatan, pada saat memasukkan *text item*, *block* data yang dipilih adalah *block* data kedua (*text item* masuk pada *block* data kedua pada *object navigator*) (lihat pada bagian [Block] di *layout editor*). Setelah menambahkan, atur *property* dari *text item* menjadi sebagai berikut:
  - a. [Name]: TXTEMPID
  - b. [Prompt]: Employee ID
  - c. [Prompt Justification]: Center
  - d. [Prompt Attachment Edge]: Top
  - e. [Prompt Alignment]: Center



Gambar 6-26 *Current Block* pada *Layout Editor*

Setelah *text item* dimasukkan, maka seharusnya akan muncul 10 *text item* dengan nama yang sama di dalam *frame* yang sudah dibuat. Jika belum, maka lakukan *right-click* pada *frame* yang sudah dibuat, lalu pilih [Update Layout].

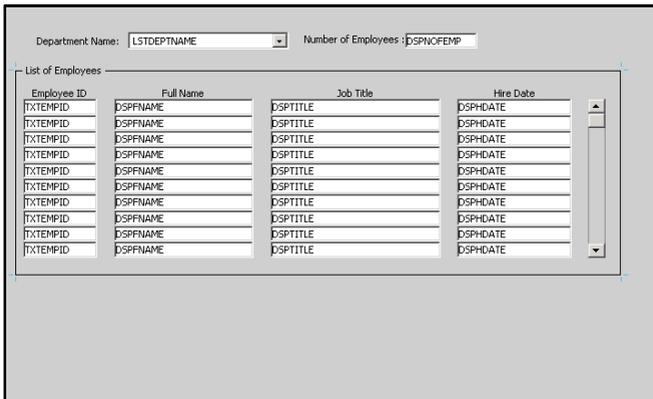


Gambar 6-27 Tampilan *Form* setelah *Text Item* ditambahkan

5. Setelah memasukkan *text item*, masih pada *block* yang sama, masukkan tiga *display item*, masing-masing dengan property sebagai berikut:
  - a. [Name]: DSPFNAME  
[Maximum Length]: 50  
[Prompt]: Full Name  
[Prompt Justification]: Center  
[Prompt Attachment Edge]: Top  
[Prompt Alignment]: Center
  - b. [Name]: DSPTITLE  
[Maximum Length]: 50  
[Prompt]: Job Title  
[Prompt Justification]: Center



[Prompt Alignment] : Start.



Department Name: LSTDEPTNAME Number of Employees: DSPNOFEMP

List of Employees

Employee ID	Full Name	Job Title	Hire Date
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE
TXTEMPID	DSPFNAME	DSPITITLE	DSPHDATE

Gambar 6-29 Tampilan *Form* setelah *Display Item* dimasukkan

### Code Example

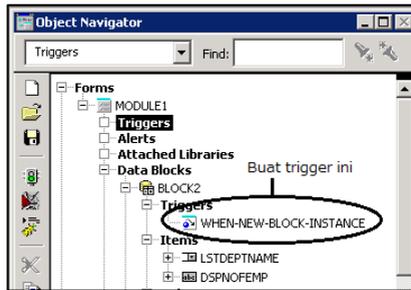
Setelah desain tampilan form sudah jadi, berikutnya adalah mengubah kebutuhan dari form menjadi susunan kode. Kebutuhan dari form adalah sebagai berikut:

- ✓ Pada saat *form* dijalankan, maka LSTDEPTNAME harus terisi dengan nama dari seluruh departemen yang ada.
- ✓ Pada saat salah satu nama departemen dipilih, maka akan keluar daftar seluruh karyawan yang bekerja di departemen tersebut (poin yang ditampilkan,

disesuaikan dengan kolom), lalu tampilkan juga jumlah karyawan pada departemen tersebut.

Untuk menyelesaikan kebutuhan ini:

1. Tambahkan sebuah *trigger* WHEN NEW BLOCK INSTANCE pada *block* pertama yang sudah dibuat.



Gambar 6-30 *Trigger* WHEN-NEW-BLOCK-INSTANCE

2. Pada *trigger* WHEN-NEW-BLOCK-INSTANCE ini, masukkan susunan kode berikut untuk mengisi *list item* pada saat *form* dipanggil.

```
DECLARE
deptrg RECORDGROUP;
deptgcol GROUPCOLUMN;
ins_ok NUMBER;
BEGIN
deptrg := find_group('deptc');
IF id_null(deptrg) THEN
deptrg := CREATE_GROUP('deptc');
deptgcol := ADD_GROUP_COLUMN (deptrg, 'l_deptname', CHAR_COLUMN, 100);
deptgcol := ADD_GROUP_COLUMN (deptrg, 'l_deptid', CHAR_COLUMN, 4);
ins_ok := POPULATE_GROUP_WITH_QUERY(deptrg, 'SELECT department_name, department_id
FROM departments');
POPULATE_LIST('BLOCK2.LSTDEPTNAME', 'deptc');
END IF;
END;
```

Dengan susunan kode di atas, maka list item akan menampilkan nama departemen pada daftar pilihan, dan akan mengembalikan nilai berupa id departemen.

- Setelah melakukan pengkodean pada trigger WHEN NEW BLOCK INSTANCE, buatlah sebuah *trigger* bernama WHEN LIST CHANGED pada list item yang sudah dibuat.



Gambar 6-31 *Trigger* WHEN-LIST-CHANGED

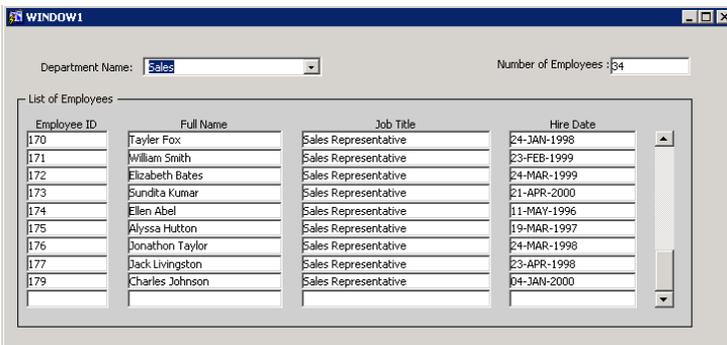
- Pada trigger WHEN-LIST-CHANGED, masukkan susunan kode berikut untuk menampilkan daftar karyawan pada departemen tersebut.

```

DECLARE
CURSOR cemplist IS SELECT e.employee_id, e.first_name || ' ' || e.last_name AS full_name,
                        j.job_title, TO_CHAR(e.hire_date, 'dd-MON-yyyy') AS hired
FROM employees e, jobs j
WHERE (e.job_id = j.job_id) AND e.department_id = RTRIM(:LSTDEPTNAME);
emprec cemplist%ROWTYPE;
nofemp NUMBER;
BEGIN
GO_BLOCK('BLOCK2');
BEGIN
CLEAR_BLOCK(NO_COMMIT);
OPEN cemplist;
LOOP
FETCH cemplist INTO emprec;
EXIT WHEN cemplist%NOTFOUND;
:ITXTEMPID := emprec.employee_id;
:DSPFNAME := emprec.full_name;
:DSPTITLE := emprec.job_title;
:DSPHDATE := emprec.hired;
NEXT_RECORD;
END LOOP;
CLOSE cemplist;
END;
GO_BLOCK('BLOCK2');
BEGIN
SELECT COUNT(employee_id) INTO nofemp FROM employees
WHERE department_id = RTRIM(:LSTDEPTNAME);
:DSPNOFEMP := nofemp;
END;
END;

```

Pada susunan kode ini, untuk menampilkan daftar karyawan pada departemen, digunakan sebuah *cursor*, lalu untuk mengganti baris data (agar bisa menampilkan semua karyawan pada departemen tersebut), digunakan sebuah perintah bernama `NEXT_RECORD`. Pada susunan kode ini, terdapat penerapan dari blok PL/SQL yang bersarang pada blok PL/SQL lain, untuk membedakan proses pada sebuah *block* dengan proses pada *block* lain, sedangkan untuk berpindah antara 1 *block* dengan *block* lain, digunakan perintah `GO_BLOCK` yang diikuti nama *block* yang dituju.



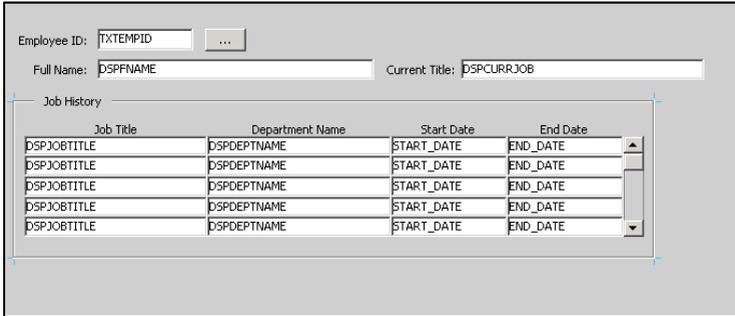
Employee ID	Full Name	Job Title	Hire Date
170	Taylor Fox	Sales Representative	24-JAN-1998
171	William Smith	Sales Representative	23-FEB-1999
172	Elizabeth Bates	Sales Representative	24-MAR-1999
173	Sundita Kumar	Sales Representative	21-APR-2000
174	Ellen Abel	Sales Representative	11-MAY-1996
175	Alyssa Hutton	Sales Representative	19-MAR-1997
176	Jonathon Taylor	Sales Representative	24-MAR-1998
177	Jack Livingston	Sales Representative	23-APR-1998
179	Charles Johnson	Sales Representative	04-JAN-2000

Gambar 6-32 Hasil Uji Coba *Multiple Block*

Cat: Jika terdapat 2 atau lebih *block* yang melakukan pemrosesan, lakukan perpindahan *block* sebelum menjalankan proses.

