



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 1 PENDAHULUAN

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

01

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang dasar-dasar dari Oracle

Kompetensi

Mampu memahami konsep dasar-dasar dari Oracle

Berinteraksi dengan database

1. Jalankan SQL*Plus dan login ke database menggunakan user SYSTEM.

Jalankan CommandPrompt.

```
C:\>sqlplus
```

```
SQL*Plus: Release 10.2.0.1.0 - Production on Mon Aug 27 13:39:56 2012
```

```
Copyright (c) 1982, 2005, Oracle. All rights reserved.
```

```
Enter user-name: system
```

```
Enter password: *****
```

```
Connected to:
```

```
Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production
```

```
SQL>
```

2. Lakukan Pengecekan terhadap nama database Anda.

```
SQL> SELECT * FROM GLOBAL_NAME;
```

3. Lakukan pengecekan, ada berapa user atau Schema di dalam database Anda.

```
SQL> SELECT USERNAME FROM DBA_USERS;
```

4. Untuk mengecek User atau Schema yang sedang aktif.

```
SQL> Show user;
```

5. Membuat user

```
SQL>CREATE USER Nama_User IDENTIFIED BY password  
DEFAULT TABLESPACE USERS  
QUOTA UNLIMITED ON USERS;  
(bisa diatur sesuai keinginan kita → 50M)
```

6. Memberi GRANT ke user (agar kita bisa masuk ke user)

```
GRANT connect, resource to Nama_User;
```

7. Menghapus User

```
Drop user Nama_User;
```

8. Membuka user yang terkunci(Unlock)

```
SQL>alter user scott account unlock;
```

9. Mengganti password user

```
SQL> alter user nama_user  
2 identified by password_user;
```

10. Pindah User atau Schema

```
SQL> Connect  
Enter User Name :  
Enter Password :  
Atau bisa juga dengan  
SQL>Connect nama_user/password_user
```

11. Melakukan pengecekan di Schema ada objek database apa saja menggunakan perintah

```
SQL> Select object_name, object_type  
2 from user_objects;
```

12. Mengakhiri Koneksi dari database

```
SQL>Disconnect  
Keluar dari SQL*Plus  
SQL>Exit
```

13. Jika ingin mematikan layanan database dari SERVICE OracleXE 10g, ketikan perintah berikut :

```
SQL>conn / as sysdba  
SQL> shutdown immediate
```

Untuk menghidupkannya kembali ketikan perintah berikut :

```
SQL>conn / as sysdba  
SQL> startup
```

14. Menampilkan informasi lengkap tentang status *database*.

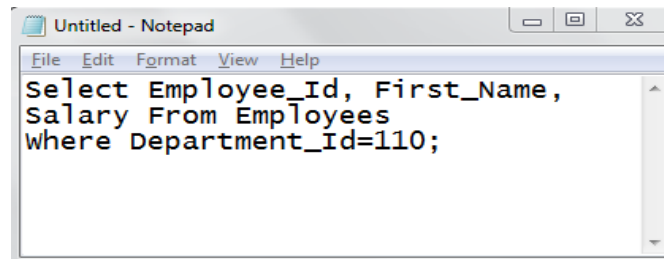
```
SQL> DESC v$database
```

15. Melihat semua *background process* yang dijalankan oleh oracle sebagai bagian dari arsitektur.

```
SQL> show sga;
```

Berinteraksi dengan database

Perintah SQL selain ditulis langsung pada SQL*Plus prompt juga bisa disimpan sebagai file script. Sebagai contoh buatlah file script SQL dengan tools editor seperti notepad dengan isi seperti berikut.



```
Select Employee_Id, First_Name,
Salary From Employees
Where Department_Id=110;
```



File name: **SCRIPT.SQL**

Save as type: **Text Documents (*.txt)**

Selanjutnya jalankan script.sql dari prompt SQL dengan menyebutkan secara lengkap folder tempat file tersebut dan tambahkan tanda @('at') di depannya.


SQL> @C:\SCRIPT.sql

EMPLOYEE_ID	FIRST_NAME	SALARY
-------------	------------	--------

205	Shelley	12000
206	William	8300

Menjalankan Oracle Database dari Web Browser

1. Jalankan Web browser Anda, lalu ketikkan alamat <http://127.0.0.1:8080/apex/>



ORACLE Database Express Edition

Database Login

Enter your database username and password.

Username:

Password:

[Click here to learn how to get started](#)

Links

- o [License Agreement](#)
- o [Documentation](#)
- o [Forum Registration](#)
- o [Discussion Forum](#)
- o [Product Page](#)

2. Login dengan user system.

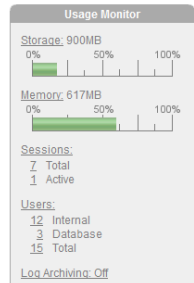
User: SYSTEM

Home

Customize



- Links
- o License Agreement
 - o Getting Started
 - o Learn more
 - o Documentation
 - o Forum Registration
 - o Discussion Forum
 - o Product Page



3. Untuk mengetahui lingkungan database Anda, silakan masuk ke bagian :

Home>Administration>About Database.

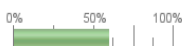
4. Untuk mengetahui konfigurasi memori yang digunakan Oracle, yaitu SGA (*System Global Area*) dan PGA (*Program Global Area*) dengan cara masuk ke bagian :

Home>Administration>Memory

User: SYSTEM

Home > Administration > Memory

	Target	Current
Maximum System Global Area (SGA) Size:	572 MB	572 MB
Program Global Area (PGA) Aggregate Target:	190 MB	43 MB
Current Configuration: (SGA + PGA):	762 MB	615 MB



Memory

Oracle Database Express Edition is limited to a memory size of 1 GB. The sum of Maximum SGA Size and PGA Aggregate Target cannot be larger than 1 GB.

Tasks

- o Configure SGA
- o Configure PGA

5. Melihat kinerja dari *database* dengan cara melihat statistic terhadap kondisi *database* OracleXE Anda saat ini. Silakan masuk ke bagian : **Home>Administration>Database Monitor>System Statistics.**

User: SYSTEM

Home > Administration > Database Monitor > System Statistics

Customize

Refresh Report

Save Statistics

Show delta between current and saved values ☐

Physical I/O Statistics

physical reads 6,324
physical writes 28

Logical I/O Statistics

session logical reads 89,460
db block changes 4,132
redo size 850,296

Memory Statistics

session uga memory 12,893,232,836
session pga memory 32,139,936
workarea executions - optimal 2,604
workarea executions - onepass 0
workarea executions - multipass 0

Time Statistics (Seconds)

CPU used by this session 12.84
DB time 304.24
cluster wait time 0.00
concurrency wait time 0.17
application wait time 0.00
user I/O wait time 25.06

SQL Cursor Statistics

opened cursors current 109
session cursor cache hits 12,090
parse time elapsed 723
parse count (total) 8,112
parse count (hard) 931
execute count 17,218

Transaction Statistics

user commits 38

Save Statistics

Application Express 2.1.0.00.39

Language: en-us

Copyright © 1999, 2006, Oracle. All rights reserved.

6. Administrasi Akun Pemakai, silakan masuk ke bagian :

Home>Administration>Manage Database Users.

User: SYSTEM

Home > Administration > Manage Database Users

Search Username View Icons Show Database Users Display 15 Go

Create >



ADIT



HR



SALES

1-3

Untuk mempermudah dalam praktek kita menggunakan database yang sudah jadi untuk prose query dll.

Membuat User Baru

```
SQL> create user sales identified by sales
```

```
2 default tablespace users
```

```
3 quota unlimited on users;
```

Mengambil script yang di sediakan

```
SQL> @C:\schema_sales.sql
```

Jika pathnya 2 suku kata, maka gunakan tanda " "

Object Schema sales

Tabel yang dibuat pada schema Sales berjumlah Sembilan table sebagai berikut :

Nama Tabel

BARANG

PEGAWAI

BAGIAN

Keterangan

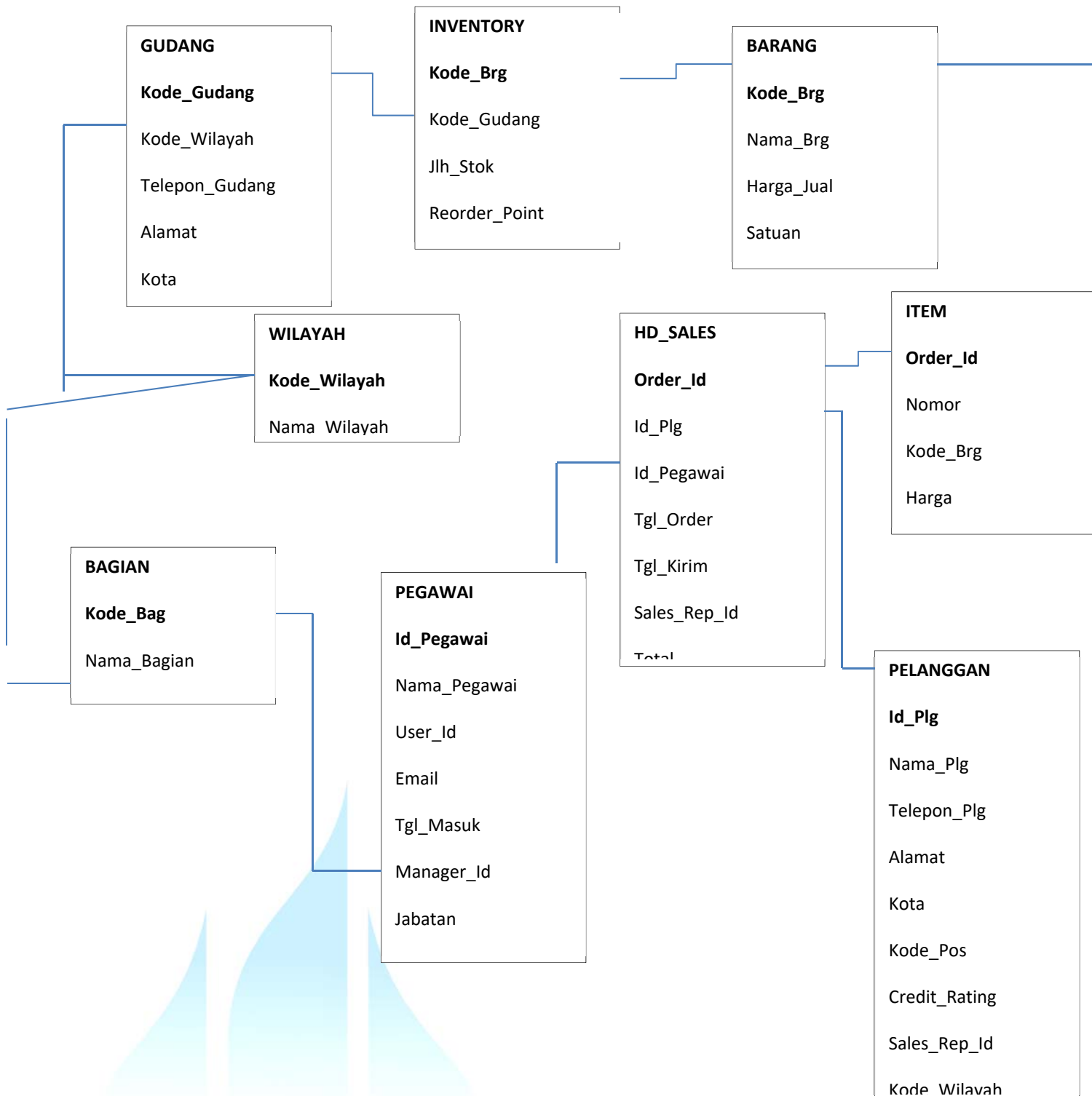
Berisi data barang yang didistribusikan atau dijual

Berisi data pegawai

Berisi data bagian dalam suatu kantor di mana seseorang pegawai ditempatkan

WILAYAH	Berisi data wilayah atau lokasi gudang
PELANGGAN	Berisi data pelanggan
GUDANG	Berisi data gudang tempat menyimpan barang
INVENTORY	Berisi Informasi ketersediaan barang di setiap gudang
HD_SALES	Berisi header data penjualan
ITEM	Berisi detail penjualan dalam satu header

Relasi antara table-table bisa dilihat pada gambar di bawah ini :



Latihan Soal

Buat rancangan suatu database untuk aplikasi yang akan anda jadikan suatu project akhir

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta





MODUL PERKULIAHAN

Pemrograman Sistem Basis Data & Sql

Pertemuan 2

**Membuat Aplikasi untuk Menampilkan Data Tabel
Menggunakan Halaman Berjenis Report**

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

TatapMuka

02

Kode MK

87043

DisusunOleh

Tim Dosen

Abstract

Membahas tentang cara membuat aplikasi untuk menampilkan data table menggunakan halaman berjenis report

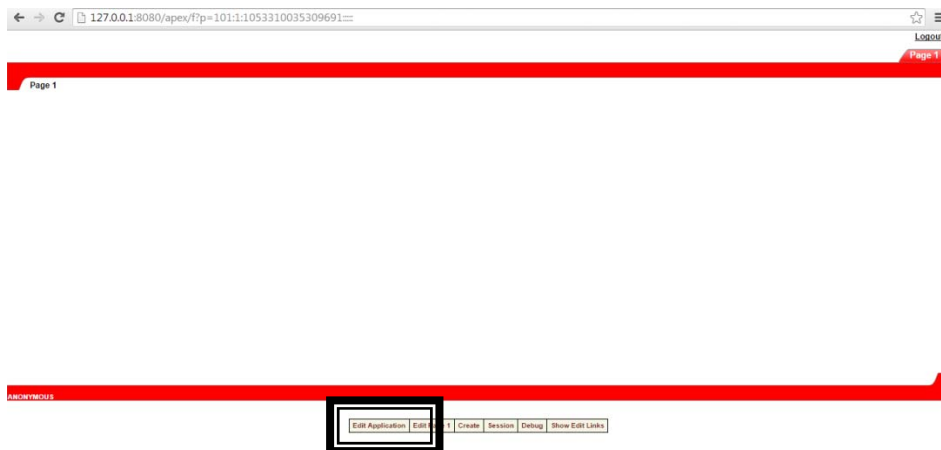
Kompetensi

Mampu memahami cara membuat aplikasi untuk menampilkan data table menggunakan halaman berjenis report

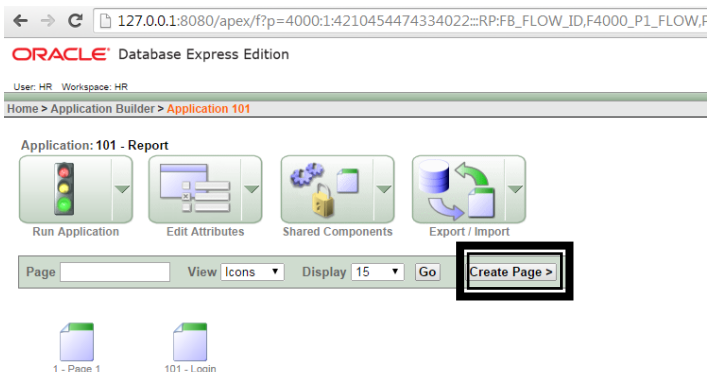
Pendahuluan

Pada modul ini disajikan langkah-langkah yang dibutuhkan untuk membuat aplikasi yang dapat menampilkan data yang terdapat di dalam sebuah tabel. Sebagaimana yang telah diuraikan sebelumnya, membuat aplikasi berbasis Web dalam kerangka Oracle Application Express, sama artinya dengan membangun halaman-halaman Web yang diisi dan dilengkapi dengan berbagai *region*, *item* dan komponen perhitungan, pemrosesan dan pencabangan. Dalam modul ini, kita akan menambahkan halaman yang berjenis “*report*” ke dalam aplikasi yang kita buat. Sebagaimana yang akan disajikan dalam modul ini, Oracle Application Express sudah menyediakan berbagai *template* yang dilengkapi dengan berbagai langkah yang dibutuhkan untuk membuat aplikasi yang dapat menampilkan data dari sebuah tabel dalam database. Untuk membuat aplikasi ini, praktis tidak satu pun baris kode program yang kita ketikkan ke dalam antarmuka Oracle Application Express. Selamat mencoba.

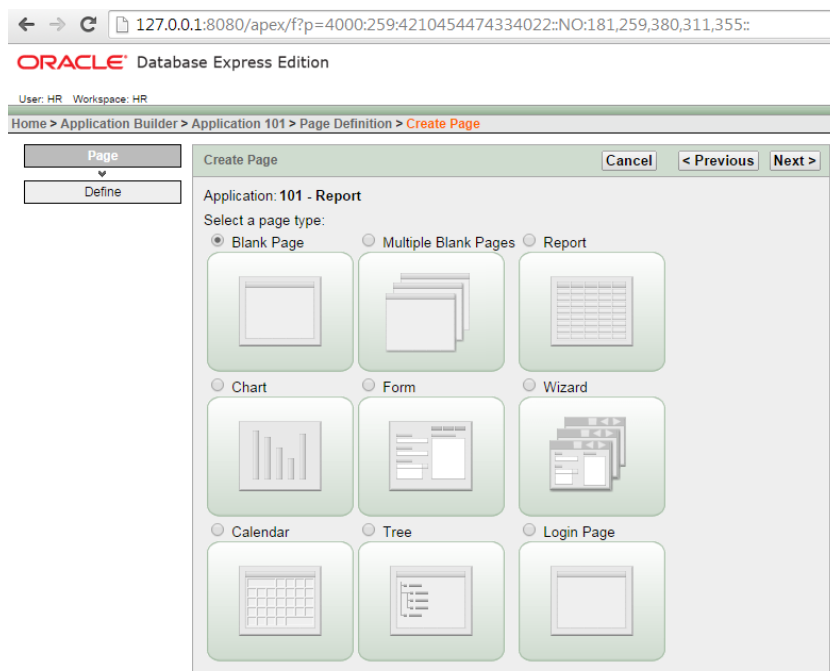
Untuk melakukan editing application dari page yang sudah di buat, klik menu Edit Application



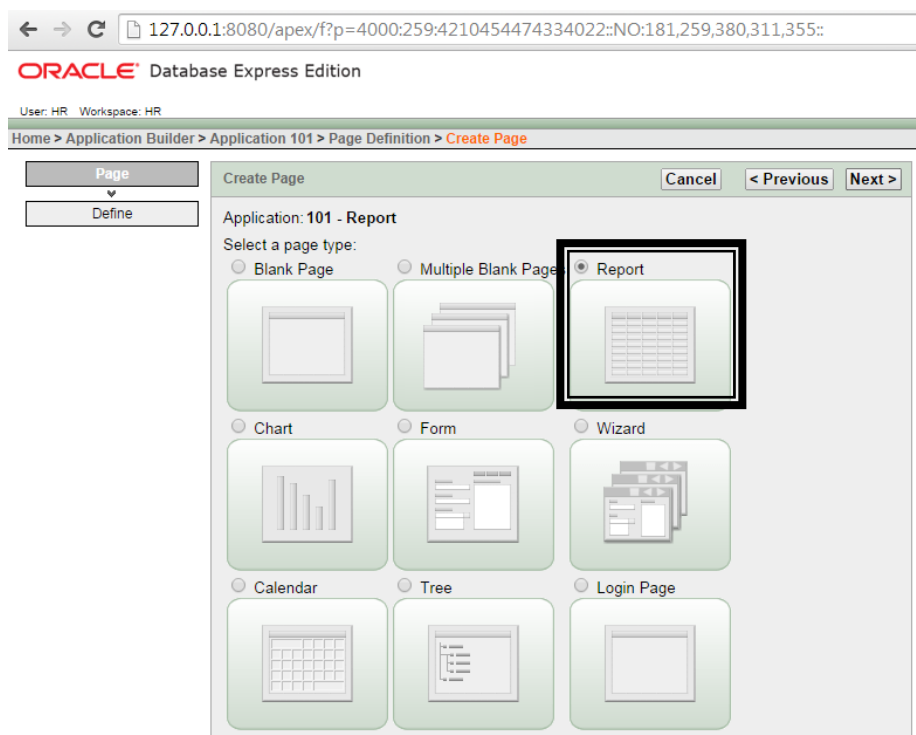
Maka akan muncul menu Application builder seperti gambar di bawah ini, create page untuk melakukan edit page tersebut



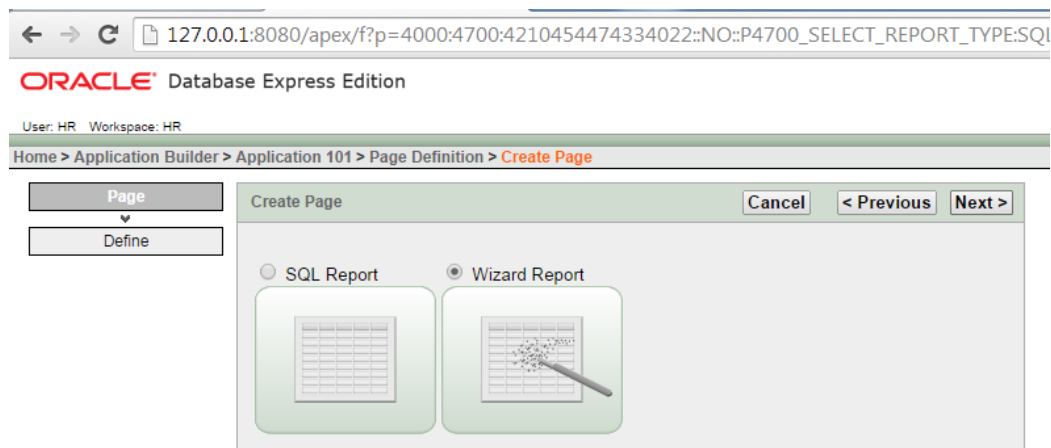
Maka akan ditampilkan menu Select a page type