## Practice

Buatlah Form Sebagai Berikut:



Employee ID:  ...

Full Name:  Current Title:

Job History

Job Title	Department Name	Start Date	End Date
DSPJOBTITLE	DSPDEPTNAME	START_DATE	END_DATE
DSPJOBTITLE	DSPDEPTNAME	START_DATE	END_DATE
DSPJOBTITLE	DSPDEPTNAME	START_DATE	END_DATE
DSPJOBTITLE	DSPDEPTNAME	START_DATE	END_DATE
DSPJOBTITLE	DSPDEPTNAME	START_DATE	END_DATE

Ketentuan Form:

1. Buatlah sebuah LOV yang mengembalikan `employee_id`, nama lengkap karyawan (`first_name` dan `last_name`) dan `job_title` dari karyawan.
2. Pada saat tombol (...) ditekan, panggil LOV yang sudah dibuat.
3. Setelah memilih data `employee` dari LOV, maka akan tampil juga `job title`, `department_name`, `start_date`, dan `end_date` dari pekerjaan yang sudah pernah diambil oleh karyawan tersebut.

## Stage 7: Oracle Developer (*Program Unit + Multiple Blocks II*)

オラクルディベロッパーII

「プログラムユニットとマルチプルブロック II」

“Three steps away..., no reason to back down right now”

人

What You'll Learn:

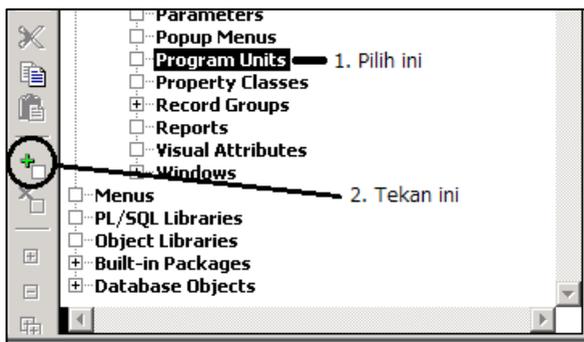
Using Program Unit

Manipulating Data Inside Detail Block

## Using Program Unit

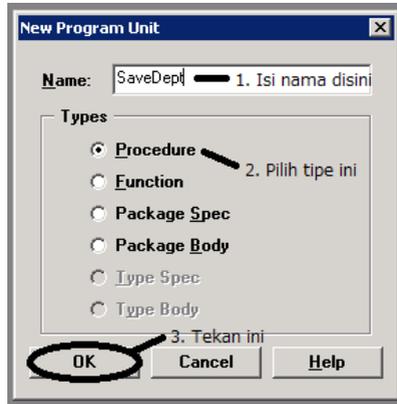
*Program unit* merupakan sebuah obyek yang merepresentasikan sebuah nama unit dari sekumpulan perintah PL/SQL yang membentuk fungsi tertentu. Program unit adalah bagian yang menyimpan semua subprogram yang ada pada sebuah *module*. Subprogram yang ada pada sebuah *module* akan tidak tersimpan pada database, tetapi tersimpan pada *module* tersebut. Terdapat beberapa macam tipe program unit yang dapat dibuat, yaitu *procedure*, *function*, *package (spec dan body)*, dan *type (spec dan body)*. Contoh membuat sebuah program unit:

1. Pilih bagian [Program Units] pada *object navigator*, lalu tekan tombol [Create].



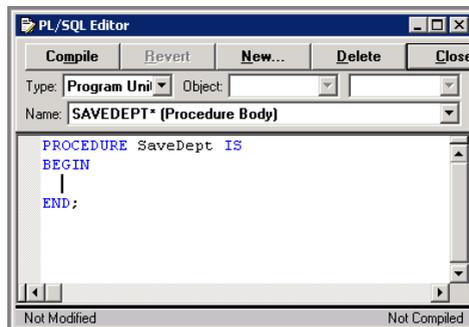
Gambar 7-1 Bagaimana Membuat Sebuah *Program Unit*

2. Akan muncul sebuah dialog box *new program unit* yang berisi jenis *program unit* yang dapat dibuat. Pada poin ini pilih *procedure* dan beri nama *SaveDept*.



Gambar 7-2 Dialog Box New Program Unit

3. Akan muncul sebuah PL/SQL Editor untuk melakukan pengkodean pada *procedure* yang sudah dibuat.



Gambar 7-3 PL/SQL Editor via Program Unit

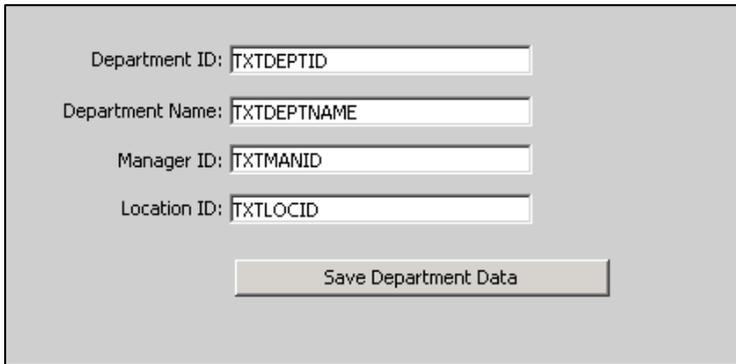
4. Beri procedure tersebut 5 buah parameter:

- [pDeptID] [IN] [NUMBER]
- [pDeptName] [IN] [Varchar2]
- [pManID] [IN] [NUMBER]
- [pLocID] [IN] [NUMBER]
- [pSuccess] [Out] [Number]

Procedure ini berfungsi untuk menyimpan data baru, sekaligus mengubah data yang sudah ada berdasarkan *DEPARTMENT\_ID* yang dimasukkan pada parameter. Pada saat procedure dijalankan, akan ada pengecekan apakah *DEPARTMENT\_ID* yang dimasukkan sudah ada, atau belum, jika belum, maka lakukan *INSERT*, sedangkan jika sudah, lakukan *UPDATE*.

```
PROCEDURE SaveDept (pDeptID IN NUMBER, pDeptName IN VARCHAR2, pManID IN NUMBER, pLocID IN NUMBER, pSuccess OUT NUMBER) IS
v_find NUMBER :=0;
BEGIN
SELECT COUNT(department_id) INTO v_find FROM departments WHERE department_id = pDeptID;
IF v_find > 0 THEN
UPDATE departments SET department_name = pDeptName, manager_id = pManID, location_id = pLocID
WHERE department_id = pDeptID;
pSuccess := SQL%ROWCOUNT;
ELSEIF v_find = 0 THEN
INSERT INTO departments VALUES (pDeptID, pDeptName, pManID, pLocID);
pSuccess := SQL%ROWCOUNT;
END IF;
STANDARD.COMMIT;
END;
```

5. Untuk mencoba menjalankan Procedure, buatlah sebuah Form sebagai berikut:



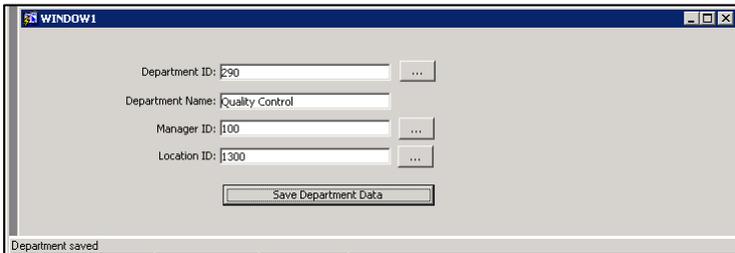
Gambar 7-4 Program Unit Test Form Sample

6. Pada tombol [Save Department Data], tambahkan *trigger* WHEN BUTTON PRESSED, masukkan susunan kode berikut.

```
DECLARE
  v_rsl NUMBER;
BEGIN
  saveDept (:TXTDEPTID, :TXTDEPTNAME, :TXTMANID, :TXTLOCID, v_rsl);
  IF v_rsl > 0 THEN
    MESSAGE ('Department saved');
  ELSE
    MESSAGE ('Department failed to be saved');
  END IF;
END;
```

7. Untuk mencoba form ini, masukkan data department baru berikut:
  - [Department ID]: 290
  - [Department Name] : Quality Control
  - [Manager ID] : 100
  - [Location ID] : 1300

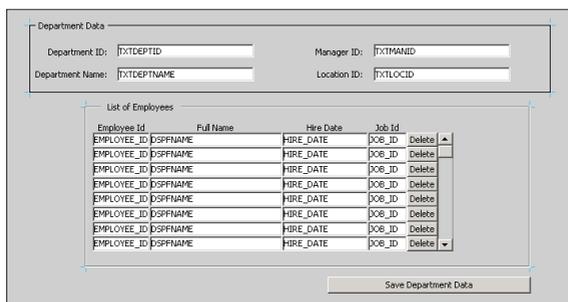
Akan muncul pesan “department saved” setelah tombol [Save Department Data] ditekan.