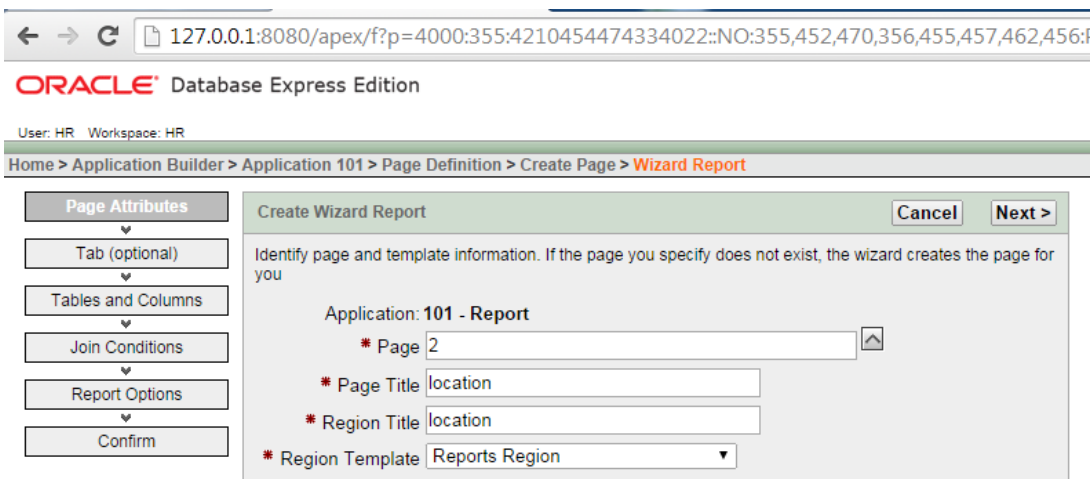
Pada menu select a page type pilih Report lalu Next



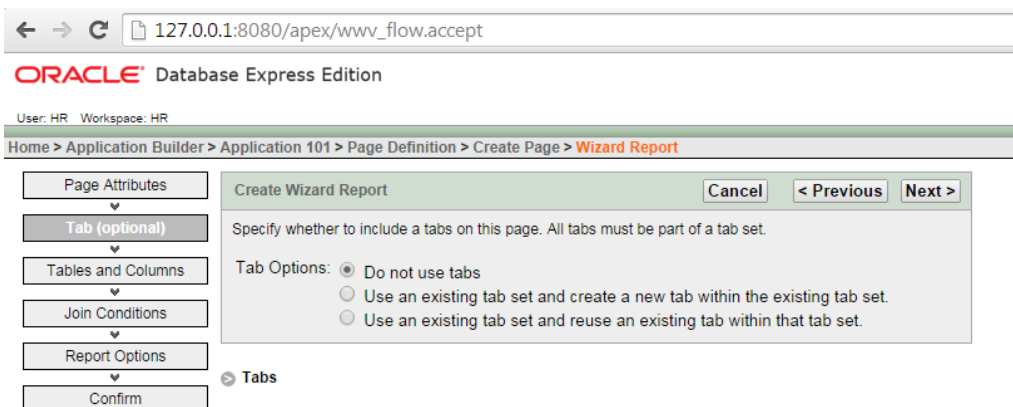
Maka akan muncul pilihan Sql Report dan Wizard Report, pilih Wizard Report lalu klik Next



Maka secara default untuk page akan diisikan 2, karena kita telah membuat 1 buah page di awal. Inputkan Page title dan Region Title (misal : Location) dan region template pilih Reports Region, lalu klik button Next



Maka akan muncul menu create wizard report



Pada menu Tab Option pilih option : Use an Existing tab and create a new within the existing tab set, pada menu Tab set pilih TS1 (page 1)

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 101 > Page Definition > Create Page > Wizard Report

Page Attributes
Tab (optional)
Tables and Columns
Join Conditions
Report Options
Confirm

Create Wizard Report [Cancel] < Previous Next >

Specify whether to include a tabs on this page. All tabs must be part of a tab set.

Tab Options: ☐ Do not use tabs
☒ Use an existing tab set and create a new tab within the existing tab set.
☐ Use an existing tab set and reuse an existing tab within that tab set.

* Tab Set: TS1 (Page 1)
 New Tab Label: - Select Tab Set -

➤ Tabs

Isikan new tab label dengan nama report yang akan di buat (misalnya : Report atas table locations), lalu klik next

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 101 > Page Definition > Create Page > Wizard Report

Page Attributes
Tab (optional)
Tables and Columns
Join Conditions
Report Options
Confirm

Create Wizard Report [Cancel] < Previous Next >

Specify whether to include a tabs on this page. All tabs must be part of a tab set.

Tab Options: ☐ Do not use tabs
☒ Use an existing tab set and create a new tab within the existing tab set.
☐ Use an existing tab set and reuse an existing tab within that tab set.

* Tab Set: TS1 (Page 1)
 New Tab Label: Report atas tabel LOCATIONS

➤ Tabs

Pada menu wizard report, pilih Table / view owner : HR, pilih combo table/view dengan LOCATION

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 101 > Page Definition > Create Page > Wizard Report

Page Attributes
Tab (optional)
Tables and Columns
Join Conditions
Report Options
Confirm

Create Wizard Report [Cancel] < Previous Next >

Table / View Owner: HR
 Table/View: - Select Table -
 Select the columns from:
 Available Columns: COUNTRIES(table), DEPARTMENTS(table), EMPLOYEES(table), EMP_DETAILS_VIEW(view), JOBS(table), JOB_HISTORY(table), LOCATIONS(table), REGIONS(table)

Pada menu tersebut akan ditampilkan isi dari table Locations pada database HR, di menu Available Columns sorot field yang akan kita gunakan, pindahkan ke menu displayed columns dengan cara mengklik button kanan

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 101 > Page Definition > Create Page > Wizard Report

Page Attributes
Tab (optional)
Tables and Columns
Join Conditions
Report Options
Confirm

Create Wizard Report [Cancel] < Previous Next >

Table / View Owner: HR
Table/View: LOCATIONS(table)

Select the columns to display in your report.

Available Columns	Displayed Columns
LOCATION_ID	
STREET_ADDRESS	
POSTAL_CODE	
CITY	
STATE_PROVINCE	
COUNTRY_ID	

Maka isi table Locations sudah dipindahkan ke bagian display columns, lalu klik Next

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 101 > Page Definition > Create Page > Wizard Report

Page Attributes
Tab (optional)
Tables and Columns
Join Conditions
Report Options
Confirm

Create Wizard Report [Cancel] < Previous Next >

Table / View Owner: HR
Table/View: LOCATIONS

Show Only Related Tables: ☒ Yes ☐ No

Select the columns to display in your report.

Available Columns	Displayed Columns
	LOCATIONS.LOCATION_ID
	LOCATIONS.STREET_ADDRESS
	LOCATIONS.POSTAL_CODE
	LOCATIONS.CITY
	LOCATIONS.STATE_PROVINCE
	LOCATIONS.COUNTRY_ID

Maka akan tampil seperti gambar di bawah ini, pada menu di bawah ini kita bisa mengatur jumlah rows per page, isikan nilai dari rows per page (misalnya : 15), lalu klik next

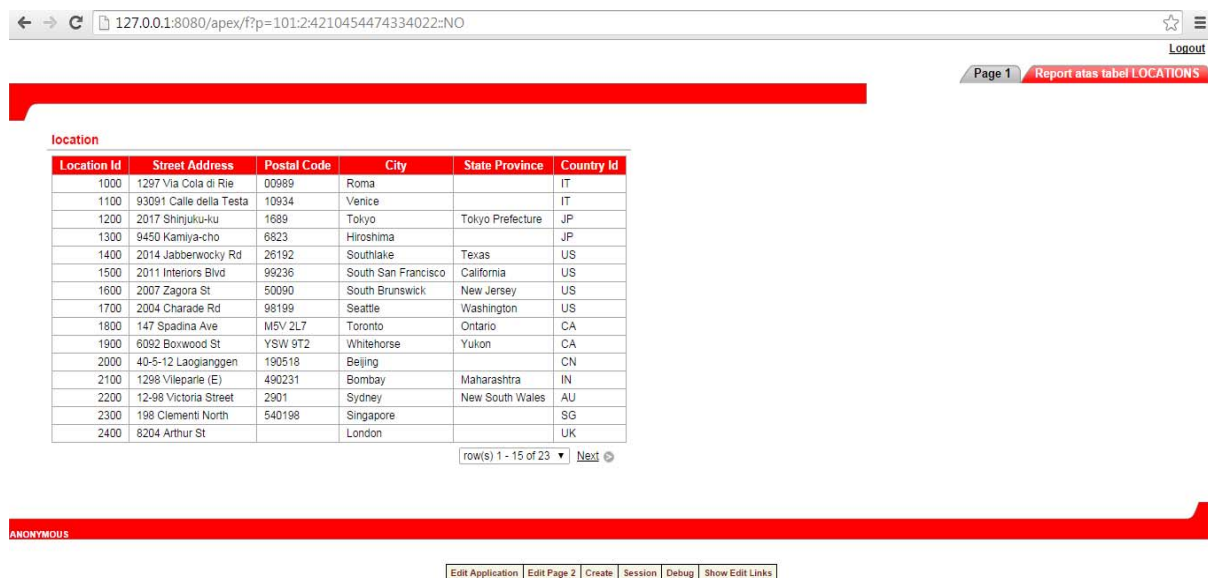
Pada menu Wizard report confirmation, klik button next jika sudah tidak ada lagi yang mau dilakukan perubahan

Page telah sukses di buat, klik Run Page untuk melihat tampilan report yang telah di buat

Maka akan tampil report di bawah ini, pada tampilan report tersebut terdapat 2 buah page (Page 1 dan Page Report Atas Tabel Locations)



Untuk melihat isi report page report atas table location, klik page tersebut



Untuk melakukan editing pada report klik edit page 2

← → ↺ 127.0.0.1:8080/apex/f?p=101:2:4210454474334022:NO

Page 1 Report atas tabel LOCATIONS

location

Location Id	Street Address	Postal Code	City	State Province	Country Id
1000	1297 Via Cola di Rie	00989	Roma		IT
1100	93091 Calle della Testa	10934	Venice		IT
1200	2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
1300	9450 Kamiya-cho	6823	Hiroshima		JP
1400	2014 Jabbenwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1600	2007 Zagora St	50090	South Brunswick	New Jersey	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	147 Spadina Ave	M5V 2L7	Toronto	Ontario	CA
1900	6092 Boxwood St	Y5W 9T2	Whitehorse	Yukon	CA
2000	40-5-12 Laogianggen	190518	Beijing		CN
2100	1298 Vileparle (E)	490231	Bombay	Maharashtra	IN
2200	12-98 Victoria Street	2901	Sydney	New South Wales	AU
2300	198 Clement North	540198	Singapore		SG
2400	8204 Arthur St		London		UK

row(s) 1 - 15 of 23 Next

ANONYMOUS

Edit Application Edit Page 2 Create Session Debug Show Edit Links

Untuk melihat kembali hasil report klik Run Page 2

← → ↺ 127.0.0.1:8080/apex/f?p=4000:4150:4210454474334022:NO:1,4150:FB_FLOW_ID,FB_FLOW_PAGE_ID,F4000_P1_FLOW,F4000_P4150_GOTO_PAGE,F4000_P1_PAGE:101,:

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 101 > Page Definition

Page 2 View Definition Go < >

Delete Copy Edit Attributes Create >

Name: location Last Updated: HR, 3 minutes ago

Page Rendering

- Regions
 - Display Point: Page Template Body (3)
 - RPT: location (10)
- Buttons
- Items
- Computations
- Processes

Page Processing

- Computations
- Validations
- Processes
- Branches

Shared Components

- Tab
 - Tab Set: T01
 - Page :1
 - Report atas tabel LOCATIONS
- Lists of Values
- Breadcrumbs
- Lists
- Theme
 - 1. Red
- Templates
 - Page: One Level Tabs
 - Region: Reports Region
 - Report: Standard
- Security
- Navigation Bar
 - Logout

Language: en-us Application Express 2.1.0.00.39 Copyright © 1996, 2008, Oracle. All rights reserved.

DaftarPustaka

Oracle. 2006. *Oracle 10 g Express Edition Help System*.



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 2 SQL Plus

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

02

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam SQL Plus

Kompetensi

Mampu memahami perintah-perintah dalam SQL Plus

Perintah SQL Plus

1. Melihat Struktur Tabel

Menggunakan Perintah **DESCRIBE** untuk menampilkan struktur table pegawai

```
SQL>DESCRIBE pegawai
```

Penulisan DESCRIBE bisa disingkat dengan DESC

```
SQL>DESC bagian
```

2. Menyimpan dan menjalankan Script

Ketik perintah berikut :

```
SQL> SELECT   id_pegawai,  
2           nama_pegawai,  
3           gaji  
4 FROM pegawai  
5 Where gaji > 10000000;
```

Untuk menampilkan isi Buffer SQL *Plus atau menampilkan perintah yang terakhir diketik, gunakan LIST atau L.

```
SQL> List
```

```
1 SELECT id_pegawai,  
2       nama_pegawai,  
3       gaji  
4 FROM pegawai  
5* Where gaji > 10000000
```

Untuk menjalankan perintah yang ada di buffer ketik RUN atau slash /. Beda Run dengan slash "/", kalau dengan slash "/" perintah tidak ditampilkan, sedangkan dengan RUN perintah dimunculkan dulu baru dieksekusi.

```
SQL>RUN
```

```
SQL>/
```

Untuk menyimpan perintah yang ada di buffer menjadi file script, menggunakan perintah **SAVE**. Misal perintah yang di atas saat ini sedang ada di buffer akan kita simpan menjadi file info_pegawai.sql

```
SQL>SAVE info_pegawai.sql
```

Di folder mana file info_pegawai.sql di simpan? Untuk mengetahui default penyimpanan ketik saja HOST untuk memanggil Ms Dos Prompt.

SQL> host

Microsoft Windows [Version 6.1.7601]

Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\oracle\app\oracle\product\10.2.0\server\BIN>

Berarti File info_pegawai.sql di simpan di

C:\oracle\app\oracle\product\10.2.0\server\BIN>

Ketik **exit** untuk kembali ke SQL *Plus

Jika ingin mengarahkan ke folder tertentu pada perintah SAVE sebutkan nama Foldernya.

SQL> Save C:\SCRIPT\info_pegawai.sql

Perintah GET untuk mengambil file script dan dimuat ke dalam buffer

SQL> GET C:\SCRIPT\info_pegawai.sql

Jika ingin langsung menjalankan file script dari SQL *Plus, gunakan perintah START atau gunakan tanda @.

SQL> START C:\Script\info_pegawai.sql

SQL> @ C:\Script\info_pegawai.sql

Untuk menghilangkan perintah dari buffer atau membersihkan buffer gunakan perintah CLEAR BUFFER.

SQL> CLEAR BUFFER

SQL> list

SP2-0223: No lines in SQL buffer.

3. Editing Perintah SQL

Untuk mempermudah kesalahan ketik dapat menggunakan perintah **EDIT**.

SQL> EDIT

4. Menyimpan Hasil Query ke File

Untuk menyimpan setiap hasil eksekusi intruksi SQL ke dalam file, pertama-tama user harus mengaktifkan terlebih dahulu mode **SPOOL** diikuti parameter nama file yang akan digunakan untuk menyimpan hasil eksekusi tersebut.

```
SQL>SPOOL C:\SCRIPT\info_pegawai.txt
```

```
SQL>SPOOL OFF;
```

5. Mengatur Format Tampilan

- Format Kolom

```
SQL> Column nama_pegawai FORMAT A20 Heading "Nama Pegawai"
```

```
SQL> Column gaji FORMAT 999G999G999
```

```
SQL> Select id_pegawai,
```

```
2      nama_pegawai,
```

```
3      gaji
```

```
4  from Pegawai;
```

ID_PEGAWAI	Nama Pegawai	GAJI
-----	-----	-----
1001	MEUTIA JOVI MAHARANI	22,500,000
1002	BUDI HARTADI	10,500,000
1003	RULLY SIANIPAR	10,000,000
1004	EDWIN ASRUL	10,500,000
1005	NOVI SETIAWATI	11,500,000
1006	ARIS MURSITO	7,000,000
1007	IRVAN SYAFE'I	7,500,000

Untuk menghilangkan format suatu kolom gunakan perintah CLEAR,

```
SQL> CLEAR nama_pegawai
```

```
SQL> CLEAR COLUMNS /untuk menghilangkan semua format column
```

- Setting Halaman

```
SQL>SET PAGESIZE 50
```

```
SQL>SELECT id_pegawai,
```

```
2      nama_pegawai
```

```
3  From pegawai;
```

- Setting Baris

Untuk menentukan panjang suatu baris data digunakan perintah SET LINESIZE.

```
SQL>SET LINESIZE 200
```

```
SQL> SELECT * From pegawai;
```

6. Mengelompokkan Data dengan BREAK ... SKIP

```
SQL> BREAK ON kode_bag
```

```
SQL> SELECT kode_bag, id_pegawai, nama_pegawai, gaji
```

```
2 from pegawai
```

```
3 Order by kode_bag, id_pegawai;
```

KODE_BAG	ID_PEGAWAI	Nama Pegawai	GAJI
-----	-----	-----	-----
10	1004	EDWIN ASRUL	10,500,000
31	1003	RULLY SIANIPAR	10,000,000
	1011	RIDWAN SANUSI	4,000,000
	1012	SUSI INDIARTI	4,900,000
	1013	POPY LUSIANA	5,150,000

Tambahlan BREAK dan SKIP

```
SQL>BREAK ON kode_bag SKIP 1;
```

KODE_BAG	ID_PEGAWAI	Nama Pegawai	GAJI
-----	-----	-----	-----
41	1025	ASFIANTI	5,000,000
50	1001	MEUTIA JOVI MAHARANI	22,500,000
	1005	NOVI SETIAWATI	11,500,000
	1021	SUHARDIATNO	2,500,000
	1022	AHMAD KOSASIH	2,000,000

7. Header dan Footer

```
SQL> SET PAGESIZE 15
```

```
SQL> SET LINESIZE 60
```

```
SQL> TTITLE LEFT 'Judul Kiri Atas'
```

```
SQL> BTITLE RIGHT 'Judul Kanan Bawah'
```

```
SQL> Select kode_bag, id_pegawai, nama_pegawai, gaji
```

```

2 FROM pegawai
3 Where kode_bag IN (31,50)
4 ORDER BY kode_bag, id_pegawai;

```

Judul Kiri Atas

KODE_BAG	ID_PEGAWAI	Nama Pegawai	GAJI
-----	-----	-----	-----
31	1003	RULLY SIANIPAR	10,000,000
	1011	RIDWAN SANUSI	4,000,000
	1012	SUSI INDIARTI	4,900,000
	1013	POPY LUSIANA	5,150,000
	1014	ERTIN	5,250,000
	1015	ESTI ARVINA	4,500,000
50	1001	MEUTIA JOVI MAHARANI	22,500,000
	1005	NOVI SETIAWATI	11,500,000

Judul Kanan Bawah

Untuk menghapus TTITLE dan BTITLE

```
SQL>TTITLE OFF
```

```
SQL>BTITLE OFF
```

```
SQL>CLEAR BREAK
```

8. COMPUTE

Menampilkan jumlah total gaji per kode bagian

```
SQL>BREAK ON kode_bag SKIP 1
```

```
SQL>SET PAGESIZE 100
```

```
SQL>BREAK ON kode_bag SKIP 1
```

```
SQL>COMPUTE sum OF gaji ON kode_bag
```

```
SQL>SELECT kode_bag, id_pegawai, gaji
```

```
2 FROM pegawai
```

```
3 ORDER BY kode_bag, id_pegawai;
```

Menampilkan gaji paling tinggi pada setiap bagian

```
SQL> COMPUTE sum OF gaji ON kode_bag
```

```
SQL> SELECT kode_bag, id_pegawai, gaji
```

```
2 FROM pegawai
```

3 ORDER BY kode_bag, id_pegawai;

Untuk menghapus COMPUTE

SQL>CLEAR COMPUTE

SQL>CLEAR BREAK

Menampilkan nilai agregasi keseluruhan

SQL> BREAK ON REPORT

SQL> COMPUTE SUM OF gaji ON Report

SQL> Select kode_bag, id_pegawai, gaji

2 From pegawai

3 Where kode_bag<=40

4 order by kode_bag;

KODE_BAG	ID_PEGAWAI	GAJI
10	1004	10,500,000
31	1011	4,000,000
31	1015	4,500,000
31	1013	5,150,000
31	1014	5,250,000
31	1003	10,000,000
31	1012	4,900,000
sum		44,300,000

9. Subtitution Variabel

SQL> SELECT id_pegawai, nama_pegawai, kode_bag, gaji

2 FROM pegawai

3 WHERE kode_bag=&bagian;

SQL> SELECT id_pegawai, nama_pegawai, kode_bag, gaji

2 FROM pegawai

3 WHERE nama_pegawai=&nama';

10. Verify

Nilai Verify bisa ON dan OFF, jika ON berarti melakukan verifikasi dan sebaliknya jika OFF tidak melakukan verifikasi. Secara Default VERIFY bernilai ON.

```
SQL>SET VERIFY OFF
```

```
SQL> SELECT id_pegawai, nama_pegawai, kode_bag, gaji
```

```
2 FROM pegawai
```

```
3 WHERE nama_pegawai='&nama';
```

Latihan Soal

1. Perintah yang digunakan dalam kelompok Data Manipulation Language adalah
2. User bisa melakukan akses ke beberapa database, pernyataan tersebut benar atau salah ? Mengapa
3. Untuk berkomunikasi dengan database kita harus menggunakan bahasa yang dipahami oleh database. Bahasa yang digunakan untuk berkomunikasi dengan database disebut ?
4. Sebutkan tools interface yang dapat digunakan untuk berinteraksi dengan database ?
5. Untuk menampilkan daftar user yang sudah terdaftar di dalam suatu database dilakukan dengan perintah ?