Gambar 7-5 Hasil Menjalankan *Program Unit*

## Manipulating Data Inside Detail Block

Berikut akan dijelaskan contoh sederhana tentang manipulasi data pada *multiple block (Master Detail Block)*. Dengan memanfaatkan form yang sudah dibuat sebelumnya, tambahkan pada form sehingga menjadi seperti pada gambar 7-6.



Gambar 7-6 *Multiple Block Manipulation Sample*

Ketentuan dari *form* ini adalah:

- ✓ Jika user menekan [Enter] pada [TXTDEPTID], maka form akan menampilkan department\_name, manager\_id, dan location\_id pada item yang sesuai, lalu menampilkan daftar karyawan (employee\_id, nama lengkap, hire\_date, dan job\_id) berdasarkan department\_id yang dimasukkan pada [TXTDEPTID].
- ✓ Jika user menekan [Enter] pada [EMPLOYEE\_ID], maka form akan menampilkan nama lengkap, hire\_date, dan job\_id berdasarkan employee\_id yang dimasukkan pada [EMPLOYEE\_ID]
- ✓ Jika user menekan tombol [Delete], maka update data karyawan dengan menghilangkan department\_id dari karyawan tersebut, lalu hapus karyawan dari tampilan.
- ✓ Jika tombol [Save Department Data] ditekan, akan muncul 2 proses:
  - Cek, apakah data department yang dimasukkan sudah ada atau belum. Jika sudah ada, maka akan dilakukan update data department, jika belum, maka akan dilakukan insert data department baru.
  - setelah mengecek data department, akan dilakukan pengecekan untuk data karyawan pada department.

untuk setiap karyawan yang ditampilkan, lakukan update untuk mengubah `department_id`, `job id`, dan `hire_date`(diubah menjadi hari ini) dari karyawan.

- Setelah melakukan update, lakukan commit untuk menyimpan permanen semua perubahan.

Untuk menyelesaikan kebutuhan ini:

- Tambahkan trigger KEY-NEXT-ITEM pada [TXTDEPTID], lalu tambahkan susunan kode berikut:

```
DECLARE
  v_find NUMBER;
  CURSOR cdeptData IS SELECT department_name, manager_id, location_id FROM departments
                      WHERE department_id = :TXTDEPTID;
  deptDataRec cdeptData%ROWTYPE;
BEGIN
  SELECT COUNT(department_id) INTO v_find FROM departments WHERE department_id = :TXTDEPTID;
  IF v_find > 0 THEN
    OPEN cdeptData;
    Fetch cdeptData INTO deptDataRec;
    :TXTDEPTNAME := deptDataRec.department_name;
    :TXTMANID := deptDataRec.manager_id;
    :TXTLOCID := deptDataRec.location_id;
    CLOSE cdeptData;
  ELSE
    MESSAGE ('Department ID Not Found');
  END IF;
  GO_BLOCK('EMPLOYEES');
  CLEAR_BLOCK(NO_VALIDATE);
  DECLARE
    CURSOR cListEmp IS SELECT employee_id, first_name||' '||last_name as full_name, hire_date, job_id FROM employees
                      WHERE department_id = :TXTDEPTID;
  listEmpRec cListEmp%ROWTYPE;
BEGIN
  OPEN cListEmp;
  LOOP
    FETCH cListEmp INTO listEmpRec;
    EXIT WHEN cListEmp%NOTFOUND;
    :EMPLOYEE_ID := listEmpRec.employee_id;
    :EMPLOYEES_DSPNAME := listEmpRec.full_name;
    :HIRE_DATE := listEmpRec.hire_date;
    :JOB_ID := listEmpRec.job_id;
    NEXT_RECORD;
  END LOOP;
END;
```

Pada susunan kode ini, digunakan 2 buah *cursor*, pertama digunakan untuk mengambil data department, lalu *cursor* yang lain, untuk menampilkan data karyawan pada department.

- Pada text item [EMPLOYEE\_ID], tambahkan trigger KEY-NEXT-ITEM, lalu tambahkan susunan kode berikut

```
DECLARE
CURSOR cEmpData IS SELECT first_name||' '||last_name as full_name, hire_date, job_id FROM EMPLOYEES WHERE employee_id = :EMPLOYEE_ID;
empDataRec cEmpData%ROWTYPE;
BEGIN
OPEN cEmpData;
FETCH cEmpData INTO empDataRec;
IF empDataRec.full_name IS NULL THEN
DELETE_RECORD;
ELSE
:EMPLOYEES.DSPFNAME := empDataRec.full_name;
:HIRE_DATE := empDataRec.hire_date;
:JOB_ID := empDataRec.job_id;
END IF;
CLOSE cEmpData;
END;
```

Seperti yang dapat dilihat, susunan kode ini sama dengan *cursor* pertama pada *trigger* KEY-NEXT-ITEM di [TXTDEPTID].

- Pada tombol [Delete], tambahkan trigger WHEN-BUTTON-PRESSED, lalu masukkan susunan kode berikut:

```
BEGIN
UPDATE employees SET department_id = NULL
WHERE employee_id = :EMPLOYEE_ID;
DELETE_RECORD;
END;
```

Pada susunan kode ini, terdapat perintah DELETE\_RECORD untuk menghapus data pada baris yang sedang diproses.

- Pada tombol [Save Department Data], tambahkan trigger WHEN-BUTTON-PRESSED, lalu masukkan susunan kode berikut:

```
DECLARE
  v_rsl NUMBER;
  v_rsl_dt1 NUMBER := 0;
  v_rsl_dt1_ttl NUMBER := 0;
BEGIN
  saveDept(:TXTDEPTID, :TXTDEPTNAME, :TXTMANID, :TXTLOCID, v_rsl);
  IF v_rsl > 0 THEN
    GO_BLOCK('EMPLOYEES');
    BEGIN
      FIRST_RECORD;
      LOOP
        updatedeptandjob(:EMPLOYEE_ID, :JOB_ID, :BLOCK29.TXTDEPTID, v_rsl_dt1);
        v_rsl_dt1_ttl := v_rsl_dt1_ttl + v_rsl_dt1;
        EXIT WHEN :SYSTEM.LAST_RECORD = 'TRUE';
        NEXT_RECORD;
      END LOOP;
    END;
  END IF;
  IF v_rsl > 0 AND v_rsl_dt1_ttl > 0 THEN
    MESSAGE ('Department saved');
    STANDARD.COMMIT;
  ELSE
    MESSAGE ('Department failed to be saved');
  END IF;
END;
```

Pada susunan kode ini, untuk menyimpan data department, digunakan sebuah procedure yang bernama SAVEDEPT, sedangkan untuk men-*update* data karyawan, digunakan sebuah procedure bernama UPDATEDEPTANDJOB. Lalu untuk melakukan pengecekan untuk setiap *record* pada *block* dengan tipe *tabular*, digunakan sebuah LOOP yang diakhiri dengan kondisi [:SYSTEM.LAST\_RECORD = TRUE] yang berarti, jika record yang sudah diproses adalah record terakhir, maka setelah proses selesai, LOOP akan berhenti. Berikut adalah susunan kode untuk *procedure* SAVEDEPT.

```
PROCEDURE SaveDept (pDeptID IN NUMBER, pDeptName IN VARCHAR2, pManID IN NUMBER, pLocID IN NUMBER, pSuccess OUT NUMBER) IS
v_find NUMBER :=0;
BEGIN
SELECT COUNT(department_id) INTO v_find FROM departments WHERE department_id = pDeptID;
IF v_find > 0 THEN
UPDATE departments SET department_name = pDeptName, manager_id = pManID, location_id = pLocID
WHERE department_id = pDeptID;
pSuccess := SQL%ROWCOUNT;
ELSEIF v_find = 0 THEN
INSERT INTO departments VALUES (pDeptID, pDeptName, pManID, pLocID);
pSuccess := SQL%ROWCOUNT;
standard.commit;
END IF;
END;
```