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrograman Sistem Basis Data & Sql

Pertemuan 3

**Membuat Aplikasi untuk Menampilkan Data Tabel
Menggunakan Halaman “Blank” yang Dilengkapi
dengan Region**

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

TatapMuka

Kode MK
87043

DisusunOleh
Tim Dosen

03

Abstract

Membahas tentang membuat aplikasi untuk menampilkan data table menggunakan halaman blank yang dilengkapi dengan region

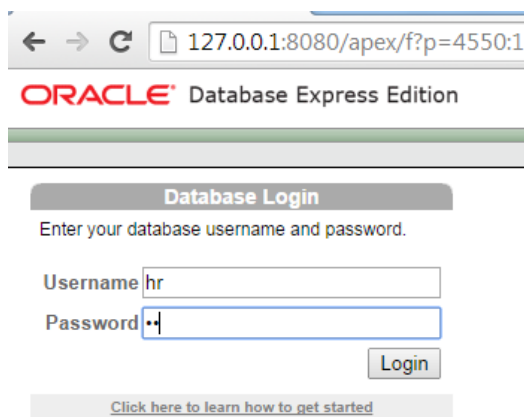
Kompetensi

Mampu memahami tentang membuat aplikasi untuk menampilkan data table menggunakan halaman blank yang dilengkapi dengan region

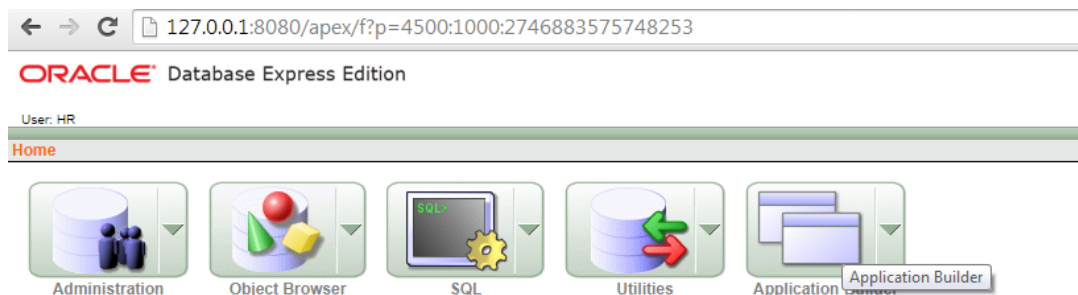
Membuat Aplikasi dengan Region

Pada modul ini disajikan langkah-langkah yang dibutuhkan untuk membuat aplikasi yang dapat menampilkan data yang terdapat di dalam sebuah tabel. Berbeda dengan langkah-langkah yang telah disampaikan pada modul sebelumnya, pada modul ini kita akan mengembangkan aplikasi dengan menggunakan halaman berjenis “blank”. Kemudian halaman “blank” tersebut akan kita lengkapi dengan dua buah *region* berjenis *report*. Selamat mencoba.

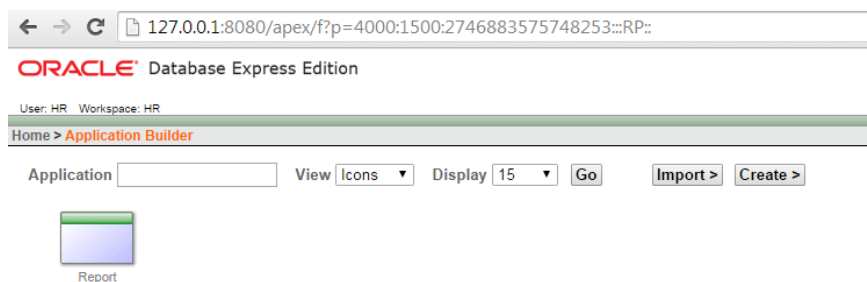
Saat ini kita masih menggunakan user HR, login dengan user dan password HR



Akan tampil seperti menu berikut ini, pilih Application Builder



Klik Create



Pilih menu Create Application, lalu klik Next

127.0.0.1:8080/apex/f?p=4000:56:2746883575748253::NO:56,103,104,106,130,131,4011,4012,4

ORACLE Database Express Edition

User: HR Workspace: HR

Home > **Create Application**

Method




Define Application

Confirm

Create Application

An application is a collection of pages linked together using tabs, buttons, or hypertext links. The pages in an application share a common session state definition and authentication method.

☒ Create Application ☐ Create from Spreadsheet ☐ Demonstration Application



Maka akan muncul menu create application

127.0.0.1:8080/apex/f?p=4000:3000:2746883575748253:CC:NO:3000,3001,3020::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > **Create Application**

Method

Name

Pages

Tabs

Shared Components

Attributes

User Interface

Confirm

Create Application

Enter an application name and a unique application ID. Then, select an application creation method and a schema.

Name

Application 100

Create Application: ☒ From scratch ☐ Based on existing application design model

Schema HR

Isikan name : (misalnya Membangun Report dengan Region), pilih Schema : HR lalu klik Next

127.0.0.1:8080/apex/f?p=4000:3000:2746883575748253:CC:NO:3000,3001,3020::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > **Create Application**

Method

Name

Pages

Tabs

Shared Components

Attributes

User Interface

Confirm

Create Application

Enter an application name and a unique application ID. Then, select an application creation method and a schema.

Name Membangun Report dengan Region

Application 100

Create Application: ☒ From scratch ☐ Based on existing application design model

Schema HR

Klik Add Page

← → ↻ 127.0.0.1:8080/apex/f?p=4000:3001:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Create Application

Method

Name

Pages

Tabs

Shared Components

Attributes

User Interface

Confirm

Create Application






Cancel < Previous Next >

Add pages to your application by selecting a page type and clicking Add Page.

Add Page

Select Page Type:

☒ Blank ☐ Report ☐ Form ☐ Tabular Form ☐ Report and Form

Action: Add blank page to application

Page Name

Klik Next

← → ↻ 127.0.0.1:8080/apex/f?p=4000:3001:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Create Application

Method

Name

Pages

Tabs

Shared Components

Attributes

User Interface

Confirm

Create Application






Cancel < Previous Next >

Page	Page Name	Page Type	Source Type	Source
1	Page 1	Blank	-	-

Add Page

Select Page Type:

☒ Blank ☐ Report ☐ Form ☐ Tabular Form ☐ Report and Form

Action: Add blank page to application

Subordinate to Page - Top Level Page -

Page Name

Pilih Menu One Level of Tabs, lalu klik next

← → ↻ 127.0.0.1:8080/apex/f?p=4000:437:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Create Application

Method

Name

Pages

Tabs

Shared Components

Attributes

User Interface

Confirm

Create Application




Cancel < Previous Next >

Application: 100

Name: Membangun Report dengan Region

Tabs:

☐ No Tabs ☒ One Level of Tabs ☐ Two Levels of Tabs

Menu berikutnya klik Next

← → ↻ 127.0.0.1:8080/apex/www_flow.accept

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Create Application

Method
Name
Pages
Tabs
Shared Components
Attributes
User Interface
Confirm

Create Application Cancel < Previous Next >

Shared components are common application elements that can be displayed or applied across multiple pages in an application. To save time or maintain consistency between applications, it is possible to copy the shared components from an existing application.

Copy Shared Components from Another Application:

☐ Yes
☒ No

Pilih Authentication Scheme pilih No Authentication, lalu klik Next

← → ↻ 127.0.0.1:8080/apex/f?p=4000:3020:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Create Application

Method
Name
Pages
Tabs
Shared Components
Attributes
User Interface
Confirm

Create Application Cancel < Previous Next >

* Authentication Scheme:
Database Account
Application Express Authentication
No Authentication
Database Account
English (United States) (en-us)

* User Language Preference Derived From:
Use Application Primary Language

Pilih theme yang diinginkan, lalu klik next

← → ↻ 127.0.0.1:8080/apex/f?p=4000:3004:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Create Application

Method
Name
Pages
Tabs
Shared Components
Attributes
User Interface
Confirm

Create Application Cancel < Previous Next >

Select a theme:

☒ Theme 1
☐ Theme 2
☐ Theme 3
☐ Theme 4
☐ Theme 5
☐ Theme 6
☐ Theme 7
☐ Theme 8
☐ Theme 9
☐ Theme 10
☐ Theme 11
☐ Theme 12

Klik Create untuk melanjutkan proses berikutnya

← → × 127.0.0.1:8080/apex/f?p=4000:3005:2746883575748253:::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Create Application

Method

Name

Pages

Tabs

Shared Components

Attributes

User Interface

Confirm

Create Application

Cancel < Previous Create

You have requested to create an application with the following attributes. Please confirm your selections.

Application	100
Name	Membangun Report dengan Region
Parsing Schema	HR
Default Language	en-us
Tabs	One Level of Tabs
Default Authentication Scheme	No Authentication
UI Theme	12

☒ Save this definition as a design model for reuse

Application Created successfully

← → ↻ 127.0.0.1:8080/apex/f?p=4000:1:2746883575748253:::&success_msg=Application%20created%20successfully.%2F9A60EA7

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100

✓ Application created successfully.

Application: 100 - Membangun Report dengan Region

Page View Icons Display 15 Go Create Page >

1-2

Klik Page yang tadi kita buat baru (page 1)

← → ↻ 127.0.0.1:8080/apex/f?p=4000:1:2746883575748253:::&success_msg=Application%20created%20suc

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100

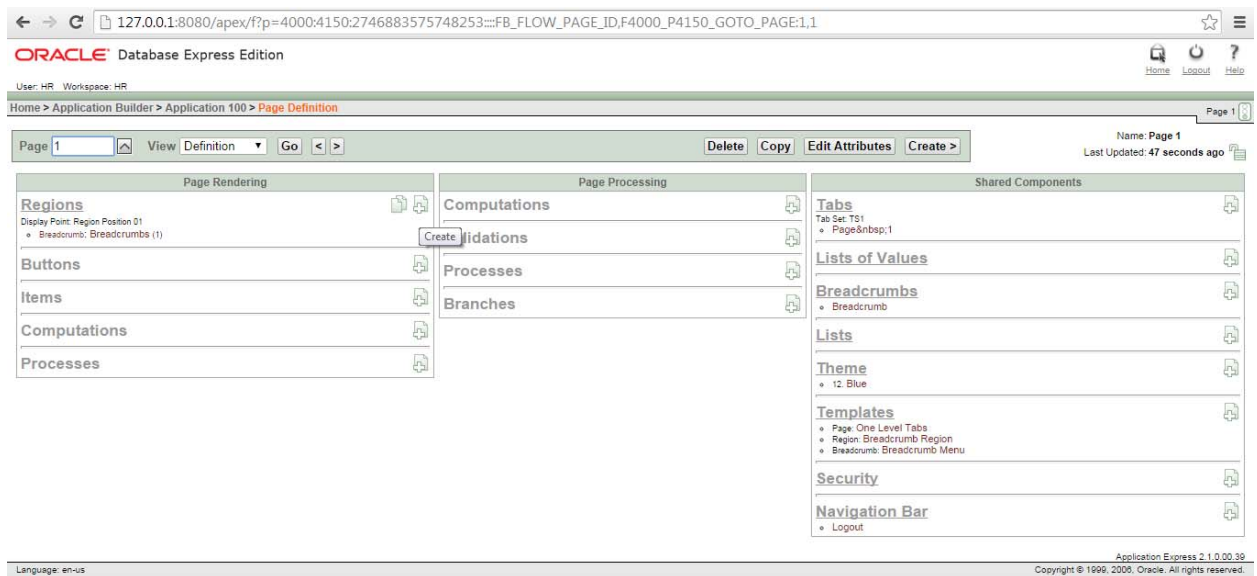
✓ Application created successfully.

Application: 100 - Membangun Report dengan Region

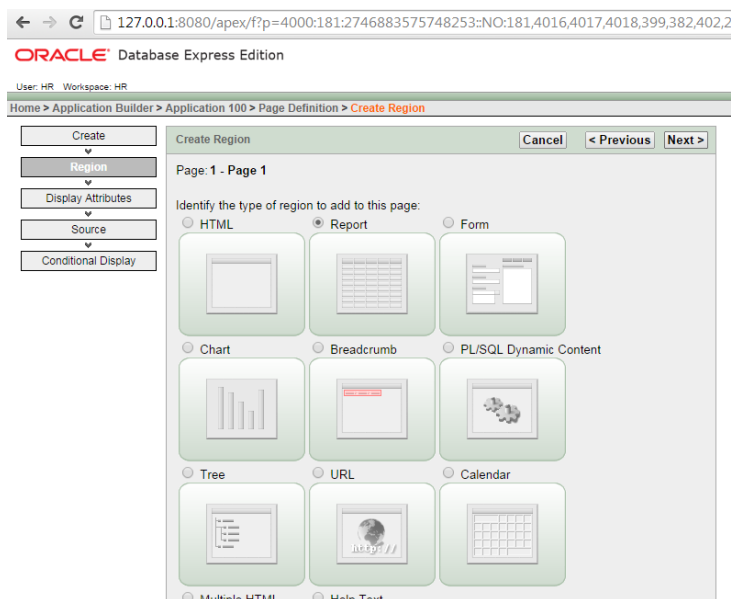
Page View Icons Display 15 Go Create Page >

1-2

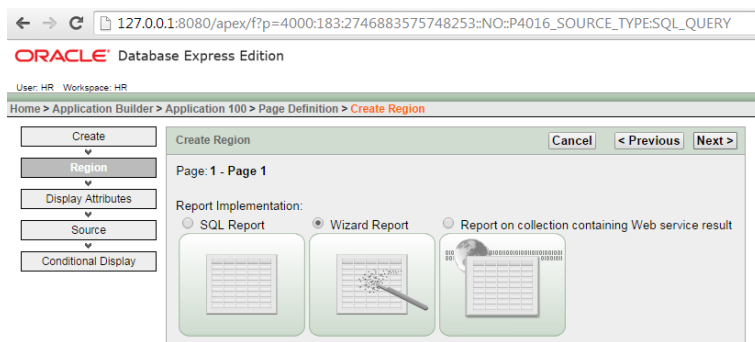
Pada menu page definition>page rendering>region> klik button create



Pilih Report lalu klik Next



Pilih Wizard Report lalu Next



Isikan Title (misal : Tabel Countries), Region template : Reports Region lalu next

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

Page: 1 - Page 1

Region Source Type: Structured Query

Title: Tabel Countries

Region Template: Reports Region

Display Point: Page Template Body (3 items above region content)

Sequence: 11 Column: 1

Pada menu di bawah ini, pilih table yang kita inginkan, dalam hal ini kita menggunakan table Countries, pilih columns dari table countries dan pindahkan ke columns selected

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

Table / View Owner: HR

Table / View Name: COUNTRIES(table)

Select Columns: COUNTRY_ID, COUNTRY_NAME, REGION_ID

Columns Selected:

Klik next

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

Table / View Owner: HR

Table / View Name: COUNTRIES

Show Related Tables Only: Yes

Select Columns:

Columns Selected: COUNTRIES.COUNTRY_ID, COUNTRIES.COUNTRY_NAME, COUNTRIES.REGION_ID

Klik Create Region

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

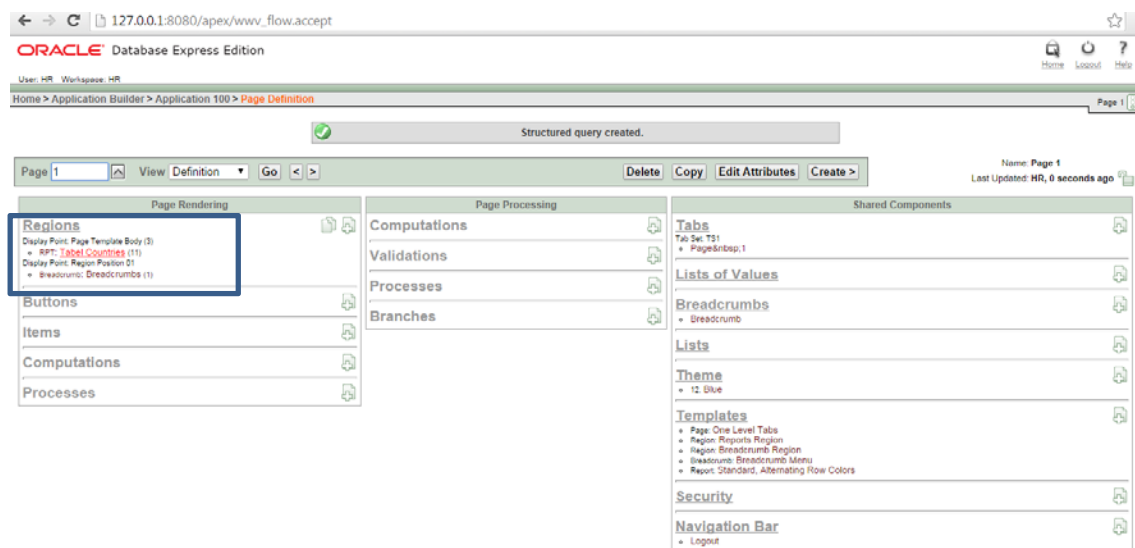
Page: 1 - Page 1

Report Template: template: 12. Standard, Alternating Row Colors

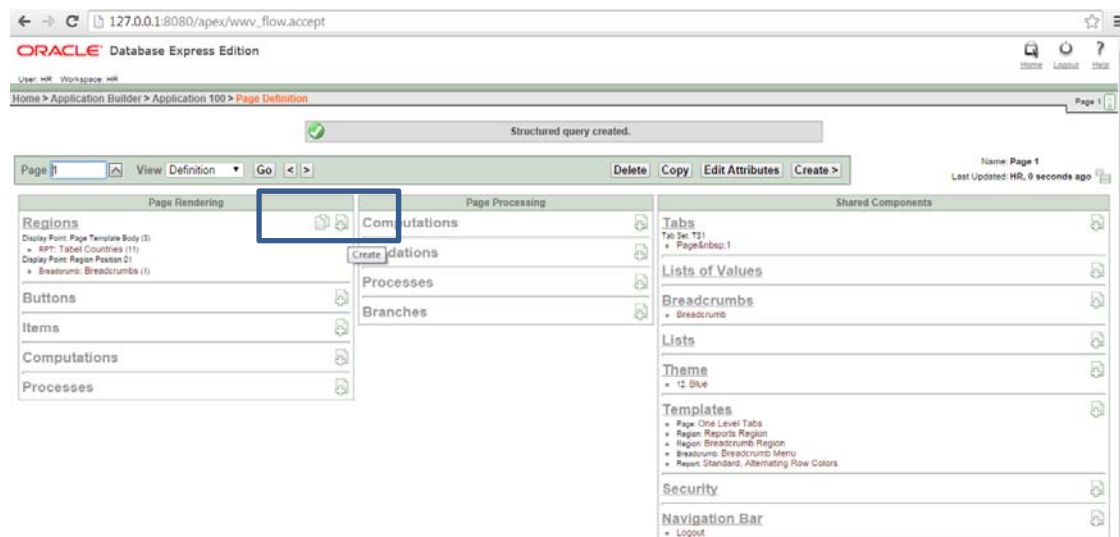
Rows Per Page: 15

SQL Query

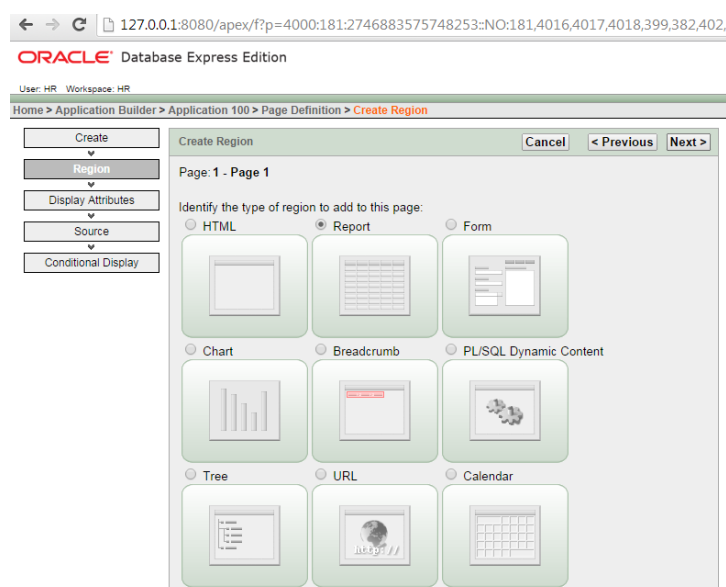
Di dalam menu Region sudah tampil Tabel Countries yang sudah kita buat



Untuk membuat table Region yang baru klik Create



Pilih Report lalu klik Next



Pilih Wizard Report lalu klik Next

127.0.0.1:8080/apex/f?p=4000:183:2746883575748253::NO:P4016_SOURCE_TYPE:SQL_QUERY

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

Page: 1 - Page 1

Report Implementation:

☐ SQL Report ☒ Wizard Report ☐ Report on collection containing Web service result

Cancel < Previous Next >

Isikan title dengan nama "Tabel Region"

127.0.0.1:8080/apex/f?p=4000:4016:2746883575748253::NO:P4016_SOURCE_TYPE:STRUCTURE

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

Page: 1 - Page 1

Region Source Type: Structured Query

* Title: Tabel Region

Region Template: Reports Region

Display Point: Page Template Body (3 items above region content)

* Sequence: 21 Column: 1

Cancel < Previous Next >

Karena kita mau menggunakan report tersebut dalam 1 halaman, maka column kita ganti menjadi 2

127.0.0.1:8080/apex/f?p=4000:4016:2746883575748253::NO:P4016_SOURCE_TYPE:STRUCTURE

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

Page: 1 - Page 1

Region Source Type: Structured Query

* Title: Tabel Region

Region Template: Reports Region

Display Point: Page Template Body (3 items above region content)

* Sequence: 21 Column: 2

Cancel < Previous Next >

Pilih Tabel Region

127.0.0.1:8080/apex/f?p=4000:399:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Create Region

Table/ View Owner: HR

Table / View Name: REGIONS(table)

Select Columns: REGION_ID, REGION_NAME

Columns Selected: REGION_ID, REGION_NAME

Cancel < Previous Next >

Select table region lalu klik Next

127.0.0.1:8080/apex/f?p=4000:399:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Table and Columns
Join Conditions
Report Attributes

Create Region

Table/ View Owner: HR
Table / View Name: REGIONS
Show Related Tables Only: ☒ Yes ☐ No

Select Columns: Columns Selected
REGIONS.REGION_ID
REGIONS.REGION_NAME

Cancel < Previous Next >

Klik Create Region

127.0.0.1:8080/apex/f?p=4000:402:2746883575748253::NO::

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition > Create Region

Table and Columns
Join Conditions
Report Attributes

Create Region

Page: 1 - Page 1
Report Template: template: 12. Standard, Alternating Row Colors
Rows Per Page: 15

Cancel < Previous Create Region

SQL Query

Maka Tabel region yang baru saja kita buat sudah terdapat 2 buah

127.0.0.1:8080/apex/www_flow.accept

ORACLE Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition

Structured query created.