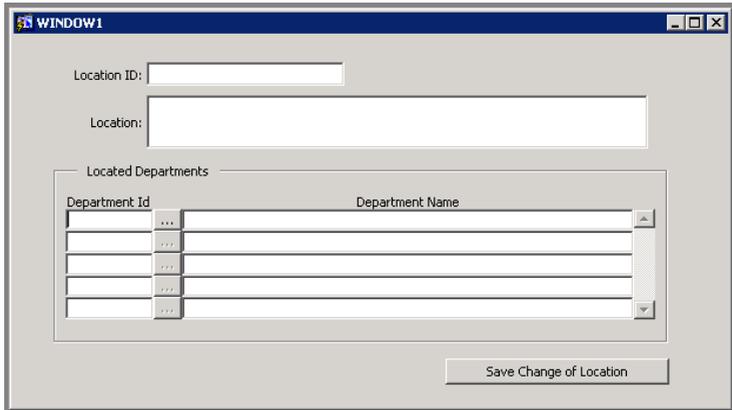
Pada procedure ini, dilakukan pengecekan dulu apakah department\_id sudah ada atau tidak, jika belum ada, maka lakukan INSERT, sedangkan jika sudah ada, lakukan UPDATE. Sedangkan berikut adalah susunan kode untuk procedure UPDATEDEPTANDJOB.

```
PROCEDURE UpdateDeptAndJob (pEmpID IN NUMBER, pJobID IN VARCHAR2, pDeptID IN NUMBER, pRowAff OUT NUMBER) IS
v_find NUMBER;
v_find_job NUMBER;
v_find_rec NUMBER;
BEGIN
SELECT COUNT(employee_id) INTO v_find FROM employees WHERE employee_id = pEmpID;
SELECT COUNT(job_id) INTO v_find_job FROM JOBS WHERE job_id = pJobID;
SELECT COUNT(employee_id) INTO v_find_rec FROM employees WHERE department_id = pDeptID AND job_id = pJobID AND hire_date = SYSDATE;
IF v_find > 0 and v_find_job > 0 and v_find_rec <= 0 THEN
UPDATE employees SET department_id = pDeptID, job_id = pJobID, HIRE_DATE = SYSDATE WHERE employee_id = pEmpID;
END IF;
pRowAff := SQL%ROWCOUNT;
--standard.commit;
END;
```

Pada procedure ini, dilakukan pengecekan apakah job\_id, employee\_id, dan hire\_date benar-benar ada, dan belum ditemukan pada tabel employees (untuk mencegah mengupdate data kembar). Jika kondisi sesuai, maka akan dilakukan UPDATE data karyawan.

## Practice

1. Buat form sebagai berikut



The screenshot shows a Java Swing window titled "WINDOW1" with a standard Mac OS-style title bar. The window contains a form with the following elements:

- A text input field labeled "Location ID:".
- A larger text input field labeled "Location:".
- A section titled "Located Departments" containing a table with two columns: "Department Id" and "Department Name".
- A "Save Change of Location" button at the bottom right.

Department Id	Department Name
...	
...	
...	
...	
...	

2. Buat LOV yang mengembalikan `department_id` dan `department_name`
3. Pada saat form dijalankan
  - 3.1. Jika user menekan Enter pada text item location id, maka sistem akan mengecek apakah location id yang dimasukkan ada atau tidak.
    - Jika ada, tampilkan lokasi-nya dengan format `[street_address], [city] [postal_code]`
    - Jika tidak, tampilkan message "Location ID Not Valid"

- 3.2. Setelah menampilkan lokasi, tampilkan daftar department pada location\_id tersebut
- 3.3. Jika tombol [...] ditekan, panggil LOV yang sudah dibuat pada poin no.2.
- 3.4. Jika tombol [Save Change of Location] ditekan, maka lakukan looping pada block [detail] untuk mengupdate location\_id dari setiap department yang department\_id nya ditampilkan pada block [detail]

## Stage 8: Report and Graphic

### レポートとデプロイメント

“Human walks his/her own path, not path chosen by others.”

人

What you'll learn:

Creating a Report

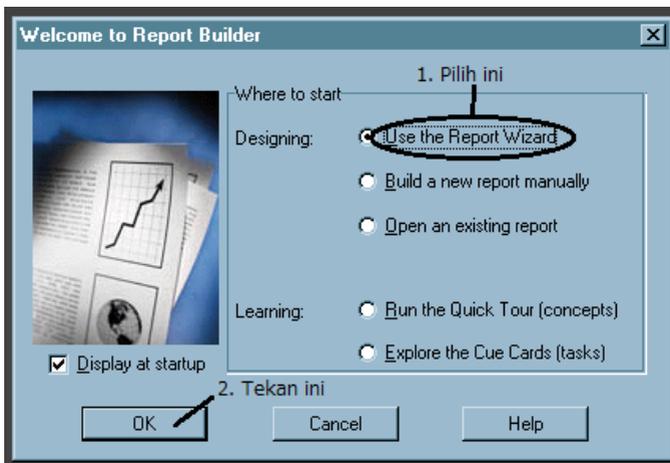
Creating a Graphic

Calling Report or Graphic

## Creating a Report

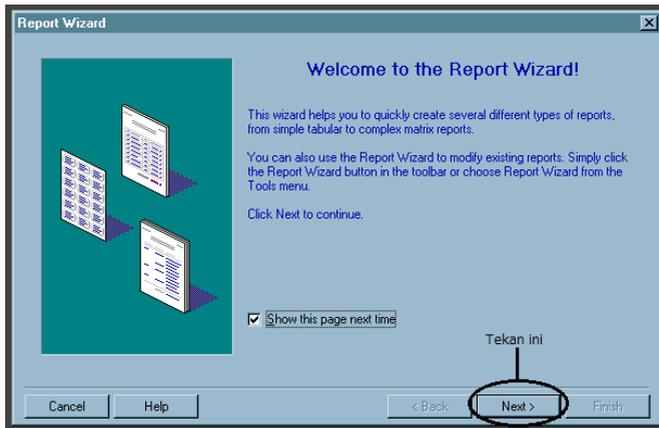
Untuk membuat sebuah laporan yang dapat dicetak, digunakan *tool* bernama *report builder*. Berikut adalah contoh untuk membuat sebuah laporan jumlah karyawan pada sebuah *department* dengan menggunakan *wizard*.

1. Pada saat *software* pertama kali dijalankan, akan muncul sebuah *dialog box* yang menampilkan pilihan, yang menentukan proses berikutnya. Pada poin ini, pilih [Use The Report Wizard], lalu tekan [OK].



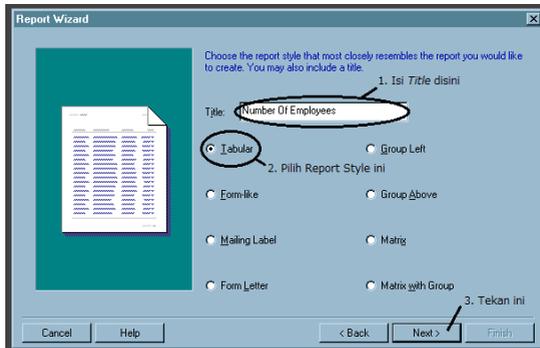
Gambar 8-1 Dialog Box Welcome pada Report Builder

2. Akan muncul *dialog box report wizard*, pada poin ini cukup tekan tombol [Next].



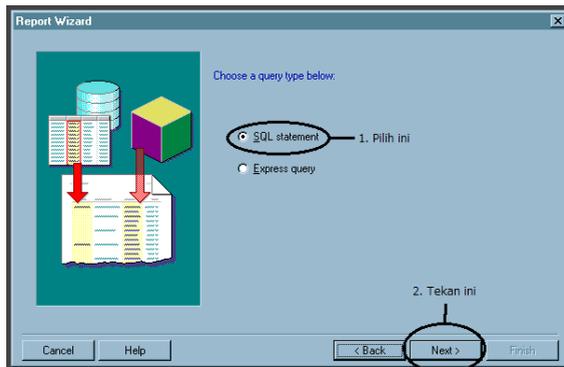
Gambar 8-2 *Dialog Box Report Wizard - Page 1*

3. Pada halaman berikutnya, user diminta untuk memberikan judul dari laporan sekaligus model tampilan dari laporan yang akan dibuat (*report style*). Pada poin ini isi [title] dengan "Number Of Employees", lalu pilih *report style* [tabular], dan terakhir, tekan tombol [Next].



Gambar 8-3 *Dialog Box Report Wizard* – Judul dan Model Laporan

4. Pada poin ini, user diminta untuk memilih jenis *query* yang akan digunakan. Pada poin ini, pilih [SQL Statement], lalu tekan tombol [Next].



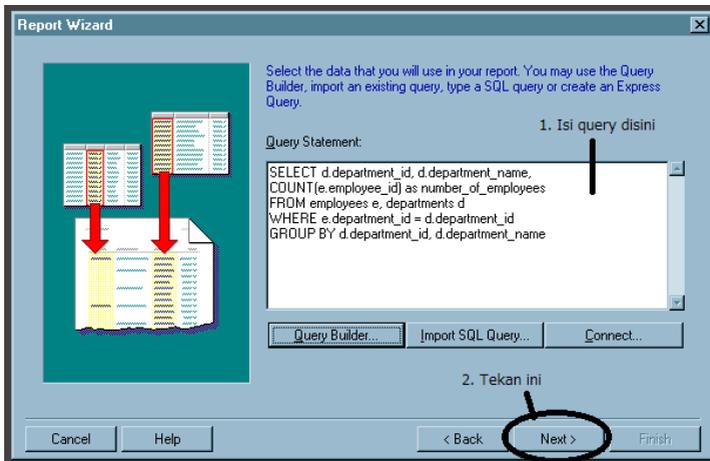
Gambar 8-4 *Dialog Box Report Wizard* - Memilih Jenis Query

5. Pada poin ini, user diminta untuk mengisikan query/perintah SELECT sebagai dasar pengambilan data untuk laporan. *User* dapat memanfaatkan tombol [Query Builder...] sebagai bantuan membangun perintah SELECT. Proses yang dilakukan pada tombol [Query Builder...] sama dengan proses membuat perintah SELECT pada LOV dengan menggunakan tombol [Build SQL Query] (hal. 114). Pada poin ini:

- ✓ Masukkan perintah SELECT berikut pada text item yang sudah ada:

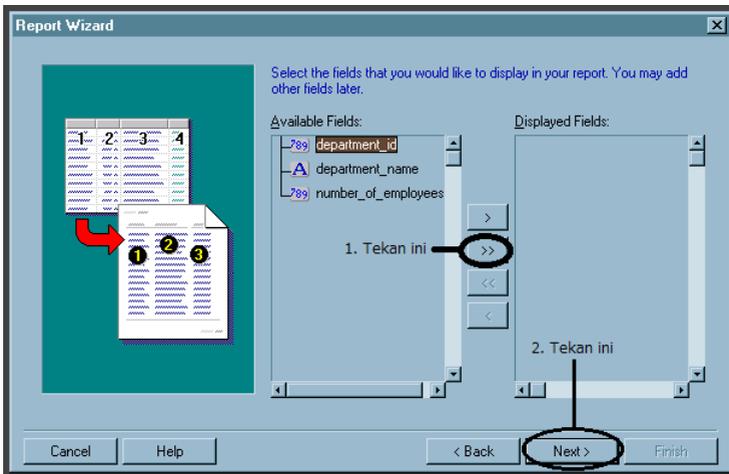
```
SELECT d.department_id, d.department_name,  
COUNT(e.employee_id) as number_of_employees  
FROM employees e, departments d  
WHERE e.department_id = d.department_id  
GROUP BY d.department_id, d.department_name
```

- ✓ Tekan tombol [Next]



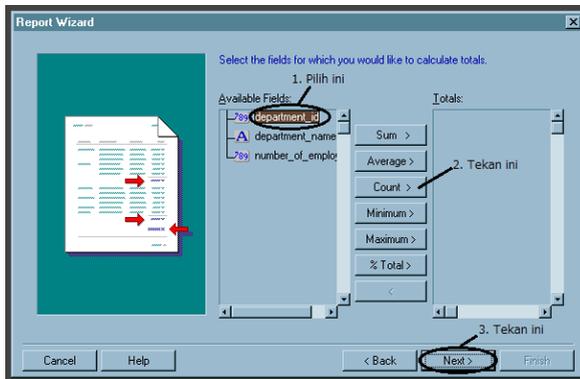
Gambar 8-5 Dialog Box Report Wizard - Mengisi Perintah SELECT

6. Pada poin ini, *user* diminta untuk memilih *field* yang akan ditampilkan pada laporan yang akan dibuat. Pada poin ini, masukkan semua field yang ada dengan menggunakan tombol [ $>>$ ], lalu tekan tombol [Next].



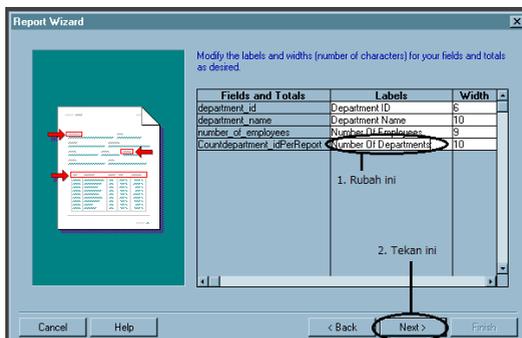
Gambar 8-6 Dialog Box Report Wizard - Memasukkan Semua Field

7. Pada poin ini, user diminta untuk memilih *field* yang akan dilakukan penghitungan (fungsi agregasi). Pada poin ini, pilih *field* [department\_id], tekan tombol [Count], lalu tekan tombol [Next].



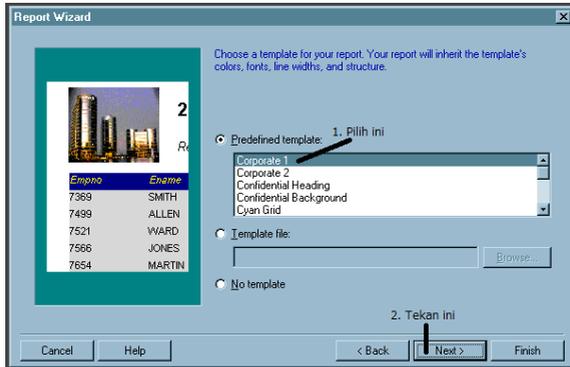
Gambar 8-7 Dialog Box Report Wizard - Memilih Fungsi Penghitungan

- Pada poin ini, user dapat mengatur panjang dari karakter yang dapat ditampilkan, dan mengatur *label* dari setiap nilai yang ditampilkan. pada poin, rubah nilai label pada field hasil COUNT dari poin sebelumnya, menjadi “number of departments”, lalu tekan [Next].



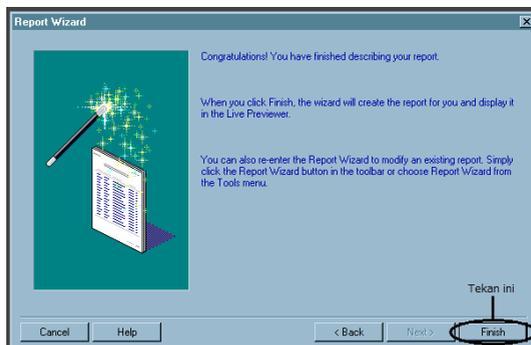
Gambar 8-8 Dialog Box Report Wizard - Setting Field

9. Pada poin ini, user diminta untuk memilih *template* yang akan digunakan sebagai desain laporan. pada poin ini, pilih “corporate 1”, lalu tekan tombol [Next].



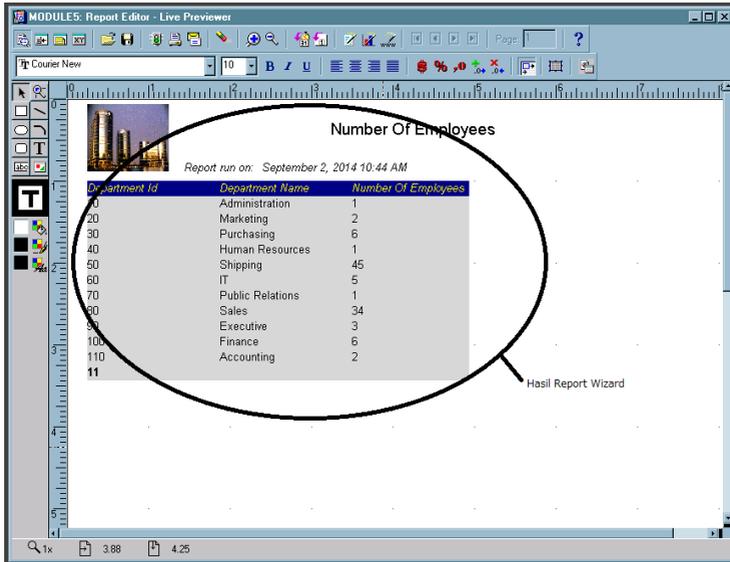
Gambar 8-9 Dialog Box Report Wizard - Memilih Template Laporan

10. Pada poin ini, proses pada wizard sudah selesai, cukup menekan tombol [Finish].



Gambar 8-10 Dialog Box Report Wizard - Finish

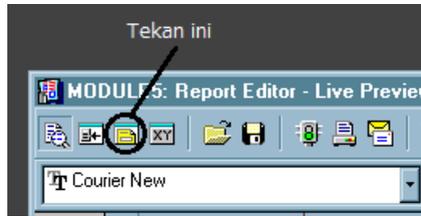
Hasil report wizard akan ditampilkan pada *report editor* bagian *live previewer*.



Gambar 8-11 Hasil *Report Wizard*

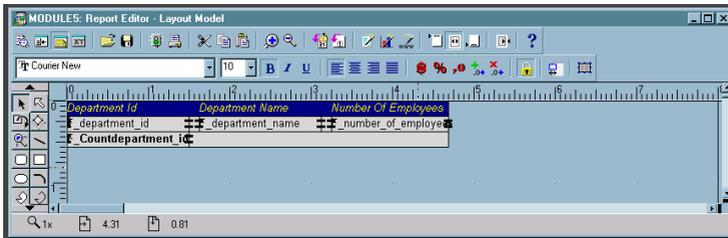
Seperti yang dapat terlihat, label “number of departments yang sudah dibuat pada poin 8 (hal. 159) tidak tampil. Untuk menambahkan label, lakukan *editing* tampilan laporan melalui bagian *layout model* pada *report editor*.