Page 1 View Definition Go < > Delete Copy Edit Attributes Create >

Name: Page 1
Last Updated: HR, 0 seconds ago

Page Rendering

- Regions
 - Display Point: Page Template Body (3)
 - RPT: Tabel Countries (11) → RPT: Tabel Region (21)
 - Display Point: Region Position 01
 - Breadcrumb: Breadcrumbs (1)
- Buttons
- Items
- Computations
- Processes

Page Processing

- Computations
- Validations
- Processes
- Branches

Shared Components

- Tabs
 - Tab Set: TS1
 - Page 1
- Lists of Values
- Breadcrumbs
 - Breadcrumb
- Lists
- Theme
 - 12. Blue
- Templates
 - Page: One Level Tabs
 - Region: Reports Region
 - Region: Breadcrumb Region
 - Breadcrumb: Breadcrumb Menu
 - Report: Standard, Alternating Row Colors
- Security
- Navigation Bar
 - Logout

Untuk melihat hasil report yang kita baru saja buat klik Run Page

Oracle Database Express Edition

User: HR Workspace: HR

Home > Application Builder > Application 100 > Page Definition

Structured query created.

Page 1 View Definition Go < > Delete Copy Edit Attributes Create >

Name: Page 1
Last Updated: HR, 0 seconds ago

Page Rendering

- Regions
 - Display Point: Page Template Body (3)
 - RPT: Tabel Countries (11)
 - RPT: Tabel Region (21)
 - Display Point: Region Position 01
 - Breadcrumb: Breadcrumbs (1)
- Buttons
- Items
- Computations
- Processes

Page Processing

- Computations
- Validations
- Processes
- Branches

Shared Components

- Tabs
 - Tab Set: TS1
 - Page 1
- Lists of Values
- Breadcrumbs
 - Breadcrumb
- Lists
- Theme
 - 12 Blue
- Templates
 - Page: One Level Tabs
 - Region: Reports Region
 - Region: Breadcrumb Region
 - Breadcrumb: Breadcrumb Menu
 - Report: Standard, Alternating Row Colors
- Security
- Navigation Bar
 - Logout

Hasil report

ANONYMOUS Logout

Page 1

Tabel Countries

Country Id	Country Name	Region Id
AR	Argentina	2
AU	Australia	3
BE	Belgium	1
BR	Brazil	2
CA	Canada	2
CH	Switzerland	1
CN	China	3
DE	Germany	1
DK	Denmark	1
EG	Egypt	4
FR	France	1
HK	HongKong	3
IL	Israel	4
IN	India	3
IT	Italy	1

row(s) 1 - 15 of 25

Tabel Region

Region Id	Region Name
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

1 - 4

Edit Application Edit Page 1 Create Session Debug Show Edit Links

Daftar Pustaka

Oracle. 2006. *Oracle 10 g Express Edition Help System*.



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 3 Table

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

03

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam SQL Plus dalam pembuatan table dan manipulasi table

Kompetensi

Mampu memahami perintah-perintah dalam SQL Plus dalam pembuatan table dan manipulasi table

Table

1. Menggunakan Dictionary Table

SQL>DESC dictionary

Mencari Informasi Table di user schema

SQL>SELECT table_name FROM user_tables;

Informasi Objek

SQL>COLUMN object_name FORMAT A15

SQL>COLUMN object_type FORMAT A15

SQL>SELECT object_name, object_type FROM user_objects;

2. TABEL

Membuat Tabel

SQL> CREATE TABLE product

```
2      (kode_product   VARCHAR2(5),
3      nama_product    VARCHAR2(20),
4      harga_jual       NUMBER(10),
5      satuan           VARCHAR2(10),
6      stock            NUMBER(4)
7      );
```

Untuk mengisi datanya gunakan perintah INSERT

SQL>INSERT INTO product VALUES('AT001','BUKU',1500,'pcs',100);

SQL>INSERT INTO product VALUES('AT002','PENSIL',1000,'pcs',50);

SQL>INSERT INTO product VALUES('AT003','PENGGARIS',2000,'pcs',20);

SQL> Create table mahasiswa(

```
2  nim char(8) primary key,
3  nama varchar2(50) not null,
4  tanggal date    default sysdate,
5  jnskel char(1)   check(jnskel IN('P','L')),
6  email varchar2(100)
7 );
```

Untuk mengisi datanya gunakan perintah INSERT

```
SQL>INSERT INTO Mahasiswa VALUES('22889292','Mrs Waru','10-Mar-80','P',null);
SQL>INSERT INTO Mahasiswa VALUES('22997878','Mr Panah',
Default,'L','nobody@yahoo.com');
```

3. Default Option

```
SQL> CREATE TABLE penjualan
2      (no_order      Number(5),
3      tgl_transaksi  Date Default sysdate,
4      nama_pembeli   Varchar2(15) Default 'Noname',
5      rp_transaksi   Number(10) Default 0);
```

Menampilkan informasi table lebih detail dari dictionary USER_TAB_COLUMNS

```
SQL> BREAK ON TABLE_NAME
SQL> COL TABLE_NAME FORMAT A10
SQL> COL COLUMN_NAME FORMAT A15
SQL> COL DATA_TYPE FORMAT A15
SQL> COL DATA_DEFAULT FORMAT A15
SQL> SELECT table_name,
2      column_name,
3      data_type,
4      data_default
5      from user_tab_columns
6      WHERE TABLE_NAME = 'PENJUALAN';
```

Cara memberi nilai default ada dua cara yaitu :

1. Mengabaikan kolom yang didefinisikan dengan nilai default

```
SQL> INSERT INTO penjualan (no_order) Values(1);
```

2. Menyebutkan secara eksplisit dengan DEFAULT

```
SQL> INSERT INTO penjualan
2      VALUES (2,DEFAULT,'Toni',20000);
```

4. Membuat table dengan Subquery

```
SQL> CREATE TABLE copy_pegawai
2      AS
3      SELECT * FROM pegawai;
```



```
SQL> CREATE TABLE copy_peg41
2      AS
3      SELECT * FROM pegawai
4      WHERE kode_bag=41;
```

5. Modifikasi Struktur Tabel

Menambah kolom

```
SQL> ALTER TABLE product
2  ADD (pct_diskon  NUMBER(5,2) DEFAULT 0,
3  keterangan  VARCHAR2(30));
*Cek Struktur Table  SQL>DESC product
```

Modifikasi Kolom

Tambah jumlah karakter pada Kolom

```
SQL> ALTER TABLE product
2  MODIFY (nama_product varchar2(25));
```

Ubah nilai default pada kolom

```
SQL> ALTER TABLE product
2  MODIFY (pct_diskon DEFAULT 10);
```

Menghapus kolom

```
SQL> ALTER TABLE product
2  DROP COLUMN keterangan;
```

Mengubah nama Column

```
SQL> ALTER TABLE product
2  RENAME COLUMN harga_jual TO harga_jual_product;
```

Set Kolom menjadi UNUSED (kolom yang tidak digunakan)

```
SQL> ALTER TABLE product SET UNUSED COLUMN satuan;
```

Informasi jumlah kolom yang sudah di UNUSED

```
SQL>SELECT * from user_unused_col_tabs;
```

Menghapus kolom yang di set UNUSED

```
SQL> ALTER TABLE product
2 DROP UNUSED COLUMNS;
```

6. Constraint Pada Tabel (batasan)

Mengetahui Constraint pada table

```
SQL> SELECT constraint_name, table_name, constraint_type from user_constraints
where table_name='PRODUCT';
```

Primary Key

```
SQL> ALTER TABLE PRODUCT
2 ADD CONSTRAINT pk_product PRIMARY KEY (kode_product);
```

Unique

```
SQL> ALTER TABLE PRODUCT
2 ADD CONSTRAINT uq_product UNIQUE (nama_product);
```

Check

```
SQL> ALTER TABLE pegawai
2 ADD CONSTRAINT ck_gaji CHECK (gaji>=1000000);
```

Not Null

```
SQL> ALTER TABLE pegawai
2 Modify userid constraint userid_nn NOT NULL;
```

Foreign Key

```
SQL> ALTER TABLE pegawai
2 ADD CONSTRAINT fk_kode_bag FOREIGN KEY (kode_bag)
3 REFERENCES bagian(kode_bag);
```

Latihan Membuat Tabel dan Constrains

```
SQL> CREATE TABLE department
2 (kode NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,
3 nama_dept Varchar2(10) NOT NULL);
```

```
SQL> CREATE TABLE employee
2 (id_emp NUMBER(5) CONSTRAINT pk_ide PRIMARY KEY,
3 nama varchar2(15),
```

```

4   email          varchar2(25) constraint uq_email_emp UNIQUE,
5   jen_kel        char(1) CONSTRAINT ck_jen CHECK (jen_kel IN ('L','P')),
6   gaji           NUMBER(10) CONSTRAINT ck_gaji_emp CHECK(gaji>=1000000),
7   kode           NUMBER(5) CONSTRAINT   fk_kode_dept REFERENCES
department(kode))

```

Mengetahui hasil CONSTRAINT

```
SQL> SET LINESIZE 200
```

```
SQL> SET PAGESIZE 50
```

```
SQL> select constraint_name, table_name, constraint_type
2  from user_constraints where
3  table_name IN('EMPLOYEE','DEPARTEMENT');
```

CONSTRAINT_NAME	TABLE_NAME	C
-----	-----	-
SYS_C004153	DEPARTMENT	C
PK_DEPT	DEPARTMENT	P
CK_JEN	EMPLOYEE	C
CK_GAJI_EMP	EMPLOYEE	C
PK_IDE	EMPLOYEE	P
UQ_EMAIL_EMP	EMPLOYEE	U
FK_KODE_DEPT	EMPLOYEE	R

Type Constraint C=Check, P= Primary, U=Unique, R=Reference

Menguji apakah Constraint sudah berlaku??

```
SQL> INSERT INTO Department
```

```
2 values(10,'ADMIN');
```

```
SQL> INSERT INTO Department
```

```
2 VALUES(10,'SALES');
```

ERROR at line 1:

ORA-00001: unique constraint (SALES.PK_DEPT) violated

Data kedua tidak masuk karena terdapat Constraint PK_Dept (Primary Key) angka 10 di Insert sebanyak dua kali.

Ulangi Perintah ini

```
SQL> INSERT INTO Department
```

```
2   VALUES(20,'SALES');
```

```
SQL> INSERT INTO employee
```

```
2   VALUES(3,'Dhila','dhila@yahoo.com','P',10000000,10);
```

```
SQL> INSERT INTO employee
```

```
2   VALUES(4,'Dahlia','dahlia@yahoo.com','W',15000000,2
```

ERROR at line 1:

ORA-02290: check constraint (SALES.CK_JEN) violated

7. Disable dan Enable Constraint

Langkan pertama adalah mengecek dahulu status constraintnya dengan perintah

```
SQL> SELECT constraint_name, table_name, status
```

```
2 FROM user_constraints WHERE
```

```
3 table_name IN('EMPLOYEE','DEPARTEMENT');
```

Disable

```
SQL> ALTER TABLE EMPLOYEE
```

```
2   DISABLE CONSTRAINT ck_gaji_emp;
```

Enable

```
SQL> ALTER TABLE EMPLOYEE
```

```
2   ENABLE CONSTRAINT ck_gaji_emp;
```

Menghapus Constraint

```
SQL> ALTER TABLE EMPLOYEE
```

```
2   DROP CONSTRAINT ck_gaji_emp;
```

8. Option On Delete Cascade dan ON Delete Set Null

Pada pendefinisian hubungan antartabel, satu kolom pada table berfungsi sebagai kolom rujukan (kolom induk), sedangkan kolom satunya lagi disebut kolom anak, yaitu kolom yang merunjuk ke kolom induk. Secara default, nilai-nilai di kolom induk tidak boleh dihapus selama masih dirujuk oleh kolom anak. Contohnya data department 10 di table tidak bisa dihapus karena masih ada data di table employee yang *kode_dept*-nya 20.

```
SQL> DELETE departement
```

```
2 where kode=10;
```

```
DELETE departement
```

```
*
```

```
ERROR at line 1:
```

```
ORA-02292: integrity constraint (SALES.FK_KODE_DEPT) violated - child record found
```

Hal ini terjadi karena pada saat pendefinisian constraint `fk_kode_dept` tidak diikuti option `ON DELETE CASCADE` atau `ON DELETE SET NULL`.

ON DELETE SET NULL

```
SQL> Alter table employee
```

```
2 ADD CONSTRAINT fk_kode_dept FOREIGN KEY(kode)
```

```
3 REFERENCES departement(kode) ON DELETE SET NULL;
```

Coba Delete kembali

```
SQL> DELETE departement
```

```
2 where kode=10;
```

ON DELETE CASCADE

```
SQL> Alter table employee
```

```
2 ADD CONSTRAINT fk_kode_dept FOREIGN KEY(kode)
```

```
3 REFERENCES departement(kode) ON DELETE CASCADE;
```

9. Truncate Tabel

Perintah untuk menghapus data pada suatu table dan membebaskan space yang dipakai oleh table. Tuncate merupakan perintah DDL sehingga tidak bisa di `ROLLBACK`.

Berbeda dengan perintah `DELETE`, perintah `DELETE` hanya menghapus data dan tidak membesakan space, perintah delete masih bisa di rollback. Jika hanya dilihat dari sisi datanya, sekilas perintah `TRUNCATE` dan `DELETE` tidak ada bedanya.

```
SQL> CREATE TABLE COPY_PEG
```

```
2 as
```

```
3 SELECT * FROM pegawai;
```

Table created.

```

SQL> BEGIN
2  FOR X IN 1..8 LOOP
3  INSERT INTO COPY_PEG
4  SELECT * FROM COPY_PEG;
5  END LOOP;
6  COMMIT;
7  END;
8  /

```

PL/SQL procedure successfully completed.

```
SQL>ANALYZE TABLE copy_peg COMPUTE STATISTICS;
```

```

SQL> SELECT TABLE_NAME, NUM_ROWS, (BLOCKS*8192)/1024 as "KB"
2  FROM user_tables
3  WHERE table_name='COPY_PEG';

```

TABLE_NAME	NUM_ROWS	KB
-----	-----	-----
COPY_PEG	6400	488

Hapus data di table copy_peg dengan perintah DELETE

```
SQL> DELETE COPY_PEG;
```

6400 rows deleted.

```
SQL> COMMIT;
```

```
SQL> ANALYZE TABLE COPY_PEG COMPUTE STATISTICS;
```

```

SQL> SELECT TABLE_NAME, NUM_ROWS, (BLOCKS*8192)/1024 as "KB"
2  FROM user_tables
3  WHERE table_name='COPY_PEG';

```

TABLE_NAME	NUM_ROWS	KB
-----	-----	-----
COPY_PEG	0	488

Perintah TRUNCATE

```
SQL> INSERT INTO COPY_PEG SELECT * FROM PEGAWAI;
```

```
SQL> BEGIN
```

```
2 FOR X IN 1..8 LOOP
```

```
3 INSERT INTO COPY_PEG
```

```
4 SELECT * FROM COPY_PEG;
```

```
5 END LOOP;
```

```
6 COMMIT;
```

```
7 END;
```

```
8 /
```

```
SQL> ANALYZE TABLE COPY_PEG COMPUTE STATISTICS;
```

```
SQL> SELECT TABLE_NAME, NUM_ROWS, (BLOCKS*8192)/1024 as "KB"
```

```
2 FROM user_tables
```

```
3 WHERE table_name='COPY_PEG';
```

```
SQL> TRUNCATE TABLE COPY_PEG;
```

Cek kembali SQL> SELECT TABLE_NAME, NUM_ROWS, (BLOCKS*8192)/1024 as "KB"

```
2 FROM user_tables
```

```
3 WHERE table_name='COPY_PEG';
```

TABLE_NAME	NUM_ROWS	KB
-----	-----	-----
COPY_PEG	0	0

10. Menghapus Tabel

```
SQL> DROP TABLE employee;
```

```
SQL> DROP TABLE DEPARTMENT
```

```
SQL> CASCADE CONSTRAINTS;
```

11. Recycle Bin

```
SQL>DROP TABLE copy_peg;
```

```
SQL> SHOW RECYCLEBIN;
```

ORIGINAL NAME	RECYCLEBIN NAME	OBJECT TYPE	DROP TIME
COPY_PEGAWAI	BIN\$iU/Fylo8RgGuHSjSh5u2mw==\$0	TABLE	2012-09-17:12:42:49
EMPLOYEE	BIN\$I9ZOPSBiQYGTKBI72HO7jw==\$0	TABLE	2012-09-17:13:07:41

```
SQL>SELECT * FROM "BIN$I9ZOPSBiQYGTKBI72HO7jw==$0";
```

Untuk mengembalikan table copy_peg digunakan perintah FLASHBACK TABLE...TO BEFORE DROP.

```
SQL> FLASHBACK TABLE COPY_PEGAWAI TO BEFORE DROP;
```

Untuk menghapus table secara permanen, tidak tersimpan di Recyclebin, tambahkan option PURGE pada perintah DROP TABLE.

```
SQL> FLASHBACK TABLE COPY_PEGAWAI TO BEFORE DROP;
```

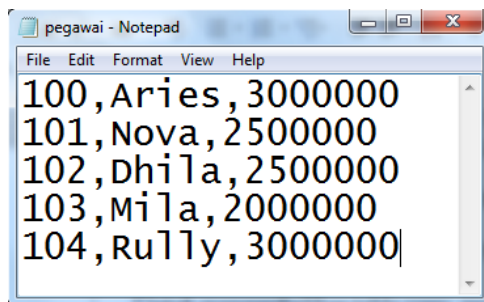
Lihat di RecycleBin

```
SQL> SHOW RECYCLEBIN;
```

Mengosongkan Recyclebin dari table-table yang pernah dihapus menggunakan perintah SQL>PURGE RECYCLEBIN;

12. External Table

Fungsi untuk melakukan loading data dari **Flat File** (Text File) ke dalam database Oracle. Untuk mempraktekkannya buat terlebih dahulu file text seperti dibawah ini :



Pegawai.txt

```
SQL> conn system
```

```
Enter password:*****
```

```
Connected.
```

```
SQL> CREATE DIRECTORY ext_dir AS 'C:\DATA';
```

```
Directory created.
```

```
SQL> GRANT read, write ON DIRECTORY ext_dir
```

```
2 TO sales;
```

```
Grant succeeded.
```

```
1 CREATE TABLE load_pegawai
```

```
2 (
```

```
3 id NUMBER(4),
```

```
4 NAMA CHAR(10),
```

```
5 GAJI NUMBER(8)
```

```
6 )
```

```
7 ORGANIZATION external
```

```
8 (
```

```
9 TYPE ORACLE_loader
```

```
10 DEFAULT DIRECTORY ext_dir
```

```
11 LOCATION ('pegawai.txt')
```

```
12 )
```

```
13 REJECT LIMIT unlimited;
```

Latihan Soal

1. Perintah yang digunakan untuk menampilkan data dari tabel pegawai adalah ?
2. Anda menggunakan SQL *Plus dan akan mengeksekusi beberapa fungsi matematika. Tabel apa yang anda akan gunakan untuk melakukan kalkulasi matematika dari ekspresi atau nilai yang bukan diambil dari tabel di dalam database ?
3. Anda ingin menulis Query untuk menampilkan data pegawai dengan memilih data tertentu berdasarkan kriteria. Pada klausa apa kriteria tersebut anda definisikan
4. Anda ingin menampilkan pegawai yang ada di bagian 30 dan namanya diawali oleh karakter 'A', perintah yang digunakan adalah
5. Karakter yang bisa digunakan sebagai wildcard atau pembentuk pola pada operator LIKE adalah

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 4 DATA MANIPULATION LANGUAGE

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

04

Kode MK
87043

Disusun Oleh
Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam SQL Plus dalam pembuatan table dan manipulasi table

Kompetensi

Mampu memahami perintah-perintah dalam SQL Plus dalam pembuatan table dan manipulasi table

Data Manipulation Language

1. Perintah SELECT

Digunakan untuk menampilkan data dari table.

```
SQL> SELECT id_pegawai,  
2          nama_pegawai,  
3          gaji  
4  FROM pegawai;
```

```
SQL> SELECT * from bagian;
```

2. Operasi Aritmatika

Ekspresi aritmetika dengan operand berupa kolom.

```
SQL> SELECT nama_pegawai,  
2  gaji,  
3  gaji+500000  
4  FROM pegawai;
```

Ekspresi Aritmetika dengan Operand

```
SQL> SELECT 20+30  
2  FROM Dual; → tabel dummy
```

```
SQL> SELECT 2*3  
2  FROM Dual;
```

```
SQL> SELECT sysdate  
2  FROM Dual;
```

```
SQL> SELECT User  
2  FROM Dual;
```

Table Dual

```
SQL> DESC dual;
```

Operari Aritmatika dengan Data Tanggal

```
SQL> SELECT to_date('20-SEP-08') +2  
2  FROM dual;
```

Tanggal-Numerik = Tanggal

```
SQL> SELECT to_date('20-SEP-08') - 1
```

```
2 FROM dual;
```

```
TO_DATE('
```

```
-----
```

```
19-SEP-08
```

Tanggal-Tanggal=Numerik

```
SQL> SELECT to_date('20-SEP-08') - to_date('15-SEP-08')
```

```
2 FROM dual;
```

```
TO_DATE('20-SEP-08')-TO_DATE('15-SEP-08')
```

```
-----
```

```
5
```

Tanggal+Tanggal=ERROR

```
SQL> SELECT to_date('20-SEP-08') + to_date('1-SEP-08')
```

```
2 FROM dual;
```

```
SELECT to_date('20-SEP-08') + to_date('1-SEP-08')
```

```
*
```

ERROR at line 1:

ORA-00975: date + date not allowed

Urutan Pengerjaan Operator

20+30/2 dikerjakan :30/2=15 -> 20+30

```
SQL> SELECT 20+30/2 FROM dual;
```

```
SQL> SELECT 2*20/2+3-1 FROM DUAL;
```

```
SQL> SELECT (20+30)/2 FROM DUAL;
```

3. Menggunakan Kolom Alias

Digunakan untuk mengganti kan nama kolom yang sudah ada.

```
SQL> SELECT nama_pegawai,
```

```
2 gaji gajipegawai
```

```
3 From pegawai;
```

```
SQL> SELECT nama_pegawai,
```

```
2 gaji,
```

```
3 gaji*0.1 kenaikan_gaji,
```

```
4 gaji + (gaji*0.1) TOTAL
```

```
5 FROM pegawai;
```

Jika ingin menambahkan space pada kolom alias, gunakan tanda petik ganda.

```
SQL> SELECT nama_pegawai,  
2 gaji,  
3 gaji*0.1 "Kenaikan Gaji",  
4 gaji + (gaji*0.1) "TOTAL Gaji"  
5 FROM pegawai;
```

Menggunakan kolom alias bisa juga dengan menggunakan keyword AS

```
SQL> SELECT nama_pegawai,  
2 gaji,  
3 gaji*0.1 AS "Kenaikan Gaji",  
4 gaji + (gaji*0.1) AS "TOTAL Gaji"  
5 FROM pegawai;
```

4. Concatination

Digunakan untuk mengkombinasikan beberapa kolom atau kolom dengan suatu text menjadi satu kolom output.

Contoh tanpa Concatination :

```
SQL> SELECT nama_pegawai,  
2 jabatan  
3 FROM pegawai;
```

Menggunakan Concatination

```
SQL> SELECT nama_pegawai || ' sebagai ' || jabatan AS " Pegawai dan Jabatan"  
2 FROM pegawai;
```

Pegawai dan Jabatan

MEUTIA JOVI MAHARANI sebagai PRESIDENT
BUDI HARTADI sebagai VP, OPERATIONS
RULLY SIANIPAR sebagai VP, SALES
EDWIN ASRUL sebagai VP, FINANCE
NOVI SETIAWATI sebagai VP, ADMINISTRATION
ARIS MURSI TO sebagai WAREHOUSE MANAGER
IRVAN SYAFE'I sebagai WAREHOUSE MANAGER
DHEA LUSIANA sebagai WAREHOUSE MANAGER

5. Nilai Null

Kolom yang belum ada nilainya di dalam database.

```
SQL> SELECT id_pegawai, nama_pegawai, email  
2 FROM pegawai;
```

ID_PEGAWAI	NAMA_PEGAWAI	EMAIL
------------	--------------	-------

1001	MEUTIA JOVI MAHARANI	
------	----------------------	--

1002	BUDI HARTADI	
------	--------------	--

1003	RULLY SIANIPAR	
------	----------------	--

1004	EDWIN ASRUL	
------	-------------	--

1005	NOVI SETIAWATI	
------	----------------	--

```
SQL> SELECT id_pegawai,  
2 nama_pegawai,  
3 NVL(email,'BELUM PUNYA EMAIL') as EMAIL  
4 FROM pegawai;
```

ID_PEGAWAI	NAMA_PEGAWAI	EMAIL
------------	--------------	-------

1001	MEUTIA JOVI MAHARANI	BELUM PUNYA EMAIL
------	----------------------	-------------------

1002	BUDI HARTADI	BELUM PUNYA EMAIL
------	--------------	-------------------

1003	RULLY SIANIPAR	BELUM PUNYA EMAIL
------	----------------	-------------------

1004	EDWIN ASRUL	BELUM PUNYA EMAIL
------	-------------	-------------------

1005	NOVI SETIAWATI	BELUM PUNYA EMAIL
------	----------------	-------------------

```
SQL> SELECT id_pegawai,  
2 nama_pegawai,  
3 gaji,  
4 gaji+ (pct_komisi*gaji) AS "Total"  
5 FROM pegawai;
```

```
SQL> SELECT id_pegawai,  
2 nama_pegawai,  
3 gaji,  
4 gaji+ (NVL(pct_komisi,0)*gaji) AS "Total"  
5 FROM pegawai;
```

6. Distinct

Digunakan untuk menghilangkan duplikasi nilai.

Sebelum menggunakan Distinct

```
SQL> SELECT id_plg FROM hd_sales;
```

ID_PLG

201

202

203

204

205

206

208

208

209

210

211

Menggunakan Distinct

```
SQL> SELECT DISTINCT(id_plg) FROM hd_sales;
```

ID_PLG

209

213

205

210

211

201

206

212

7. Where

Digunakan untuk menyeleksi data yang akan ditampilkan, perintah Where ditambahkan pada perintah SELECT setelah klausa FROM.

```
SQL> SELECT id_pegawai,
```



```

2  nama_pegawai,
3  gaji
4  FROM pegawai
5  WHERE id_pegawai=1020;

```

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
1020	ASWIR MATONDANG	1500000

```

SQL> SELECT id_pegawai,
2      nama_pegawai,
3      gaji
4  FROM pegawai
5  WHERE gaji<5000000;

```

```

SQL> SELECT nama_pegawai,
2      userid,
3      tgl_masuk
4  FROM pegawai
5  WHERE userid='edwirul'; -> Case Sensitif

```

```

SQL> SELECT nama_pegawai,
2      userid,
3      tgl_masuk
4  FROM pegawai
5  WHERE tgl_masuk >='01-jan-02'; ->Format Sensitif (DD:MM:RR)

```

Operator Between ... AND ...

```

SQL> SELECT nama_pegawai,
2      gaji
3  FROM pegawai
4  WHERE gaji BETWEEN 4000000 AND 6000000;

```

Operator IN

```

SQL> SELECT nama_pegawai,
2      gaji

```

```
3 FROM pegawai
4 WHERE gaji IN (2500000,4000000,5000000);
```

Operator LIKE

```
SQL> SELECT nama_pegawai
2 FROM pegawai
3 WHERE nama_pegawai LIKE 'S%';
```

```
SQL> SELECT nama_pegawai
2 FROM pegawai
3 WHERE nama_pegawai LIKE '_O%';
```

Operator IS NULL

```
SQL> SELECT id_pegawai, nama_pegawai, email
2 FROM pegawai
3 WHERE email is Null;
```

8. Logical Condition

Operator AND

```
SQL> SELECT id_pegawai,
2      nama_pegawai,
3      gaji,
4      kode_bag
5 FROM pegawai
6 WHERE kode_bag=41 AND gaji >8000000;
```

Operator OR

```
SQL> SELECT id_pegawai,
2      nama_pegawai,
3      gaji,
4      kode_bag
5 FROM pegawai
6 WHERE kode_bag=41 OR gaji >8000000;
```

Operator NOT

```
SQL> SELECT id_pegawai,
2      nama_pegawai,
```

```
3      gaji,  
4      kode_bag  
5 FROM pegawai  
6 WHERE NOT(kode_bag=41);
```

Urutan Pengerjaan Operator

```
SQL> SELECT id_pegawai,  
2      nama_pegawai,  
3      gaji,  
4      kode_bag  
5 FROM pegawai  
6 WHERE kode_bag=31 OR kode_bag=41 AND gaji > 8000000;
```

```
SQL> SELECT id_pegawai,  
2      nama_pegawai,  
3      gaji,  
4      kode_bag  
5 FROM pegawai  
6 WHERE (kode_bag=31 OR kode_bag=41) AND gaji > 8000000;
```

9. Order By

Mengurutkan secara Ascending (Default)

```
SQL> SELECT id_pegawai,  
2      nama_pegawai,  
3      gaji  
4 FROM pegawai  
5 ORDER BY gaji ASC;
```

Mengurutkan secara Descending

```
SQL> SELECT id_pegawai,  
2      nama_pegawai,  
3      gaji  
4 FROM pegawai  
5 ORDER BY gaji DESC;
```

Mengurutkan berdasarkan kolom Alias

```
SQL> SELECT id_pegawai,
```

```

2  nama_pegawai,
3  gaji,
4  pct_komisi,
5  gaji + (pct_komisi*gaji) as "Total"
6 FROM pegawai
7 ORDER BY "Total";

```

ID_PEGAWAI	NAMA_PEGAWAI	GAJI	PCT_KOMISI	Total
-----	-----	-----	-----	-----
1011	RIDWAN SANUSI	4000000	10	44000000
1013	POPY LUSIANA	5150000	10	56650000
1012	SUSI INDIARTI	4900000	12.5	66150000
1015	ESTI ARVINA	4500000	17.5	83250000
1014	ERTIN	5250000	15	84000000

Mengurutkan berdasarkan dua kolom

```

SQL> SELECT id_pegawai,
2  nama_pegawai,
3  kode_bag,
4  gaji
5 FROM pegawai
6 ORDER BY kode_bag, gaji DESC;

```

ID_PEGAWAI	NAMA_PEGAWAI	KODE_BAG	GAJI
-----	-----	-----	-----
1004	EDWIN ASRUL	10	10500000
1003	RULLY SIANIPAR	31	10000000
1014	ERTIN	31	5250000

Menggunakan urutan kolom pada Klausa SELECT

```

SQL> SELECT id_pegawai,
2  nama_pegawai,
3  kode_bag,
4  gaji
5 FROM pegawai
6 ORDER BY 3,4 DESC;

```

Angka 3 dan 4 menggambarkan urutan kolom yang ditampilkan 3(kode_bag), 4(gaji).

10. Perintah INSERT

Mengisi Data ke Semua Kolom.

```
SQL>INSERT INTO PEGAWAI (id_pegawai,  
                           nama_pegawai,  
                           userid,  
                           email,  
                           tgl_masuk,  
                           manager_id,  
                           jabatan,  
                           kode_bag,  
                           gaji,  
                           pct_komisi)  
values (1200, 'DianNovci','dinov','dinov@yahoo.com','01-JAN-2009','1005','STAFF  
ADMINISTRASI', 50, 2000000, 0);
```

```
SQL>INSERT INTO pegawai  
values(1201,'Tiara Insani','tiara','tiara@yahoo.com','01-JAN-2009','1005','STAFF  
ADMINISTRASI',50,2000000,0);
```

```
SQL>INSERT INTO pegawai(id_pegawai,nama_pegawai, tgl_masuk)  
2 VALUES (1202,'Tono Hendarto' ,'05-JAN-2010');
```

Data Date dengan Format non default

```
SQL>INSERT INTO pegawai (id_pegawai, nama_pegawai, tgl_masuk)  
2 VALUES (1203,'Dindin Haerudin', TO_DATE('20091001','YYYYMMDD'));
```

Data Dengan Tanda Kutip Tunggal

```
SQL>INSERT INTO pegawai (id_pegawai, nama_pegawai, tgl_masuk) VALUES  
(1204,'MA"Arif','05-JAN-2010');
```

Copy Pegawai dari satu Table ke Tabel Lain

```
SQL>CREATE TABLE copy_pegawai  
2 (nip NUMBER(4),  
3 nama VARCHAR(20),
```

```
4   tgl_masuk DATE,  
5   gaji      NUMBER(12))
```

SQL>INSERT into copy_pegawai (**nip, nama, tgl_masuk**)

```
2  SELECT id_pegawai, nama_pegawai, tgl_masuk  
3  FROM pegawai  
4  WHERE kode_bag=50;
```

11. Perintah UPDATE

SQL>UPDATE pegawai

```
2  SET gaji=2250000  
3  WHERE id_pegawai = 1201;
```

Correlated Update

SQL> UPDATE copy_pegawai x

```
2  SET gaji = (SELECT gaji  
               FROM pegawai WHERE id_pegawai = x.nip);
```

Kalusa RETURNING

Deklarasi variable dengan nama nilai_rata2 untu menampung hasil AVG(gaji).

SQL>VARIABEL nilai_rata2 NUMBER;

Lakukan UPDATE

```
SQL>UPDATE pegawai SET gaji = gaji*1.1  
2> RETURNING avg(gaji) INTO :nilai_rata2;
```

Cetak nilai Variabel

SQL> PRINT nilai_rata2;

12. Perintah DELETE

SQL> DELETE pegawai

```
2  WHERE id_pegawai=1024;
```

**)Hati-hati, perintah delete tanpa di ikuti klausa WHERE akan menghapus semua data yang ada di table.*

13. COMMIT dan ROLLBACK

SQL> SELECT id_pegawai, nama_pegawai, gaji

```

2 FROM pegawai
3 WHERE id_pegawai=1010;

```

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
-----	-----	-----
1010	PURNAMA RIYANTO	8070000

```

SQL> UPDATE pegawai
2 SET gaji=8500000
3 WHERE id_pegawai=1010;

```

```

SQL> COMMIT;
Commit complete.

```

```

SQL> SELECT id_pegawai, nama_pegawai, gaji
2 FROM pegawai
3 WHERE id_pegawai=1010;

```

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
-----	-----	-----
1010	PURNAMA RIYANTO	8500000

```

SQL> UPDATE pegawai
2 SET gaji=8700000
3 WHERE id_pegawai=1010;

```

```

SQL> ROLLBACK;
Rollback complete.

```

```

SQL> SELECT id_pegawai, nama_pegawai, gaji
2 FROM pegawai
3 WHERE id_pegawai=1010;

```

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
-----	-----	-----
1010	PURNAMA RIYANTO	8500000

Perintah Commit di gunakan untuk membuat transaksi menjadi **Permanen** dan tidak bisa di ROLLBACK (di kembalikan seperti semula sebelum dilakukan manipulasi)

14. Save Point

Contoh Penggunaan SAVE POINT.

```
SQL> SELECT id_pegawai, nama_pegawai, gaji
```

```
2 FROM pegawai
```

```
3 WHERE id_pegawai=1011;
```

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
1011	RIDWAN SANUSI	4000000

```
SQL> UPDATE pegawai
```

```
2 SET gaji=4100000
```

```
3 WHERE id_pegawai=1011;
```

```
SQL> SELECT id_pegawai, nama_pegawai, gaji
```

```
2 FROM pegawai
```

```
3 WHERE id_pegawai=1011;
```

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
1011	RIDWAN SANUSI	4100000

```
SQL> SAVEPOINT tahap1;
```

Savepoint created.

```
SQL> UPDATE pegawai
```

```
2 SET gaji=4200000
```

```
3 WHERE id_pegawai=1011;
```

```
SQL> SELECT id_pegawai, nama_pegawai, gaji
```

```
2 FROM pegawai
```

```
3 WHERE id_pegawai=1011;
```


ID_PEGAWAI	NAMA_PEGAWAI	GAJI
-----	-----	-----
1011	RIDWAN SANUSI	4200000

SQL> SAVEPOINT tahap2;
Savepoint created.

SQL> UPDATE pegawai
 2 SET gaji=4300000
 3 WHERE id_pegawai=1011;

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
-----	-----	-----
1011	RIDWAN SANUSI	4300000

SQL> ROLLBACK TO SAVEPOINT tahap1;
Rollback complete.

SQL> SELECT id_pegawai, nama_pegawai, gaji
 2 FROM pegawai
 3 WHERE id_pegawai=1011;

ID_PEGAWAI	NAMA_PEGAWAI	GAJI
-----	-----	-----
1011	RIDWAN SANUSI	4100000

SQL> ROLLBACK TO SAVEPOINT tahap2;
Rollback ROLLBACK TO SAVEPOINT tahap2
 *

ERROR at Line 1:

Latihan Soal

1. Untuk mengkombinasikan beberapa kolom atau kolom dengan suatu text menjadi satu kolom output menggunakan perintah ?
2. Perintah Save point digunakan untuk ?
3. Perintah yang digunakan untuk menghapus tabel ?
4. Sebutkan logical condition yang bisa kita gunakan dalam proses query ?
5. Perintah Where digunakan untuk ?

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 5 SQL FUNCTION

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

05

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam SQL Function

Kompetensi

Mampu memahami perintah-perintah dalam SQL Function

SQL Function

SQL Function di bagi menjadi 2 Bagian.

1. **Single Row Function**
2. **Group Row Function**

Single Row Function

1. **Character Function**

ASCII() dan CHR()

Digunakan untuk mengetahui nilai ASCII dan CHR

```
SQL> SELECT ASCII('a'), ASCII('A')  
2 FROM DUAL;
```

```
SQL> SELECT CHR(97), CHR(65)  
2 FROM DUAL;
```

CONCAT()

CONCAT(x,y) Digunakan untuk menggabungkan karakter x dan y.

```
SQL> SELECT CONCAT(id_pegawai, nama_pegawai)  
2 FROM pegawai  
3 WHERE kode_bag=31;
```

```
SQL> SELECT CONCAT('saya',' menulis cerita')  
2 FROM dual;
```

INITCAP(), LOWER(), UPPER()

Initcap -> mengubah karakter awal dari string x menjadi huruf capital.