1. Buka *layout model* pada *report editor* dengan menekan icon [layout model] di bagian atas form.



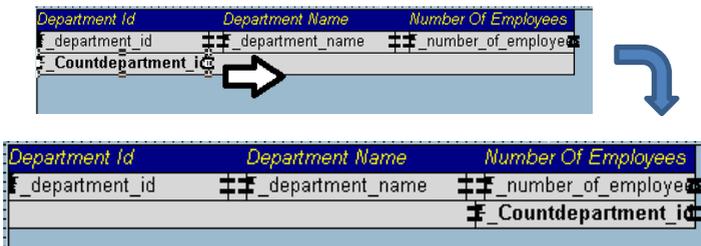
Gambar 8-12 Bagaimana Mengakses *Layout Model* pada *Report Editor*

2. Akan terbuka *Layout Model* sebagai berikut.

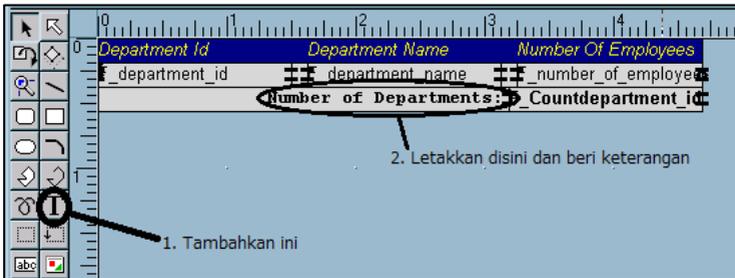


Gambar 8-13 Tampilan *Layout Editor* pada *Report Editor*

Pada poin ini, lakukan *drag* pada field [Countdepartment\_idPerReport] ke kanan.

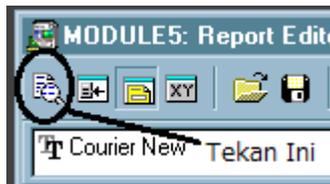


3. Tambahkan sebuah [text] pada posisi field sebelumnya, beri keterangan “Number of Departments:”

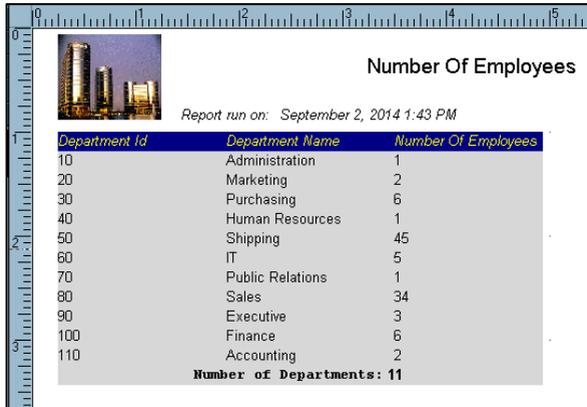


Gambar 8-14 Memasukkan *Text* pada Laporan

Hasil penamabahan [text] dapat dilihat pada bagian *live previewer* dari *report editor*. Untuk menampilkan live previewer di *report editor*, user cukup menekan tombol [live previewer] di sebelah tombol [layout editor].



Gambar 8-15 Bagaimana Mengakses Live Previewer pada Report Editor



Number Of Employees

Report run on: September 2, 2014 1:43 PM

Department Id	Department Name	Number Of Employees
10	Administration	1
20	Marketing	2
30	Purchasing	6
40	Human Resources	1
50	Shipping	45
60	IT	5
70	Public Relations	1
80	Sales	34
90	Executive	3
100	Finance	6
110	Accounting	2

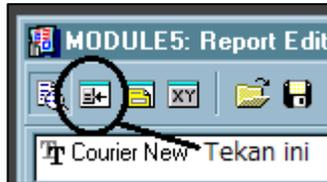
Number of Departments: 11

Gambar 8-16 Laporan Setelah Penambahan [Text]

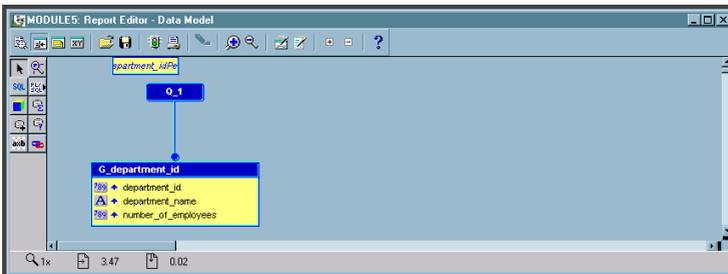
## Creating A Parametered Report

Laporan yang dibuat dengan menggunakan Report Builder juga dapat diatur agar menjadi laporan yang dinamis. Berikut adalah contoh pemanfaatan user parameter untuk membuat tampilan laporan menjadi dinamis, dengan memanfaatkan laporan yang sudah dibuat sebelumnya.

1. Buka bagian *Data Model* pada *report editor* dengan menekan tombol [data model] di sebelah kanan tombol [live previewer].



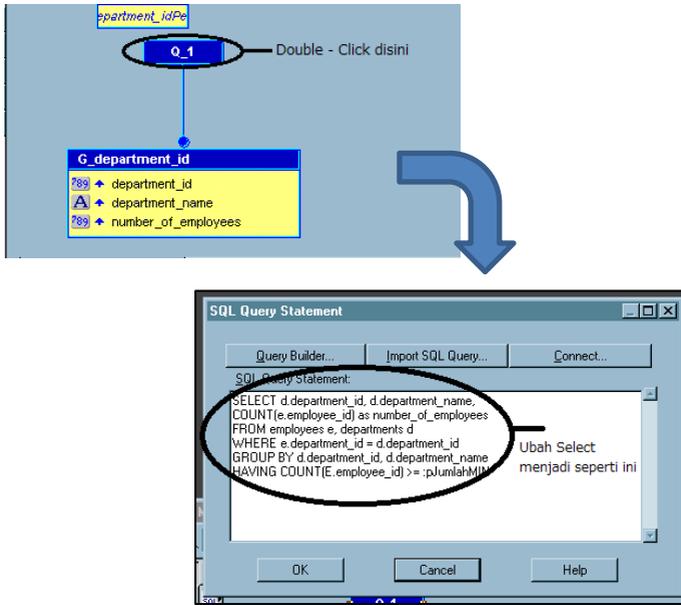
Gambar 8-17 Bagaimana Mengakses Data Model pada Report Editor



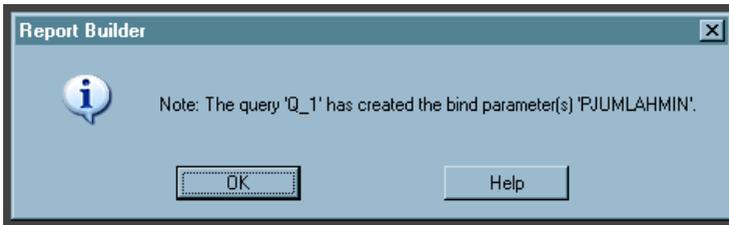
Gambar 8-18 Tampilan *Data Model* pada *Report Editor*

2. Pada bagian data model, lakukan *double-click* pada [Q\_1]. Double click akan membuka sebuah *dialog box* yang berisi perintah SELECT. Ubah perintah SELECT menjadi seperti berikut.

```
SELECT d.department_id, d.department_name,  
COUNT(e.employee_id) as number_of_employees  
FROM employees e, departments d  
WHERE e.department_id = d.department_id  
GROUP BY d.department_id, d.department_name  
HAVING COUNT(E.employee_id) > :pJumlahMIN
```

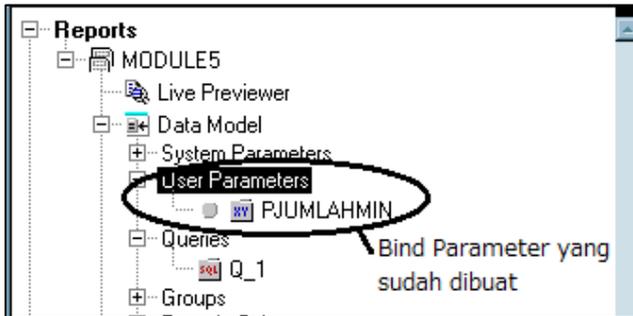


Setelah merubah perintah SELECT, tekan tombol [OK]. Akan muncul pesan, bahwa terdapat *bind parameter* : pJumlahNIM yang terbuat.



Gambar 8-19 Pesan *Bind Parameter* Dibuat

*Bind parameter* yang sudah dibuat dapat dilihat pada bagian [User Parameter] pada *object navigator*.