Lower -> mengubah string x menjadi huruf kecil

Upper -> mengubah string x menjadi huruf capital

```
SQL> SELECT 'sATu kaTa' as "String",  
2 INITCAP('sATu kaTa') as "Initcap",  
3 LOWER('sATu kaTa') as "Lower",  
4 UPPER('sATu kaTa') as "Upper"  
5 From Dual;
```

String	Initcap	Lower	Upper
-----	-----	-----	-----
sATu kaTa Satu Kata		satu kata	SATU KATA

```
SQL> SELECT nama_pegawai
2 FROM pegawai
3 WHERE nama_pegawai='ahmad kosasih';
no rows selected
```

```
SQL> SELECT nama_pegawai
2 FROM pegawai
3 WHERE LOWER(nama_pegawai)='ahmad kosasih';
```

```
NAMA_PEGAWAI
-----
AHMAD KOSASIH
```

INSTR()

INSTR(x,y) Digunakan untuk mencari posisi suatu huruf pada suatu kalimat.

```
SQL> SELECT nama_pegawai,
2 INSTR(nama_pegawai,'A') as "Posisi 'A'"
3 FROM pegawai;
```

LENGTH()

Digunakan untuk mengembalikan jumlah karakter dari string x.

```
SQL> SELECT nama_pegawai,
2 LENGTH(nama_pegawai) AS "Jml Karakter"
3 FROM pegawai;
```

LTRIM(), RTRIM dan TRIM()

LTRIM -> digunakan untuk memotong string pada sisi kiri

RTRIM -> digunakan untuk memotong string pada sisi kanan

TRIM -> digunakan untuk memotong string pada sisi kiri dan kanan

```
SQL> SELECT 'aabbaabbaa' as "STRING",
2 LTRIM('aabbaabbaa','a') as "LTRIM",
3 RTRIM('aabbaabbaa','a') as "RTRIM",
4 TRIM('a' FROM 'aabbaabbaa') as "TRIM"
5 FROM dual;
```

STRING	LTRIM	RTRIM	TRIM
-----	-----	-----	-----
aabbaabbaa	bbaabbaa	aabbaabb	bbaabb

REPLACE

REPLACE(x,y,z) digunakan untuk mengganti substring y menjadi substring z pada string x.

```
SQL> SELECT nama_brg,
2 REPLACE(nama_brg,'Gigabyte','GB') as BARU
3 FROM barang
4 WHERE nama_brg LIKE 'G%';
```

SUBSTR()

SUBSTR(x,y [,z]) digunakan untuk menampilkan substring mulai dari posisi y sebanyak z karakter dari string x.

```
SQL> SELECT nama_pegawai,
2 SUBSTR(nama_pegawai,1,4),
3 SUBSTR(nama_pegawai,-10,5)
4 FROM pegawai;
```

NAMA_PEGAWAI	SUBSTR(NAMA_PEGAWAI,	
SUBSTR(NAMA_PEGAWAI,		
-----	-----	-----
MEUTIA JOVI MAHARANI	MEUT	I MAH
BUDI HARTADI	BUDI	DI HA
RULLY SIANIPAR	RULL	Y SIA
EDWIN ASRUL	EDWI	DWIN
NOVI SETIAWATI	NOVI	SETI

ARIS MURSITO	ARIS	IS MU
IRVAN SYAFE'I	IRVA	AN SY

Kombinasi Character Function

```
SQL> SELECT nama_pegawai,
2 LOWER(CONCAT(SUBSTR(nama_pegawai,1,4),'@ins.com')) AS "EMAIL"
3 FROM pegawai;
```

NAMA_PEGAWAI	EMAIL
-----	-----
MEUTIA JOVI MAHARANI	meut@ins.com
BUDI HARTADI	budi@ins.com

2. NUMERIC FUNCTION

ABS()

Digunakan untuk mengembalikan nilai Absolut

```
SQL> SELECT -10 AS "Nilai1", ABS(-10) as "Absolut"
2 FROM dual;
Nilai1 Absolut
```

```
-----
-10    10
```

CEIL()

CEIL(X) -> Digunakan untuk pembulatan ke atas dari nilai decimal x.

```
SQL> SELECT CEIL(5.1)
2 FROM dual;
CEIL(5.1)
```

```
-----
6
```

FLOOR()

FLOOR(x) -> Digunakan untuk pembulatan ke bawah dari nilai decimal x.

```
SQL> SELECT FLOOR(5.1)
2 FROM dual;
FLOOR(5.1)
```

```
-----
5
```

MOD()

MOD(x,y) -> Digunakan untuk mencari sisa bagi antara nilai x dengan y.

```
SQL> SELECT MOD(7,2)
```

```
2 FROM dual;
```

```
MOD(7,2)
```

```
-----
```

```
1
```

```
SQL> SELECT nama_pegawai, gaji, MOD(gaji,1000000)
```

```
2 FROM pegawai;
```

NAMA_PEGAWAI	GAJI MOD	(GAJI,1000000)
-----	-----	-----
MEUTIA JOVI MAHARANI	22500000	500000
BUDI HARTADI	10500000	500000
RULLY SIANIPAR	10000000	0

ROUND()

ROUND(x [,y]) -> Digunakan untuk pembulatan dari nilai x ke decimal y kebelakang koma.

```
SQL> SELECT ROUND(5.71), ROUND(10.65634,2)
```

```
2 FROM dual;
```

```
ROUND(5.71)    ROUND(10.65634,2)
```

```
-----
```

```
6
```

```
-----
```

```
10.66
```

TRUNC()

TRUNC(x [,y]) -> Digunakan untuk memotong nilai x menjadi y decimal tanpa dilakukan pembulatan.

```
SQL> SELECT TRUNC(5.71), TRUNC(10.65634,2)
```

```
2 FROM dual;
```

```
TRUNC(5.71)    TRUNC(10.65634,2)
```

```
-----
```

```
5
```

```
-----
```

```
10.65
```


3. DATE FUNCTION

SYSDATE dan CURRENT_TIMESTAMP

Digunakan untuk menampilkan tanggal dan waktu system saat ini.

```
SQL> SELECT sysdate  
2 FROM dual;
```

SYSDATE

05-OCT-12

```
SQL> SELECT current_timestamp  
2 FROM dual;
```

CURRENT_TIMESTAMP

05-OCT-12 04.14.45.131000 PM +07:00

MONTHS_BETWEEN

MONTHS_BETWEEN(x,y) -> Digunakan untuk mencari jumlah bulan antara tanggal y dan tanggal x.

```
SQL> SELECT months_between('09-NOV-09','09-OCT-08')  
2 FROM dual;
```

Fungsi-Fungsi Bulan lainnya

- **ADD_MONTHS**
- **LAST_DAY**
- **NEXT_DAY**
- **ROUND**
- **TRUNC**

4. GENERAL FUNCTION

NVL()

Digunakan untuk menangani nilai null.

```
SQL> SELECT nama_pegawai,  
2 gaji,  
3 pct_komisi,
```

```

4 gaji + (pct_komisi*gaji) as "Total"
5 FROM pegawai;

```

```

SQL> SELECT nama_pegawai,
2 gaji,
3 pct_komisi,
4 gaji + (NVL (pct_komisi,0)*gaji) as "Total"
5 FROM pegawai;

```

NULLIF(x,y)

Digunakan untuk membandingkan nilai x dan y. Jika nilai x sama dengan nilai y maka akan menghasilkan nilai Null.

```

SQL> SELECT kode_brg,
2 LENGTH(nama_brg) AS x,
3 LENGTH(keterangan) AS y,
4 NULLIF(LENGTH(nama_brg),LENGTH(keterangan))
5 AS "NULLIF(x,y)"
6 FROM barang;

```

KODE_BRG	X	Y	NULLIF(x,y)
10011	12	18	12
10012	23	29	23
10013	20	27	20

CASE

Digunakan untuk percabangan seperti IF-THEN-ELSE

```

SQL> SELECT nama_pegawai, kode_bag, gaji as "Gaji",
2 CASE kode_bag WHEN 31 THEN gaji*1.1
3 WHEN 41 THEN gaji*1.15
4 ELSE gaji
5 END as "Gaji Baru"
6 FROM pegawai;

```

GROUP FUNCTION

GROUP FUNCTION akan memproses banyak data dan memberikan satu hasil. GROUP FUNCTION sering disebut juga **AGGREGATE FUNCTION**.

1. AVG

AVG(x) di gunakan untuk menghitung nilai rata-rata dari kelompok data x.

```
SQL> SELECT AVG(harga_jual) AS "Rata2 Harga Jual"
2 FROM barang;
```

```
SQL> SELECT AVG(gaji) AS "Rata-Rata Gaji"
2 From pegawai;
```

2. COUNT

COUNT(x) digunakan untuk menghitung jumlah data dari kelompok data x.

```
SQL> SELECT COUNT(id_pegawai) AS "Jlh Pegawai"
2 FROM pegawai;
```

Jlh Pegawai

24

Jika ingin menghitung jumlah data sebaiknya gunakan kolom yang tidak ada nilai null.

```
SQL> SELECT COUNT(pct_komisi) AS "Jlh Pegawai"
2 FROM pegawai;
```

Jlh Pegawai

5

Atau bisa menggunakan kolom ROWID.

```
SQL> SELECT COUNT(rowid)
2 FROM pegawai;
```

COUNT(ROWID)

24

3. MAX dan MIN

Max(x) Digunakan untuk mencari nilai paling tinggi dari kelompok data x dan MIN(x) digunakan untuk mencari nilai yang paling rendah dari kelompok data x.

```
SQL> SELECT MAX(harga_jual) AS "Harga Tertinggi",  
2 MIN(harga_jual) AS "Harga Terendah"  
3 FROM barang;
```

Harga Tertinggi	Harga Terendah
-----------------	----------------

-----	-----
5500000	139000

```
SQL> SELECT MAX(tgl_masuk) AS "Baru",  
2 MIN(tgl_masuk) AS "Lama"  
3 FROM pegawai;
```

Baru	Lama
------	------

-----	-----
30-NOV-02	03-MAR-99

4. SUM

SUM(x) digunakan untuk menghitung jumlah total dari data-data yang ada di kelompok data x.

```
SQL> SELECT SUM(total)  
2 FROM hd_sales;
```

```
SQL> SELECT SUM(gaji)  
2 FROM pegawai;
```

Mengelompokan data dengan GROUP BY

GROUP By digunakan untuk mengelompokan data hasil perhitungan dengan Fungsi Agregasi.

```
SQL> SELECT kode_bag, SUM(gaji) as "TOTAL"  
2 FROM pegawai  
3 GROUP By kode_bag;
```

```
SQL> SELECT kode_bag, jabatan,  
2 SUM(gaji) as "Total Gaji"
```

```
3 FROM pegawai
4 GROUP BY kode_bag, jabatan
5 ORDER BY kode_bag;
```

```
SQL> SELECT kode_bag, jabatan,
2 COUNT(id_pegawai) AS "Jlh Pegawai",
3 SUM(gaji) as "Total Gaji"
4 FROM pegawai
5 GROUP BY kode_bag, jabatan
6 ORDER BY kode_bag;
```

Menyeleksi Hasil GROUP BY dengan HAVING

Pada Single Function digunakan klause WHERE untuk melakukan seleksi data berdasarkan kriteria. Pada Prinsipnya fungsi Having sama dengan where. HAVING digunakan untuk menyeleksi data hasil dari GROUP FUNCTION.

Sebagai contoh, ingin ditampilkan bagian yang rata-rata gaji pegawainya di atas 7000000.

```
SQL> SELECT kode_bag as "Bagian",
2 AVG(gaji) as "Rata2"
3 FROM pegawai
4 HAVING AVG(gaji)>7000000
5 GROUP BY kode_bag;
```

Jika kita menyeleksinya menggunakan WHERE maka akan terjadi ERROR.

```
SQL> SELECT kode_bag as "Bagian",
2 AVG(gaji) as "Rata2"
3 FROM pegawai
4 WHERE AVG(gaji)>7000000
5 GROUP BY kode_bag;
WHERE AVG(gaji)>7000000
*
```

ERROR at line 4:
ORA-00934: group function is not allowed here

Sebenarnya klausa WHERE tetap bisa digunakan bersama dengan HAVING, tetapi klausa WHERE tetap digunakan untuk non-group function.

Sebagai contoh, akan menampilkan bagian yang rata-rata gaji pegawainya di atas 4000000 tetapi pegawai yang jabatannya VICE President (VP) tidak ikut di kalkulasi.

```
SQL> SELECT kode_bag as "Bagian",  
2 AVG(gaji) as "Rata2"  
3 FROM pegawai  
4 WHERE jabatan NOT LIKE 'VP%'  
5 GROUP BY kode_bag  
6 HAVING AVG(gaji)>4000000  
7 ORDER BY kode_bag;
```

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta





MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 6 SUB QUERY

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

06

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam sub query

Kompetensi

Mampu memahami perintah-perintah dalam sub query

SUB QUERY

1. Pengertian SubQuery

Subquery merupakan “**Query di dalam query**” atau perintah SELECT yang berada dalam perintah SELECT lainnya.

Struktur Penulisan Subquery :

```
SELECT      kolom...
FROM table
WHERE      kondisi (SELECT      kolom
                        FROM table
                        WHERE      kondisi);
```

Contoh solusi tanpa subquery

```
SQL> SELECT nama_pegawai,
2 gaji
3 FROM pegawai
4 WHERE nama_pegawai='ADRIAN';
```

NAMA_PEGAWAI	GAJI
-----	-----
ADRIAN	4000000

Tampilkan pegawai yang gajinya lebih kecil dari 4000000

```
SQL> SELECT nama_pegawai,
2 gaji
3 FROM pegawai
4 WHERE gaji<4000000;
```

Solusi dengan subquery

```
SQL> SELECT nama_pegawai,
2 gaji
3 FROM pegawai
4 WHERE gaji<
5 (SELECT gaji
6 FROM pegawai
7 WHERE nama_pegawai='ADRIAN');
```


Note :

- Subquery ditulis dalam tanda kurung.
- Letakkan Subquery di sisi kanan tanda pembanding
- Gunakan tanda pembanding tunggal untuk single-row subquery dan tanda pembanding jamak untuk multi-row subquery

Jenis-jenis SubQuery

- **Single-row subquery**
- **Multi-row subquery**
- **Multi-column subquery**
- **Corelated subquery**
- **Inline View**

2. Single Row SubQuery

Single-row subquery adalah subquery yang hanya menghasilkan satu baris dan satu kolom data atau data tunggal.

Tanda pembanding yang digunakan pada single-row subquery adalah tanda pembanding tunggal seperti =,<,>,<=,>=,<>.

Contoh : Tampilkan pegawai yang satu bagian dengan 'ASFIANTI' dan gajinya lebih kecil dari pegawai 'RIDWAN SANUSI'.

```
SQL> SELECT nama_pegawai,  
2 gaji,  
3 kode_bag  
4 FROM pegawai  
5 WHERE kode_bag=(SELECT kode_bag  
6 FROM pegawai  
7 WHERE nama_pegawai='ASFIANTI')  
8 AND  
9 gaji < (SELECT gaji  
10 FROM pegawai  
11 WHERE nama_pegawai='RIDWAN SANUSI')  
12 ORDER BY kode_bag, gaji;
```

- **Group Function pada Subquery**

Pada Subquery bisa menggunakan group function seperti AVG, MAX, MIN.

Contoh : Tampilkan pegawai yang gajinya berada di atas rata-rata.

```
SQL> SELECT nama_pegawai,  
2 gaji  
3 FROM pegawai  
4 WHERE gaji > (SELECT avg(gaji) FROM pegawai);
```

Pegawai yang gajinya paling rendah.

```
SQL> SELECT nama_pegawai,  
2 gaji  
3 FROM pegawai  
4 WHERE gaji = (SELECT min(gaji) FROM pegawai);
```

- **Klausula Having pada Subquery**

Contoh kasus, ingin menampilkan bagian yang rata-rata gajinya di atas gaji pegawai 'ASFIANTI'.

```
SQL> SELECT kode_bag, avg(gaji)  
2 FROM pegawai  
3 GROUP BY kode_bag  
4 HAVING avg(gaji) > (SELECT gaji  
5 FROM pegawai  
6 WHERE nama_pegawai='ASFIANTI');
```

- **Klausula ORDER BY pada Subquery**

Data yang di tampilkan ingin di urutkan.

Pegawai yang gajinya paling rendah.

```
SQL> SELECT nama_pegawai,  
2 gaji  
3 FROM pegawai  
4 WHERE gaji = (SELECT min(gaji) FROM pegawai);  
5 ORDER BY kode_bag;
```

3. Multi-Row SubQuery

Multi-row subquery adalah subquery yang menghasilkan lebih dari satu row. Tanda pembandingan yang digunakan pada multi-row adalah tanda pembandingan jamak yaitu **IN**, **ANY**, dan **ALL**.

```
SQL> SELECT id_pegawai, nama_pegawai, gaji
2 FROM pegawai
3 WHERE gaji = (SELECT min(gaji)
4 FROM pegawai
5 GROUP BY kode_bag);
```

```
WHERE gaji = (SELECT min(gaji)
                *
```

ERROR at line 3:

ORA-01427: single-row subquery returns more than one row

Perintah di atas error, kenapa? Karena hasil dari subquery-nya lebih dari satu data.

Coba kita cek :

```
SQL> SELECT min(gaji)
2 FROM pegawai
3 GROUP BY kode_bag;
```

MIN(GAJI)

2000000
4000000
11500000
1400000
10500000

Untuk mengatasi hal tersebut, kita harus mengganti tanda pembandingan dengan pembandingan jamak seperti **IN**.

- Operator IN

Operator IN merupakan tanda pembandingan jamak, kita bisa membuat sekumpulan nilai atau ekspresi sebagai pembandingan.

Dari contoh sebelumnya bermaksud mencari pegawai yang gajinya sama dengan minimal gaji yang ada di setiap bagian. Jika menggunakan operator IN menjadi sebagai berikut.

```
SQL> SELECT id_pegawai, nama_pegawai, gaji, kode_bag
2 FROM pegawai
3 WHERE gaji IN (SELECT min(gaji)
4               FROM pegawai
5               GROUP BY kode_bag);
```

- Operator ANY

Operator ANY digunakan untuk membandingkan data yang dicari dengan sekumpulan data yang ada. Pada saat digunakan operator ANY harus dipasangkan dengan salah satu dari operator <, >, atau = dengan ketentuan sebagai berikut :

- o = ANY ~> mempunyai karakteristik sama dengan Operator IN, yaitu semua data dibandingkan dengan elemen dari hasil subquery.

```
SQL> SELECT id_pegawai, nama_pegawai, gaji, kode_bag
2 FROM pegawai
3 WHERE gaji =ANY (SELECT min(gaji)
4                 FROM pegawai
5                 GROUP BY kode_bag);
```

- o <ANY ~> Menampilkan data di out query yang lainnya lebih kecil dari data yang paling besar dari hasil subquery

```
SQL> SELECT id_pegawai
2 nama_pegawai,
3 gaji,
4 kode_bag
5 FROM pegawai
6 WHERE gaji <ANY (SELECT gaji
7                 FROM pegawai
8                 WHERE kode_bag=31);
```

- o >ANY ~> Menampilkan data di out query yang lainnya lebih besar dari data yang paling kecil dari hasil subquery

```
SQL> SELECT id_pegawai
2  nama_pegawai,
3  gaji,
4  kode_bag
5  FROM pegawai
6  WHERE gaji >ANY (SELECT gaji
7                  FROM pegawai
8                  WHERE kode_bag=31);
```

- Operator ALL

Digunakan untuk membandingkan data yang dicari dengan sekumpulan data yang ada. Pada saat digunakan operator ALL. Harus dipasangkan dengan salah satu dari operator < dan > dengan ketentuan sebagai berikut :

- o **<ALL ~>** Menampilkan data di out query yang nilainya lebih kecil dari data yang paling kecil dari hasil subquery.

```
SQL> SELECT id_pegawai,
2  nama_pegawai,
3  gaji,
4  kode_bag
5  FROM pegawai
6  WHERE gaji < ALL (SELECT gaji
7                  FROM pegawai
8                  WHERE kode_bag=31);
```

- o **>ALL ~>** Menampilkan data di out query yang nilainya lebih besar dari data yang paling besar dari hasil subquery.

```
SQL> SELECT id_pegawai,
2  nama_pegawai,
3  gaji,
4  kode_bag
5  FROM pegawai
6  WHERE gaji < ALL (SELECT gaji
7                  FROM pegawai
```

4. Multi Coloumn SubQuery

Multi Coloumn Subquery adalah subquery yang menghasilkan lebih dari satu kolom. Jumlah dan posisi kolom dari outer query yang akan dibandingkan harus sama dengan jumlah dan posisi kolom pada subquery.

Struktur penulisan multi-coloumn subquery adalah sebagai berikut :

```
SELECT kolom1, kolom2 ...
FROM table
WHERE (kolomA, kolomB)
      IN
      (SELECT kolom1, kolom2)
      FROM table
      WHERE kondisi);
```

Jumlah kolom pada klausa WHERE di out query harus sama dengan jumlah kolom pada subquery.

Contoh kasus : Menampilkan pegawai yang gajinya paling kecil di setiap bagian.

```
SQL> SELECT id_pegawai,
2  nama_pegawai,
3  gaji,
4  kode_bag
5 FROM pegawai
6 WHERE (kode_bag, gaji) IN (SELECT kode_bag, min(gaji)
7                           FROM pegawai
8                           GROUP by kode_bag);
```

5. Correlated SubQuery

Pada Correlated subquery, outer query akan diproses lebih dulu kemudian hasilnya akan dijadikan rujukan untuk memproses query.

Struktur perintah Correlated subquery :

```
SELECT kolom1, kolom2 ...
FROM table table_alias
WHERE kondisi (SELECT kolom1
               FROM table_alias
               WHERE kolom=table_alias.kolom);
```

Contoh kasus sama seperti Multi-Column subquery diselesaikan dengan Correlated subquery.

```
SQL> SELECT id_pegawai,  
2 nama_pegawai,  
3 gaji,  
4 kode_bag  
5 FROM pegawai p  
6 WHERE gaji = (SELECT min(gaji)  
7 FROM pegawai  
8 WHERE kode_bag=p.kode_bag);
```

Contoh lain, perintah menampilkan pegawai yang gajinya di bawah rata-rata perbagian.

```
SQL> SET PAGESIZE 20  
SQL> SELECT id_pegawai,  
2 nama_pegawai,  
3 gaji,  
4 kode_bag  
5 FROM pegawai p  
6 WHERE gaji < (SELECT avg(gaji)  
7 FROM pegawai  
8 WHERE kode_bag=p.kode_bag)  
9 ORDER BY kode_bag;
```

Operator EXISTS

Operator Exists digunakan untuk memeriksa apakah suatu subquery menghasilkan data atau tidak. Jika suatu subquery menghasilkan data, operator EXISTS akan mengembalikan nilai TRUE sebaliknya akan mengembalikan nilai FALSE.

Contoh penggunaan EXISTS misalnya kita ingin memeriksa mana saja pegawai yang mempunyai bawahan, yaitu dengan memeriksa apakah id_pegawainya ada tercatat di kolom manager_id.

```
SQL> SELECT id_pegawai, nama_pegawai  
2 FROM pegawai outer  
3 WHERE EXISTS (SELECT 'x'  
4 FROM pegawai  
5 WHERE manager_id = outer.id_pegawai);
```

Sebaliknya jika ingin menampilkan pegawai yg tidak punya bawahan gunakan **NOT EXISTS**

```
SQL> SELECT id_pegawai, nama_pegawai  
2 FROM pegawai outer  
3 WHERE NOT EXISTS (SELECT 'x'  
4 FROM pegawai  
5 WHERE manager_id = outer.id_pegawai);
```

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 7 INDEX DAN VIEW

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

07

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam index dan view

Kompetensi

Mampu memahami perintah-perintah dalam index dan view

INDEX

Index adalah Objek database yang berfungsi untuk mempercepat pencarian data.

Secara umum, kita harus menggunakan index untuk menampilkan sebagian kecil data dari jumlah yang sangat banyak pada suatu table.

Di Oracle terdapat 2 jenis Index berdasarkan struktur penyimpanannya

1. **B-Tree Index** – digunakan untuk membuat index berdasarkan kolom yang unique untuk setiap baris datanya seperti id_pegawai.
2. **Bitmap Index** – digunakan untuk kolom tingkat keragamannya rendah.

Buatlah index berdasarkan kolom yang sering digunakan sebagai dasar pencarian.

Membuat B-Tree Index

Oracle membuat otomatis B-Tree index ketika suatu kolom didefinisikan dengan constraint sebagai Primary Key atau Unique.

B-Tree Index dibuat dengan perintah CREATE INDEX, dengan aturan penulisan seperti berikut:

***CREATE [UNIQUE] INDEX nama_index ON nama_table(kolom,
[kolom,...])***

Misal kita sering melakukan query pencarian data pegawai berdasarkan nama pegawai selain dengan id pegawai, maka sebaiknya kita membuat index berdasarkan kolom nama_pegawai pada table pegawai.

```
SQL> CREATE INDEX nama_peg_idx
```

```
2 ON pegawai(nama_pegawai);
```

Index created.

```
SQL> CREATE INDEX bag_nama_peg_idx
```

```
2 ON pegawai(kode_bag, id_pegawai);
```

Membuat Bitmap Index

Bitmap index digunakan untuk membuat index berdasarkan kolom yang tingkat keragamannya rendah. Misalkan kolom credit_rating yang datanya hanya berupa BAIK, SEDANG, dan BURUK.

Bitmat Index dibuat dengan perintah CREATE BITMAT INDEX.

Membuat bitmap index berdasarkan kolom `credit_rating` pada table pelanggan.

```
SQL> CREATE BITMAP INDEX bi_credit_rating  
2 ON pelanggan(credit_rating);
```

Informasi tentang index bisa dilihat dari dictionary `user_ind_coloumn` atau `user_indexes`.

```
SQL> SELECT index_name, table_name  
2 FROM user_indexes  
3 WHERE TABLE_NAME IN('PEGAWAI','PELANGGAN');
```

Menghapus Index

Untuk menghapus index gunakan perintah `DROP INDEX`.

```
SQL> DROP INDEX NAMA_PEG_IDX;
```

VIEW

View merupakan pendefinisian query yang hanya mengakses ke satu atau beberapa table. Lebih sederhana, view bisa saja disebut sebagai **perintah SELECT yang diberi nama**.

- Kita sering melakukan perintah query seperti ini :

```
SELECT id_pegawai, nama_pegawai, nama_bag  
FROM pegawai, bagian  
WHERE pegawai.kode_bag = bagian.kode_bag;
```
- Agar memudahkan kita setiap kali ingin melakukan query tersebut, sebaiknya kita buat view. View dibuat dengan perintah **CREATE VIEW**. Cara membuat VIEW query di atas adalah sebagai berikut :

```
SQL> CREATE VIEW v_peg_bag  
2 AS  
3 SELECT id_pegawai, nama_pegawai, nama_bag  
4 FROM pegawai, bagian  
5 WHERE pegawai.kode_bag=bagian.kode_bag;
```

View created.

- Selanjutnya kita tinggal melakukan query ke view v_peg_bag.

```
SQL> SELECT * FROM v_peg_bag;
```

Membuat dan Menggunakan View

View dibuat dengan perintah CREATE VIEW yang aturan penulisannya adalah sebagai berikut :

CREATE [OR REPLACE] VIEW [FORCE | NOFORCE] nama_view AS subquery

Ket :

- OR REPLACE berarti jika nama view sudah pernah ada akan ditimpa.
- FORCE berarti view bisa dibuat walaupun table yang disebutkan dalam subquery belum dibuat.
- NO FORCE berarti view tidak bisa dibuat jika tabel yang disebutkan dalam subquery belum dibuat.

Berdasarkan kompleksitas subquery, view dibagi dua yaitu simple-view dan complex-view.

- Simple-View, view yang melakukan query dalam satu table dan tidak menggunakan group function.
- Complex-view, view yang subquerynya terdiri dari dua table atau lebih serta menggunakan DISTINCT atau GROUP BY.

Membuat Simple-View

Simple-view adalah view yang berisi query yang hanya mengakses ke satu table dan tidak melibatkan group function atau ekspresi kompleks.

```
SQL> CREATE OR REPLACE VIEW v_brg_murah
2 AS
3 SELECT kode_brg, nama_brg, harga_jual
4 FROM barang
5 WHERE harga_jual <300000;
```

```
SQL> CREATE OR REPLACE VIEW v_peg31
2 AS
3 SELECT id_pegawai,
4 nama_pegawai,
5 kode_bag,
6 gaji
```

```
7 FROM pegawai
8 WHERE kode_bag=31;
```

Setelah membuat view, kita bisa melakukan query ke view tersebut sebagaimana kita melakukan query terhadap table.

```
SQL> SELECT * from v_brg_murah where harga_jual<200000;
```

INSERT, UPDATE, dan DELETE melalui VIEW

View hanya berupa perintah query, view tidak menyimpan data. Data selalu disimpan di table. Kita hanya bisa melakukan manipulasi data di table melalui simple-view, complex-view tidak selalu bisa digunakan untuk memanipulasi data.

Untuk memperlihatkan proses INSERT, UPDATE, dan DELETE data melalui view ikuti tahapan berikut :

1. Masukan data melalui View.

```
SQL> INSERT INTO v_brg_murah
2 VALUES (90000,'EXTERNAL DISK',150000);
```

2. Periksa data melalui View.

```
SQL> SELECT * from v_brg_murah
2 WHERE kode_brg=90000;
```

KODE_BRG	NAMA_BRG	HARGA_JUAL
90000	EXTERNAL DISK	150000

3. Periksa data di table.

```
SQL> SELECT kode_brg, nama_brg, harga_jual
2 FROM barang
3 WHERE kode_brg=90000;
```

KODE_BRG	NAMA_BRG	HARGA_JUAL
90000	EXTERNAL DISK	150000

4. v_brg_murah didefinisikan untuk menampilkan barang yang harganya di bawah 300.000. Bisakah v_brg_murah digunakan untuk memasukan data barang yang harganya di atas 300.000??

```
5. SQL> INSERT INTO v_brg_murah
2 VALUES(70000,'MONITOR SAMSUNG 15 inch', 8750000);
```

Ternya bisa dilakukan proses INSERT, tetapi jika di periksa data tersebut tidak ada di view maupun di table. Karena kita memberi CONSTRAINT pada v_brg_murah yaitu harga di bawah 300.000.

```
SQL> SELECT * from v_brg_murah
2 WHERE kode_brg=70000;
```

no rows selected

View dengan Constraint CHECK OPTION

Cara Membuat view dengan Constraint CHECK OPTION adalah sbb :

```
SQL> CREATE OR REPLACE VIEW v_brg_murah
2 AS
3 SELECT kode_brg, nama_brg, harga_jual
4 FROM barang
5 WHERE harga_jual <300000
6 WITH CHECK OPTION CONSTRAINT ck_hrg_300000;
```

Coba sekarang masukkan data barang yang harganya lebih besar dari 300.000

```
SQL> INSERT INTO v_brg_murah
2 VALUES (90000,'EXTERNAL DISK',500000);
INSERT INTO v_brg_murah
*
```

ERROR at line 1:

ORA-01402: view WITH CHECK OPTION where-clause
Violation

View dengan Constraint READ ONLY

Sebagai contoh, buat view v_peg31 menjadi read only.

```
SQL> SELECT id_pegawai,
2
SQL> CREATE OR REPLACE VIEW v_peg31
2 AS
3 SELECT id_pegawai,
```

```
4  nama_pegawai,  
5  kode_bag,  
6  gaji  
7 FROM pegawai  
8 WHERE kode_bag=31  
9 WITH READ ONLY CONSTRAINT ck_ready_only_peg31;
```

View yang sudah di create dengan Constrain READ ONLY maka view tersebut tidak bisa di insert, update dan delete.

Membuat Complex View

Complex View berisi query dengan karakteristik :

- Menampilkan data dari dua table atau lebih.
- Data dikeompokkan dengan GROUP BY atau DISTINCT
- Menggunakan Function SQL.

Complex View tidak selalu bisa digunakan untuk melakukan insert, update, dan delete data.

Berikut ini kita buat view v_peg_rata2_gaji yang digunakan untuk menampilkan pegawai yang gajinya lebih besar dari rata-rata gaji di bagiannya.

```
SQL> CREATE OR REPLACE VIEW v_peg_rata2_gaji  
2 AS  
3 SELECT a.id_pegawai, a.nama_pegawai, a.gaji, b.avg_gaji  
4 FROM pegawai a, (SELECT kode_bag, AVG(gaji) avg_gaji  
5                 FROM pegawai  
6                 GROUP BY kode_bag) b  
7 WHERE a.kode_bag = b.kode_bag  
8 AND a.gaji > b.avg_gaji;
```

Menghapus View

Untuk menghapus view digunakan perintah DROP VIEW. Contoh menghapus view v_peg31.

```
SQL> DROP VIEW v_peg31;
```

Soal Quis

1. Perintah yang digunakan dalam kelompok Data Manipulation Language adalah
2. User bisa melakukan akses ke beberapa database, pernyataan tersebut benar atau salah ? Mengapa
3. Untuk berkomunikasi dengan database kita harus menggunakan bahasa yang dipahami oleh database. Bahasa yang digunakan untuk berkomunikasi dengan database disebut ?
4. Sebutkan tools interface yang dapat digunakan untuk berinteraksi dengan database ?
5. Untuk menampilkan daftar user yang sudah terdaftar di dalam suatu database dilakukan dengan perintah ?
6. Perintah yang digunakan untuk menampilkan data dari tabel pegawai adalah ?
7. Anda menggunakan SQL *Plus dan akan mengeksekusi beberapa fungsi matematika. Tabel apa yang anda akan gunakan untuk melakukan kalkulasi matematika dari ekspresi atau nilai yang bukan diambil dari tabel di dalam database ?
8. Anda ingin menulis Query untuk menampilkan data pegawai dengan memilih data tertentu berdasarkan kriteria. Pada klausa apa kriteria tersebut anda definisikan
9. Anda ingin menampilkan pegawai yang ada di bagian 30 dan namanya diawali oleh karakter 'A', perintah yang digunakan adalah
10. Karakter yang bisa digunakan sebagai wildcard atau pembentuk pola pada operator LIKE adalah
11. Untuk mengkombinasikan beberapa kolom atau kolom dengan suatu text menjadi satu kolom output menggunakan perintah ?
12. Perintah Save point digunakan untuk ?
13. Perintah yang digunakan untuk menghapus tabel ?
14. Sebutkan logical condition yang bisa kita gunakan dalam proses query ?
15. Perintah Where digunakan untuk ?

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 8 JOIN

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

08

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam join

Kompetensi

Mampu memahami perintah-perintah dalam join

JOIN

1. Pengertian JOIN

Selama ini, dengan perintah SELECT yang kita lakukan hanya menampilkan data dari satu table, padahal dalam kondisi sebenarnya kerap kali kita harus menampilkan data dari **beberapa table** dalam satu perintah SELECT agar data yang ditampilkan lebih informative.

Jika ingin menampilkan informasi di mana data yang diperlukan harus diambil dari beberapa table maka kita harus melakukan **JOIN**. Join antara dua table memerlukan kondisi join yang digunakan untuk menghubungkan data dari satu table ke data yang ada di table lain. Kondisi join didefinisikan pada klausa **WHERE**.

Contoh : Kita bisa memperoleh informasi tentang order_id, tanggal_order, total Order, dan id_ pelanggan dengan perintah berikut :

```
SQL> SELECT order_id,  
2   tgl_order,  
3   id_plg,  
4   total  
5 FROM hd_sales;
```

ORDER_ID	TGL_ORDER	ID_PLG	TOTAL
-----	-----	-----	-----
97	28-AUG-06	201	15090000
98	31-AUG-06	202	850000
99	31-AUG-06	203	1420000
100	31-AUG-06	204	11750000
101	31-AUG-06	205	2482900

Dari data diatas hanya di tampilkan **ID_PLG**, nama pelanggannya tidak ditampilkan karena nama pelanggan hanya ada di table pelanggan sehingga data di atas kurang informative. Untuk menampilkan nama pelanggan kita perlu melakukan **JOIN** antara table hd_sales dengan table pelanggan.

```
SQL> SELECT hd_sales.tgl_order,  
2   hd_sales.order_id,
```

```

3  pelanggan.nama_plg,
4  hd_sales.total
5 FROM hd_sales, pelanggan
6 WHERE hd_sales.id_plg = pelanggan.id_plg;

```

Dengan Query diatas, maka nama pelanggan dari table pelanggan bisa di tampilkan.

Beberapa hal yang harus diperhatikan dalam melakukan join table adalah :

- Pada klausa SELECT, untuk setiap kolom yang akan ditampilkan sebaiknya diawali dengan nama table asal kolom dengan dibatasi oleh tanda titik (.).
- Pada klausa WHERE yang digunakan untuk mendefinisikan kondisi nama table harus disebutkan sebelum nama kolom penghubung, jika tidak akan menyebabkan error.

2. Kolom Alias

Untuk mempersingkat penulisan, biasanya nama table dibuat **ALIAS** yang lebih pendek dari nama table aslinya. Nama alias suatu table didefinisikan pada klausa FROM setelah nama table aslinya. Misalkan table hd_sales akan dialiaskan sebagai **h** dan table pelanggan akan dialiaskan sebagai **p** maka pendefinisian alias di klausa FROM adalah sebagai berikut :

```

SQL> SELECT h.tgl_order,
2      h.order_id,
3      p.nama_plg,
4      h.total
5 FROM hd_sales h, pelanggan p
6 WHERE hd_sales.id_plg = pelanggan.id_plg;

```

3. Cartesian Product

Cartesian Product adalah suatu kondisi jika suatu join tanpa ada kondisi join, di mana hasil join merupakan kombinasi antara data yang ada di kedua table. Jumlah data hasil Cartesian Product adalah hasil perkalian antara jumlah data di table pertama dengan jumlah data di table kedua.

```

SQL> SELECT h.tgl_order,
2      h.order_id,
3      p.nama_plg,
4      h.total

```

5 FROM hd_sales h, pelanggan p

4. Kondisi Join dan Jenis Join

Berdasarkan tanda pembanding yang digunakan pada kondisi join di klausa WHERE join dibagi dalam dua bagian yaitu :

- **Equi Join** -> Join antara dua table yang kondisi joinnya menggunakan tanda sama dengan (=). **Contoh join sebelumnya selalu menggunakan Equi Join.**
- **Non-equi Join** -> Join antara dua table yang kondisi joinnya menggunakan selain tanda sama dengan seperti <,>,BETWEEN, dan tanda pembanding lainnya.

Sedangkan berdasarkan dari data yang dihasilkan jenis join ada tiga, yaitu :

- **Inner Join** -> Join yang menghasilkan data yang bersesuaian atau data yang memenuhi kondisi join saja.
- **Outer Join** -> Join yang hasilnya merupakan gabungan data yang bersesuaian atau data yang memenuhi kondisi join dan data yang tidak punya pasangan di table lawannya.
 - **Right Join**
 - **Left Join**
- **Self Join** -> Join yang dihasilkan yang dilakukan antar kolom dalam satu table.

5. Non Equi Join

Berikut ini adalah contoh pemakaian non-equi join dengan menggunakan pembading BETWEEN. Dari data pegawai yang ada kita ingin mengelompokkan pegawai dengan level tertentu berdasarkan gaji pegawai. Tabel yang akan digunakan adalah table GRADE dan table PEGAWAI.

Untuk mengelompokkan seorang pegawai masuk grade mana kita tinggal melakukan join antar kolom gaji pada table pegawai menggunakan pembading BETWEEN ke kolom bts_bawah dan bts_atas pada table grade.

Perintah joinnya adalah sebagai berikut :

```
SQL> SELECT p.nama_pegawai,  
2   p.gaji,  
3   g.kode_grade  
4 FROM pegawai p, grade g  
5 WHERE p.gaji BETWEEN g.bts_bawah AND g.bts_atas;
```

NAMA_PEGAWAI	GAJI	KODE_GRADE
-----	-----	-----
HERU KUTANTO	1400000	A
ASWIR MATONDANG	1500000	A
AMAURA	1950000	A
DHILA LESTARI	1950000	A

6. Outer Join

Outer Join akan menghasilkan data yang memenuhi kondisi join baik yang ada pasangannya ataupun tidak ada pasangannya di table lawan joinnya.

Jika ingin menampilkan semua data yang ada di table yang sisi kiri baik yang ada pasangannya ataupun tidak Maka join ini di sebut **LEFT OUTER JOIN** sebaliknya disebut **RIGHT OUTER JOIN**.

- **Left Outer Join**

Di sini kita akan menampilkan semua pegawai baik yang sudah ditempatkan pada bagian tertentu maupun yang belum ditempatkan. Perhatikan letak tanda (+) dan posisi table pada kondisi join.

```
SQL> SELECT p.id_pegawai,
2   p.nama_pegawai,
3   b.nama_bag
4 FROM pegawai p, bagian b
5 WHERE p.kode_bag = b.kode_bag (+)
6 ORDER BY b.nama_bag;
```

ID_PEGAWAI	NAMA_PEGAWAI	NAMA_BAG
-----	-----	-----
1001	MEUTIA JOVI MAHARANI	ADMINISTRATION
1005	NOVI SETIAWATI	ADMINISTRATION
1004	EDWIN ASRUL	FINANCE
1025	ASFianti	OPERATIONS
1021	SUHARDIATNO	----
1022	AHMAD KOSASIH	----

- **Right Outer Join**

Contoh untuk Right Outer join kita akan menampilkan semua bagian baik yang sudah ada pegawainya maupun yang belum ada pegawainya. Perhatikan letak tanda (+) dan posisi table pada kondisi join.

```
SQL> SELECT p.id_pegawai,
2    p.nama_pegawai,
3    b.nama_bag
4 FROM pegawai p, bagian b
5 WHERE p.kode_bag (+) = b.kode_bag
6 ORDER BY b.nama_bag;
```

ID_PEGAWAI	NAMA_PEGAWAI	NAMA_BAG
-----	-----	-----
1012	SUSI INDIARTI	SALES
1015	ESTI ARVINA	SALES
1003	RULLY SIANIPAR	SALES
1011	RIDWAN SANUSI	SALES
1014	ERTIN	SALES
----	----	SECURITY
		STRATEGIC PLANNING
		TECHNICAL WRITING

- **Full Outer Join**

Hasil dari Full Outer Join pada dasarnya merupakan gabungan dari hasil Left Outer Join dengan Right Outer Join. Untuk menggabungkan kedua perintah tersebut gunakan operator UNION.

Join untuk table pegawai dan bagian adalah sebagai berikut :

```
SQL> SELECT p.id_pegawai,
2    p.nama_pegawai,
3    b.nama_bag
4 FROM pegawai p, bagian b
5 WHERE p.kode_bag = b.kode_bag (+)
6 UNION
7 SELECT p.id_pegawai,
8    p.nama_pegawai,
9    b.nama_bag
```

```

10 FROM pegawai p, bagian b
11 WHERE p.kode_bag (+) = b.kode_bag;

```

7. Self Join

Self Join merupakan join atarkolom dalam satu table. Join dengan nama table yang sama sebenarnya tidak bisa dilakukan. Self Join dilakukan dengan menggunakan table alias. Jadi satu table dibuat menjadi dua buah table alias sehingga seolah-olah join dilakukan dari dua table yang berbeda.

Contoh join kita ambil dari table pegawai, yaitu join antar kolom manger_id dengan kolom id_pegawai. Kedua kolom tersebut membunyai hubungan karena nilai kolom manager id sebenarnya merupakan id_pegawai yang sudah punya bawahan.

```

SQL> SELECT staff.id_pegawai,
2   staff.nama_pegawai,
3   mgr.nama_pegawai AS "Nama Manager"
4 FROM pegawai staff, pegawai mgr
5 WHERE staff.manager_id=mgr.id_pegawai;

```

ID_PEGAWAI	NAMA_PEGAWAI	Nama Manager
1005	NOVI SETIAWATI	MEUTIA JOVI MAHARANI
1004	EDWIN ASRUL	MEUTIA JOVI MAHARANI
1003	RULLY SIANIPAR	MEUTIA JOVI MAHARANI
1002	BUDI HARTADI	MEUTIA JOVI MAHARANI
1010	PURNAMA RIYANTO	BUDI HARTADI

Menggunakan Syntax Join ANSI/SQL

Join yang sudah di bahas sebelumnya merupakan join dari ORACLE. Selain join tersebut, Oracle juga mengadopsi syntax join dari ANSI SQL/92 yang mulai dipakai oleh Oracle sejak Oracle 9i. Join dari Oracle maupun dari ANSI SQL akan menghasilkan data yang sama hanya berbeda dalam cara penulisan dan struktur perintanya.

1. Natural Join

Naral Join bisa dilakukan antardua table dengan ketentuan :

- Ada Kolom yang sama pada kedua kolom
- Kolom yang namanya sama hanya ada satu kolom

- Data yang muncul adalah data yang saling bersesuaian di antara dua table atau inner Join.

Contoh :

```
SQL> SELECT id_pegawai,
2  nama_pegawai,
3  nama_bag
4 FROM pegawai NATURAL JOIN bagian;
```

2. Klausa Using

Klausa Using merupakan pengembangan dari Natural Join, dengan ketentuan sebagai berikut :

- Jika ada lebih dari satu kolom yang namanya sama pada tabel yang akan di join.
- Klausa USING digunakan untuk menentukan nama kolom yang akan digunakan sebagai kondisi join
- Pada nama kolom yang digunakan pada klausa USING tidak boleh diawali dengan nama table.

Contoh :

```
SQL> SELECT id_pegawai,
2  nama_pegawai,
3  nama_bag
4 FROM pegawai USING JOIN bagian;
```

3. Klausa ON

Klausa ON digunakan untuk melakukan join pada dua table walaupun pada kedua table tersebut tidak ada kolom yang namanya sama untuk dijadikan sebagai kondisi join tetapi data pada kolom-kolom tersebut bersesuaian sehingga memungkinkan untuk dilakukan join.