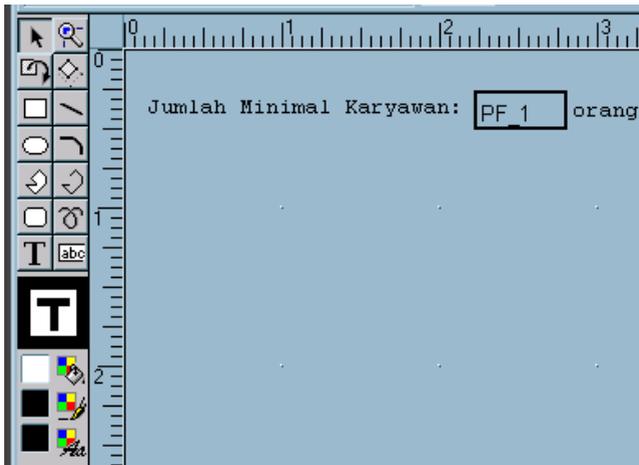
Gambar 8-20 *Bind Parameter* pada *Object Navigator*

3. Setelah membuat *bind parameter*, langkah berikutnya adalah membuat *parameter form* yang akan muncul pada saat laporan. *parameter form* adalah *form* yang digunakan untuk memasukkan *parameter* ke laporan.
  - 3.1. Untuk membuka bagian *parameter form* pada *report editor*, tekan tombol [parameter form] yang ada di sebelah kanan tombol [layout editor].



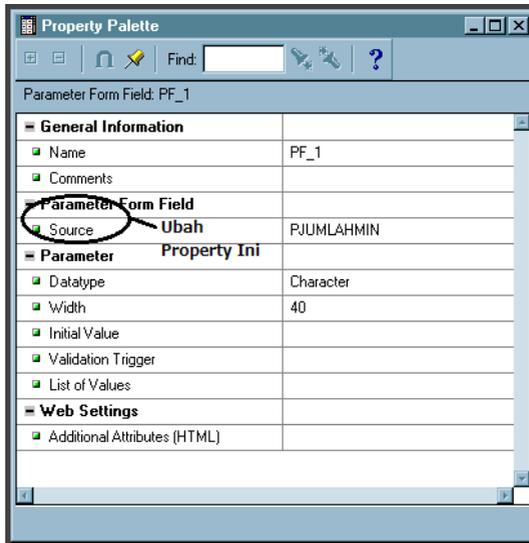
Gambar 8-21 Bagaimana Mengakses *Parameter Form* Pada *Report Editor*

- 3.2. Pada bagian parameter form, buat desain sebagai berikut:



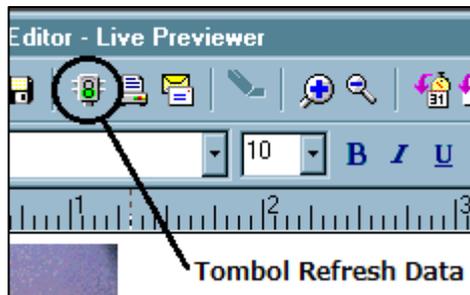
Gambar 8-22 Contoh Desain *Parameter Form*

Tampilan *parameter form* di atas dibuat dengan memanfaatkan [text] dan [field] pada *tool palette*. Setelah melakukan desain, ubah nilai *property* [Source] dari *item* [PF\_1] menjadi *bind parameter* yang sudah dibuat sebelumnya.

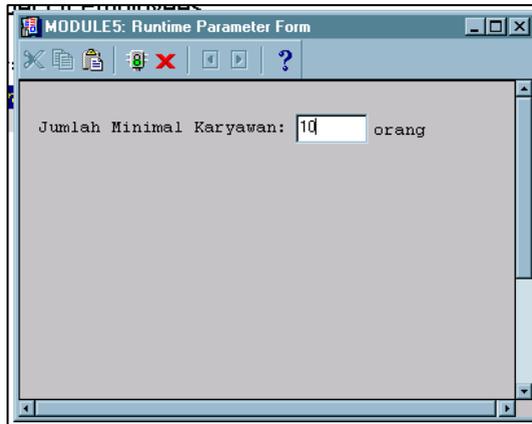
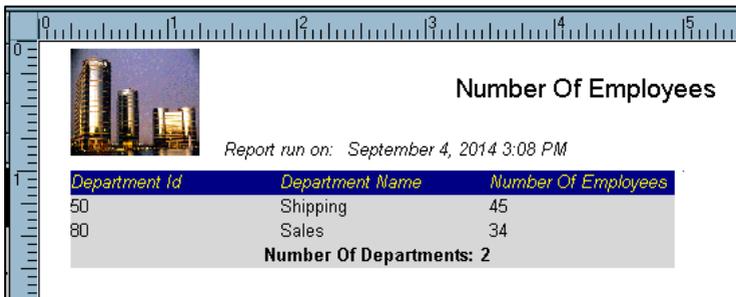


Gambar 8-23 *Property Pallete Parameter Field*

untuk menjalankan laporan, tekan tombol [refresh data]. Untuk mencoba parameter, masukkan nilai 10 pada parameter field, lalu tekan tombol [Enter].



Gambar 8-24 Bagaimana Menjalankan Laporan

Gambar 8-25 *Parameter Form* yang Dijalankan

Department Id	Department Name	Number Of Employees
50	Shipping	45
80	Sales	34

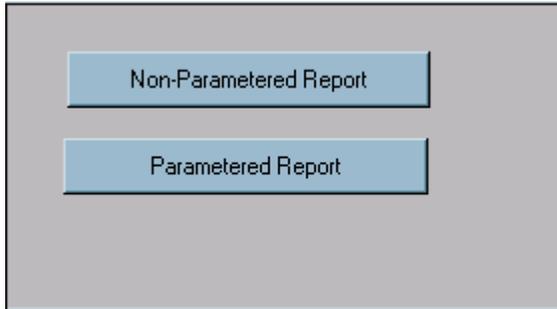
Number Of Departments: 2

Gambar 8-26 Tampilan Laporan Setelah Memasukkan  
*Parameter*

## Call a Report From a Form

Berikut akan diberikan contoh untuk memanggil laporan yang sudah dibuat.

1. Buatlah sebuah form dengan desain sebagai berikut



Gambar 8-27 Contoh Desain Form Untuk Memanggil Laporan

2. Tambahkan Trigger WHEN-BUTTON-PRESSED pada Tombol [Non-Parametered Report] untuk menampilkan laporan yang tidak memiliki *parameter*.

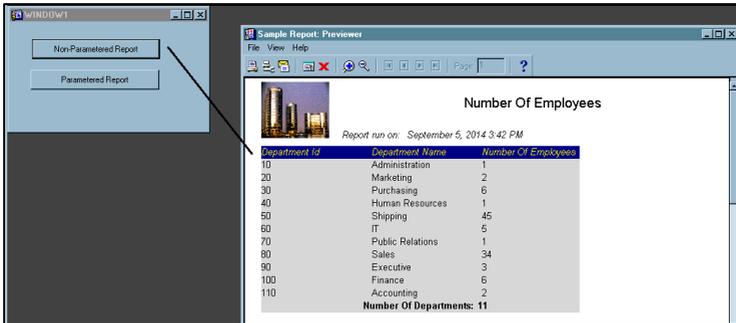
```
DECLARE
  v_path VARCHAR2(50) :=
    'C:\Report\Report_Without_Parameter';
BEGIN
  RUN_PRODUCT(REPORTS, v_path, SYNCHRONOUS, RUNTI
ME, FILESYSTEM, ' ', NULL);
END;
```

Catatan: File laporan yang akan ditampilkan harus diletakkan dan diberi nama yang sesuai dengan variabel `v_path`.

3. Tambahkan trigger WHEN-BUTTON-PRESSED pada tombol [Parametered Report] untuk menampilkan laporan yang memiliki *parameter*.

```
DECLARE
  pl_id PARAMLIST;
  v_path VARCHAR2(50):='C:\Report\Sample Report.RDF';
BEGIN
  pl_id := GET_PARAMETER_LIST('paramlist');
  IF NOT ID_NULL(pl_id) THEN
    DESTROY_PARAMETER_LIST(pl_id);
  END IF;
  pl_id := CREATE_PARAMETER_LIST('paramlist');
  ADD_PARAMETER(pl_id, 'Pjumlahmin', TEXT_PARAMETER, 10);
  ADD_PARAMETER(pl_id, 'PARAMFORM', TEXT_PARAMETER, 'NO');
  RUN_PRODUCT(REPORTS, v_path, SYNCHRONOUS, RUNTIME, FILESYSTEM, pl_id, NULL);
END;
```

Pada susunan kode di atas, dibuat dulu sebuah variabel yang bertipe PARAMLIST untuk menampung semua *parameter* (beserta nilai) yang akan dimasukkan ke parameter laporan, lalu pada saat perintah RUN\_PRODUCT dipanggil, variabel bertipe PARAMLIST dimasukkan sebagai salah satu nilai pada parameter perintah tersebut. Terdapat parameter Pjumlahmin yang namanya disesuaikan dengan nama parameter pada report yang sudah dibuat sebelumnya, dan diisi nilai 10. Terdapat juga parameter bernama PARAMFORM yang diberi nilai "NO", yang berfungsi untuk menghilangkan *runtime parameter form* pada saat laporan dipanggil.



Gambar 8-28 Hasil Memanggil Laporan Dari Form



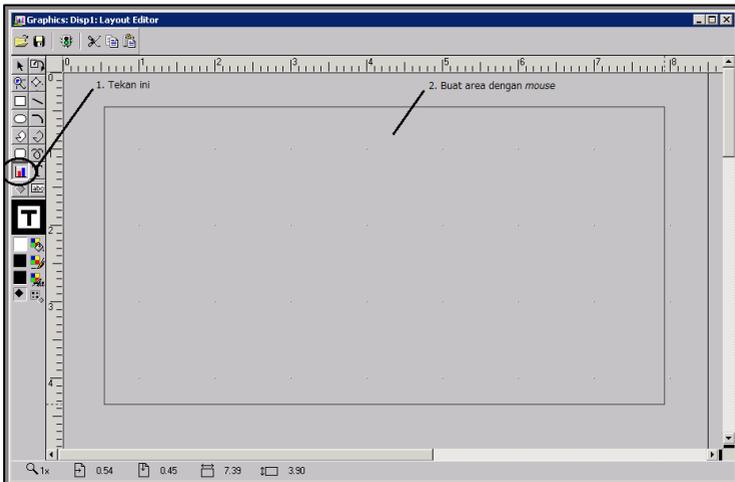
Gambar 8-29 Hasil Memanggil Laporan (Parameter) Dari Form

## Creating A Graphic

Untuk membuat sebuah grafik, digunakan *tool* bernama *graphic builder*. Berikut adalah contoh untuk membuat sebuah grafik jumlah karyawan pada sebuah *department*.

1. Dari tampilan awal *graphic builder*, pilih item [chart] pada *tool pallete* di *layout editor*. Setelah itu, lakukan *dragging*

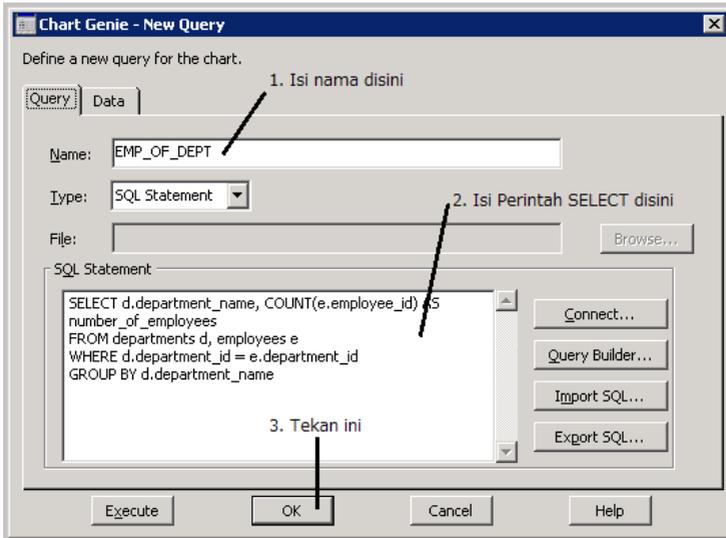
pada *layout editor* untuk memilih area yang digunakan untuk menampilkan grafik.



Gambar 8-30 Memilih Area Grafik

2. Setelah melakukan *dragging*, akan muncul dialog box chart genie. Dialog box ini terdiri atas 2 *tab*, yaitu *tab query* dan *tab data*. *Tab query* digunakan untuk memasukkan perintah SELECT untuk mengambil data dari graphic yang akan ditampilkan, sedangkan *tab data* digunakan untuk menampilkan hasil dari perintah SELECT yang dimasukkan pada *tab query* (dengan menekan tombol [Execute]). Pada poin ini, masukkan perintah SELECT yang sama dengan perintah SELECT pada saat

membuat laporan dengan *report builder* pada *tab query*, beri nama “EMP\_OF\_DEPT”, dan tekan [OK].



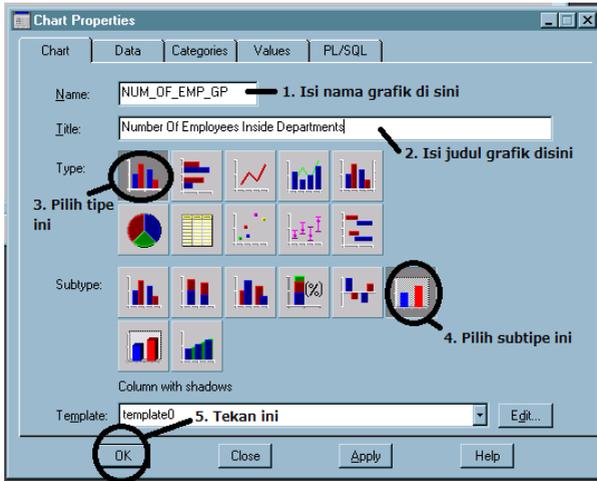
Gambar 8-31 Memasukkan Perintah SELECT

3. Setelah menekan tombol [OK] pada *dialog box chart genie*, akan muncul *dialog box chart properties*. Dialog box ini terdiri atas beberapa *tab*, yaitu *tab chart*, *tab data*, *tab categories*, *tab values*, dan *tab PL/SQL*. Pada *tab chart*, lakukan pengisian sebagai berikut:

- ✓ [Name] : “NUM\_OF\_EMP\_GP”
- ✓ [Title] : “Number of Employees Inside Department”
- ✓ [Type] : Column

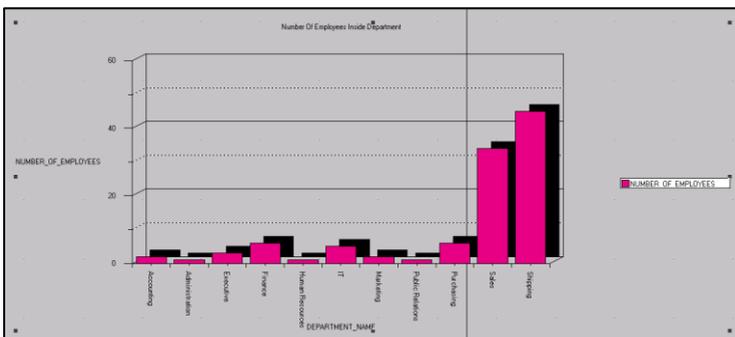
✓ [Subtype] : Column With Shadow

Tab lain tidak perlu dirubah nilainya.



Gambar 8-32 Dialog Box Chart Properties

Grafik yang dihasilkan akan ditampilkan pada *layout editor*.



Gambar 8-33 Hasil Grafik yang sudah dibuat

Untuk memanggil Graphic yang sudah dibuat, gunakan perintah.

```
RUN_PRODUCT(GRAPHICS, '[Lokasi Fisik File]',  
SYNCHRONOUS, RUNTIME, FILESYSTEM, '', NULL);
```

Contoh:

```
DECLARE  
  v_path VARCHAR2(50) := 'C:\\Report\\Sample  
Graphic Pertemuan 8.ogd';  
BEGIN  
  RUN_PRODUCT(GRAPHICS, v_path, SYNCHRONOUS,  
RUNTIME, FILESYSTEM, '', NULL);  
END;
```

## Practice

1. Buatlah Sebuah Laporan yang menampilkan daftar rata-rata gaji dari setiap department
2. .Buatlah sebuah Grafik yang menampilkan jumlah department pada sebuah negara.

## Bibliography

- Oracle. 2004. *Oracle Database 10g: Develop PL/SQL Program Units Volume 1*. Redwood Shores, Amerika Serikat: Oracle Corporation.
- Oracle. 2004. *Oracle Database 10g: PL/SQL Fundamentals Volume 1*. Redwood Shores, Amerika Serikat: Oracle Corporation.
- X-Oerang Technology. 2003. *Pemrograman Menggunakan Oracle Developer*. Yogyakarta: Penerbit Andi.
- X-Oerang Technology. 2004. *Pemrograman Menggunakan Oracle Developer Tingkat Lanjut*. Yogyakarta: Penerbit Andi.

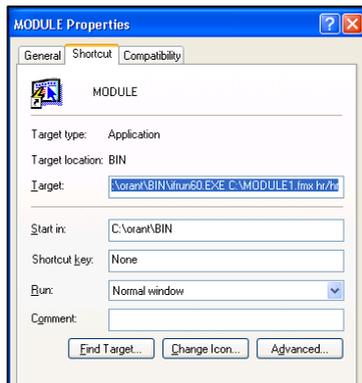
## Bonus Track

Untuk membantu dalam menjalankan sebuah *form* yang sudah di-*compile*, dapat dibuat sebuah *shortcut* pada *desktop* agar user tidak perlu mengakses *form* yang sebenarnya. Berikut adalah langkah - langkah untuk membuat *shortcut*.

1. Buatlah sebuah *shortcut* pada *desktop* dari file *.fmx* yang sudah dihasilkan pada saat *compile*.
2. Lakukan pengaturan *property* dari shortcut sebagai berikut.

Target: [direktori dari orant]\BIN\ifrun60.EXE [direktori dari file *.fmx*] [*user/password* untuk *log in* ke Oracle]

Start In: [direktori dari folder BIN milik orant].



Gambar xxxiv. Contoh Pengaturan *Property Shortcut*