Join antara table Pegawai dan Bagian menggunakan kolom kode_bag.

```
SQL> SELECT p.id_pegawai,
2  p.nama_pegawai,
3  b.nama_bag
4 FROM pegawai p JOIN bagian b
5 ON (p.kode_bag = b.kode_bag)
```


6 ORDER BY p.id_pegawai;

4. Cross Join

Cros Join akan menghasilkan cross product dari kedua table. Jumlah data yang dihasilkan merupakan perkalian jumlah data table pertama dengan table kedua. Ini sama seperti **Cartesian Product**.

```
SQL> SELECT b.nama_bag,  
2 p.nama_pegawai  
3 FROM pegawai p CROSS JOIN bagian b;
```

5. Join Banyak Tabel

Dalam kondisi sebenarnya untuk menghasilkan laporan yang baik kadang kita harus melakukan join dari beberapa table. Berikut ini adalah contoh join empat buah table menggunakan klausa ON untuk menampilkan informasi pembelian barang yang pernah dilakukan oleh Pelanggan.

Tabel yang di gunakan Pelanggan, Hd_Sales, Item, dan Barang.

```
SQL> SELECT p.nama_plg,  
2 h.order_id,  
3 h.total,  
4 b.kode_brg,  
5 b.nama_brg,  
6 b.harga_jual,  
7 i.jlh  
8 FROM pelanggan p JOIN hd_sales h  
9 ON (p.id_plg = h.id_plg)  
10 JOIN item i  
11 ON (h.order_id = i.order_id)  
12 JOIN barang b  
13 ON (b.kode_brg = i.kode_brg)  
14 ORDER BY p.nama_plg, b.kode_brg;
```

Menggunakan syntax oracle :

```
SQL> SELECT p.nama_plg,  
2 h.order_id,
```

```

3      h.total,
4      b.kode_brg,
5      b.nama_brg,
6      b.harga_jual,
7      i.jlh
8 FROM pelanggan p,
9      hd_sales h,
10     item i,
11     barang b
12 WHERE p.id_plg=h.id_plg
13 AND h.order_id = i.order_id
14 AND b.kode_brg = i.kode_brg
15 ORDER BY p.nama_plg, b.kode_brg;

```

Keduanya menghasilkan data yang sama.

6. Outer Join dengan ANSI/SQL

Pada prinsipnya Outer Join dengan ANSI SQL sama persis dengan Outer Join yang dari syntax Oracle.

Ketentuan outer join dengan ANSI SQL adalah sebagai berikut :

- Klausa LEFT OUTER JOIN digunakan untuk membuat left outer join.
- Klausa RIGHT OUTER JOIN digunakan untuk membuat right outer join.
- Klausa FULL OUTER JOIN digunakan untuk membuat full outer join.

LEFT OUTER JOIN

```

SQL> SELECT p.id_pegawai,
2   p.nama_pegawai,
3   b.nama_bag
4 FROM pegawai p LEFT OUTER JOIN bagian b
5 ON (p.kode_bag = b.kode_bag)
6 ORDER BY b.nama_bag;

```

RIGHT OUTER JOIN

```
SQL> SELECT p.id_pegawai,  
2    p.nama_pegawai,  
3    b.nama_bag  
4 FROM pegawai p RIGHT OUTER JOIN bagian b  
5 ON (p.kode_bag = b.kode_bag)  
6 ORDER BY b.nama_bag;
```

FULL OUTER JOIN

```
SQL> SELECT p.id_pegawai,  
2    p.nama_pegawai,  
3    b.nama_bag  
4 FROM pegawai p FULL OUTER JOIN bagian b  
5 ON (p.kode_bag = b.kode_bag)  
6 ORDER BY b.nama_bag;
```

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 9 PL / SQL

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

09

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah dalam PL / SQL

Kompetensi

Mampu memahami perintah-perintah dalam PL / SQL

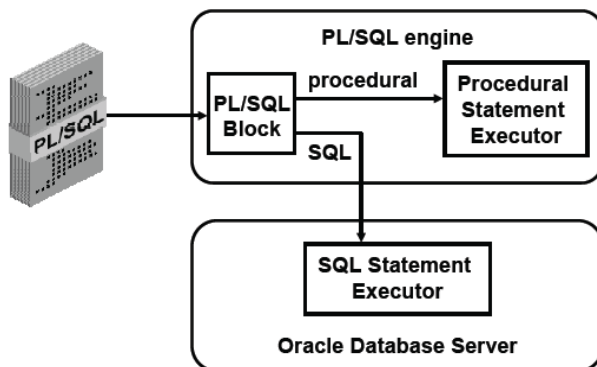
PL / SQL

PL/SQL adalah kepanjangan dari **PROCEDURAL LANGUAGE/STRUCTURE QUERY LANGUAGE** yang mempunyai definisi : suatu blok yang berisi skrip-skrip bahasa procedural.

PL/SQL mewakili perluasan bahasa procedural ke SQL, bahasa akses data standar dari oracle corporation untuk relational database. PL/SQL menyediakan satu struktur block untuk menjalankan suatu kode. Maintenance dari kode yang dibuat lebih mudah dengan mendesain struktur yang baik. Selain itu PL/SQL menyediakan konstruksi procedural seperti:

- Variable, konstanta, dan type data
- Mengatur struktur seperti pernyataan kondisional dan perulangan
- Menggunakan kembali unit program setelah ditulis dan dijalankan di berbagai waktu

PL/SQL Environment



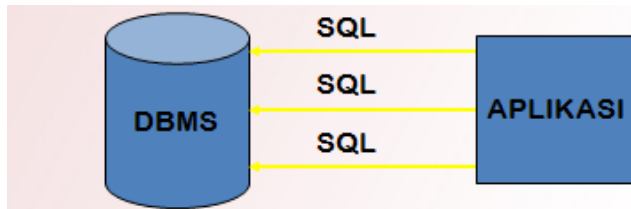
Keuntungan dari PL/SQL adalah :

- Pengembangan modularitas program
- Integrasi dengan tool-tools oracle
- Portability
- Penanganan kejadian
- Hemat network resource karena beberapa perintah sql bisa digabung menjadi Satu
- Bisa membuat custom function

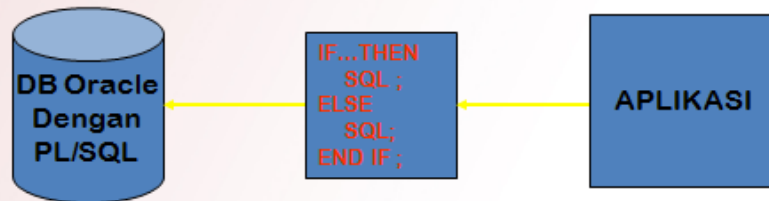
Manfaat Penggunaan PL/SQL

- PL / SQL diatur secara terpusat dalam database Oracle. Jadi tidak perlu mengatur aplikasi yang tersebar di client, yang tentunya sulit menanganinya.

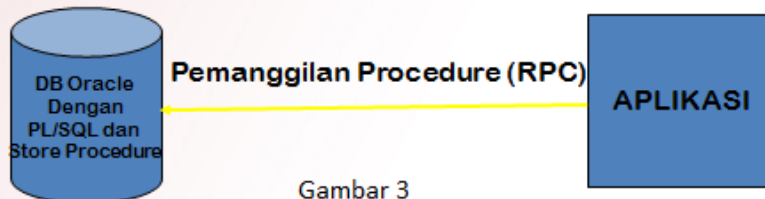
- PL / SQL merupakan “bahasa Oracle “ yang didesain khusus untuk berkomunikasi dengan objek database Oracle.
- PL / SQL adalah bahasa procedural yang mudah untuk dipahami dan powerfull



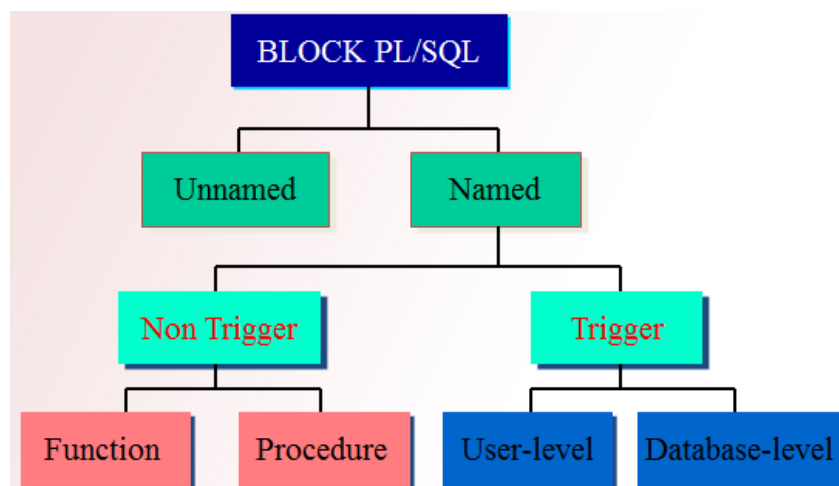
Gambar 1



Gambar 2



Gambar 3



Kode-kode dari PL/SQL ini tersimpan permanen di disk, yaitu pada tablespace.

Kode PL/SQL yang lengkap adalah sebagai berikut:

DECLARE(optional)

– Berisi deklarasi variabel, konstanta, prosedur, cursor, ataupun fungsi

BEGIN(wajib ditulis)

– Berisi Statement-statement yang akan dieksekusi

EXCEPTION(optional)

– *Berisi perintah error yang mungkin terjadi*

END;(wajib ditulis)

Struktur diatas dapat dijelaskan sebagai berikut :

1. **Bagian Judul (Header)**

Bagian ini hanya digunakan jika PL/SQL diberikan nama, misalnya untuk prosedur atau fungsi. Bagian ini berisi nama blok, daftar parameter, dan pengembalian hasil (return) jika blok adalah fungsi.

2. **Bagian Deklarasi (declaration)**

Bagian ini untuk membuat deklarasi mengenai semua variable dan konstanta yang direferensikan dalam pernyataan PL/SQL. Bagian deklarasi ini dimulai dengan perintah DECLARE. Jika tidak ada variable atau konstanta yang ingin dideklarasikan bagian ini boleh dihilangkan.

3. **Bagian Eksekusi (Execution).**

Bagian ini memuat pernyataan-pernyataan PL/SQL yang akan ditulis. Bagian eksekusi ini harus dimulai dengan perintah BEGIN.

4. **Bagian Perkecualian (Exception)**

Bagian ini memuat cara menangani kesalahan-kesalahan (error) pada waktu eksekusi program PL/SQL. Jika program tidak memuat cara menangani kesalahan, bagian ini boleh dihilangkan.

Setiap pernyataan PL/SQL harus diakhiri dengan tanda titik-koma(;) dan semua program PL/SQL harus diakhiri dengan perintah END. Pada bagian deklarasi ditempatkan variable dan konstanta yang dipakai oleh pernyataan PL/SQL yang dibuat.

Perbedaan variabel dengan konstanta adalah :

- **Konstanta** – nilai ditentukan pada saat deklarasi dibuat sehingga nilainya tetap pada saat program dieksekusikan.
- **Variabel** – dapat menerima nilai baru atau sebaliknya diubah pada saat program dieksekusi.

PL/SQL paling minimal adalah :

```
Begin
Null;
End
```

Agar dapat mengeluarkan pernyataan maka **serveroutput** harus di on kan. Untuk melihat status dari serveroutput adalah **show serveroutput**;
Untuk mengaktifkan adalah **set serveroutput on**;

Contoh sederhana program PL/SQL :

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Belajar Oracle');
END;
```

Aturan Penulisan Perintah PL/SQL

- Dapat ditulis dalam beberapa baris (tidak harus dalam satu baris perintah)
- Dapat berupa nested blok
- Karakter dan literal diapit oleh tanda kutip
- Setiap perintah/blok diakhiri dengan titik-koma (;)
- Komentar diawali dengan tanda min dua-kali (--) atau diapit dengan tanda /* ... */
- Pemberian nilai menggunakan :=
- Dapat menggunakan Bind Variable
- Tanda garis miring (/), berarti 'RUN'

Membuat Variable di PL/SQL

Variable ini letaknya di lokasi memori, yaitu punya nama, diisi dengan nilai dan nilainya berubah-ubah. Dalam **blok PL/SQL** kita juga dapat mendeklarasikan Variabel dan Konstanta. Variabel dideklarasikan dan diinisialisasi pada bagian **DECLARE**

Bentuk Umum Deklarasi Variabel :

<nama><type> [: <nilai variable>];

Symbol dalam PL/SQL

Symbol	Keterangan
	gabungan
=	Equality
:=	assignment

Contoh PL/SQL paling sederhana:

```
SQL> DECLARE
2 BEGIN
```



```
3 dbms_output.put_line('Hello world!!!, you know me??');  
4 END;  
5 /
```

Untuk menjalankannya cukup memberikan perintah SQL>/

Hasil yang diperoleh:

Hello world!!!, you know me???

Kegunaan Variabel

- Sebagai Tempat Penyimpanan Data Sementara
- Dapat dipakai berulang kali
- Memanipulasi Nilai Yang Telah Tersimpan Dalam Database

Tipe Data PL/SQL

@ Tipe Data Dasar :

- **Numerik** : NUMBER, BINARY_INTEGER, DEC, DOUBLE PRECISION, INTEGER, INT, NUMERIC, REAL, SMALLINT
- **Karakter** : VARCHAR2, CHAR, LONG
- **DATE**
- **BOOLEAN**

@ Tipe Data Tambahan :

- **Record**
- **Array**

IDENTIFIERS

Dalam membuat sebuah variabel, terlebih dahulu kita tentukan Identifier nya, dimana salah satu kegunaan Identifier adalah Sebagai Nama dari suatu variabel.

Dalam memberikan penamaan variabel, terdapat aturan yang harus kita patuhi yaitu:

- Tidak diawali dengan angka dan harus diawali dengan huruf
- Dapat memuat Angka
- Dapat memuat karakter spesial seperti _ , \$, dan #
- Terbatas hanya 30 karakter
- Bukan Kata kunci (int, varchar, table, trigger) dalam Oracle Server

Dalam Mendeklarasi dan Menginisialisai Variabel, Perhatikan Hal-hal berikut :

1. Patuhi Aturan Penamaan Variabel
2. Berikan Nama Variabel Yang sesuai dengan fungsinya
3. Inisialisasi Variabel dengan **NOT NULL** jika variabel tersebut harus mengandung sebuah nilai, dan gunakan **CONSTANT** jika variabel tersebut membutuhkan nilai yang permanen dan tidak dapat dirubah
4. Inisialisai variabel dengan asignment Operator (:=) atau dengan **DEFAULT**
 - o Name **VARCHAR(20):='Fern'** atau
 - o Name **NUMBER DEFAULT 'Fern'**
5. Deklarasikan satu Variabel hanya untuk satu baris dan 2 variabel untuk 2 baris yang berguna dalam hal maintenance dan readability
6. Hindari penggunaan Nama Kolom Pada suatu Tabel dalam penamaan Variabel

Deklarasi dan inisialisasi Variabel

Syntax :

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

Contoh Deklarasi dan inisialisasi:

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

Contoh pembatas Literal String:

```
SET SERVEROUTPUT ON  
DECLARE  
event VARCHAR2(15);  
BEGIN  
event := q'!Father's day!';  
DBMS_OUTPUT.PUT_LINE('3rd Sunday in June is : '||event);  
event := q'[Mother's day]';  
DBMS_OUTPUT.PUT_LINE('2nd Sunday in May is : '||event);  
END;  
/
```

Hasilnya:

3rd Sunday in June is : Father's day

2nd Sunday in May is : Mother's day

Nb: Karakter q pada q'!Father's day!' digunakan sebagai quote yang berguna agar karakter single quote (') pada kalimat tersebut dapat dipakai dan karakter !, [digunakan sebagai pembatas dimana quote dimulai dan berakhir.

Contoh Penggunaan Variabel:

1. Penggunaan Variabel

```
SQL> DECLARE
2 v_nama varchar(15);
3 v_alamat varchar2(20):='Jakarta';
4 BEGIN
5 v_nama:='Jakarta';
6 DBMS_OUTPUT.PUT_LINE('Nama :'||v_nama);
7 DBMS_OUTPUT.PUT_LINE('Alamat :'||v_alamat);
8 END;
9 /
```

Nama : Mercubuana

Alamat : Jakarta

2. Untuk delimiter dalam string literal

```
SQL> DECLARE
2 event varchar2(25);
3 BEGIN
4 event:='Tanggal STMIK BUDDHI';
5 DBMS_OUTPUT.PUT_LINE('5 November adalah '||event);
6 event:='Tanggal Indonesia';
7 DBMS_OUTPUT.PUT_LINE('17 Agustus adalah '||event);
8 END;
9 /
```

5 November adalah Tanggal STMIK BUDDHI

17 Agustus adalah Tanggal Indonesia

3. Memberi komentar

```
SQL> DECLARE
2 BEGIN
3 /*Mengeluarkan Pernyataan*/
4 DBMS_OUTPUT.PUT_LINE('Coba Komentar...');
```

```
5 END;
```

```
6 /
```

Coba Komentar...

4. untuk nilai not null dan konstanta

```
Sql>DECLARE
```

```
2 v_no_ktp char(15) not null := 'abc123';
```

```
3 v_phi constant number := 3.14;
```

```
4 begin
```

```
5 dbms_output.put_line('No KTP : ' || v_no_ktp);
```

```
6 dbms_output.put_line('PHI : ' || v_phi);
```

```
7 END;
```

```
8 /
```

5. Membuat sebuah variable baru yang bernama **name** dan bertipe **varchar2** yang berisikan data **first_name + last_name** dari tabel employees yang memiliki **id = 100**.

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
name VARCHAR2(50);
```

```
BEGIN
```

```
SELECT nama_pegawai INTO name
```

```
FROM pegawai
```

```
WHERE id_pegawai = 1024;
```

```
DBMS_OUTPUT.PUT_LINE('Name of the  
Employee is ' || name);
```

```
END;
```

Hasilnya adalah :

Name of the Employee is Steven King

PL/SQL procedure successfully completed.

6. Menampilkan data yang ada di dalam table yang di simpan ke dalam variabel kemudian di tampilkan ke layar.

```
SET SERVEROUTPUT ON
```

```
DECLARE
```

```
name VARCHAR2(122);
```

```
kar_id NUMBER := 1024;
```

```

bag_name VARCHAR2(13);
sal NUMBER;
BEGIN
SELECT nama_pegawai, gaji, nama_bag INTO Name, Sal, bag_name
FROM pegawai a join bagian b using(kode_bag)
WHERE id_pegawai = kar_id;
DBMS_OUTPUT.PUT_LINE('Nama= '||' '||name);
DBMS_OUTPUT.PUT_LINE('Salary = '||' '||Sal);
DBMS_OUTPUT.PUT_LINE('Name Department= '||' '||bag_name);
END;

```

Hasilnya :

Nama= Weiss

Salary = 8000

Name Department= Shipping

Boolean, Bind dan Substitusi Variabel

BOOLEAN VARIABLE

BOOLEAN merupakan tipe variabel yang hanya memiliki nilai TRUE, FALSE, dan NULL.

Contoh :

DECLARE

```

tipe1 BOOLEAN := TRUE;
tipe2 BOOLEAN := (1=2); // bernilai FALSE
tipe3 BOOLEAN := ('a'='b'); // bernilai False
tipe4 BOOLEAN := NULL; // bernilai NULL
tipe5 BOOLEAN := NOT NULL; // bernilai NOT NULL

```

BIND VARIABLE

Variabel **BIND** dibuat di dalam Environment dan juga disebut dengan HOST Variabel, dalam bahasa prosedural Seperti Bahasa C, **BIND** variabel disebut dengan global variabel. **BIND** variables diawali dengan kata kunci **VARIABLE** dan direferensikan dengan tanda titik dua (:). **Bind** Variables dapat digunakan pada SQL statement maupun **PL/SQL Block**.

Contoh :

```
VARIABLE emp_salary NUMBER
BEGIN
  SELECT gaji INTO :emp_salary
  FROM pegawai WHERE id_pegawai = 1024;
END;
/
PRINT emp_salary
```

EMP_SALARY
1000

Nb: Command **PRINT** digunakan untuk menampilkan value dari BIND VARIABLES

Di Dalam iSQL *PLUS juga terdapat COMMAND yang dapat menampilkan value dari BIND Variable secara otomatis, yaitu **SET AUTOPRINT ON**

Contoh :

```
SET AUTOPRINT ON
VARIABLE emp_salary NUMBER
BEGIN
  SELECT salary INTO : emp_salary
  FROM employees WHERE employee_id = 100;
END;
/
```

BIND Variabel yang telah kita buat tadi juga dapat dipakai pada SQL Statement

Contoh :

```
SELECT nama_pegawai FROM pegawai WHERE gaji = :emp_salary
```

LAST_NAME
King

SUBSTITUTION VARIABLE

Substitution Variables atau Variabel Pengganti merupakan variabel yang digunakan untuk memerintahkan user agar memasukkan nilai yang diinginkannya sendiri. Subtitution Variabel diawali dengan tanda ampersand (&).

Contoh :

```
SET AUTOPRINT ON
VARIABLE emp_salary NUMBER
DECLARE
```

```
emp_no NUMBER(6):=&empno;
BEGIN
SELECT gaji INTO :emp_salary
FROM pegawai WHERE id_pegawai = emp_no;
END;
/
PL/SQL procedure successfully completed.
```

EMP_SALARY	
	8000

Di Dalam SQL *PLUS juga terdapat COMMAND yang dapat memverifikasi nilai apa yang dimasukkan oleh user yaitu dengan menggunakan Command **SET VERIFY ON**

Contoh :

```
SET AUTOPRINT ON
SET VERIFY ON
VARIABLE emp_salary NUMBER
DECLARE
emp_no NUMBER(6):=&empno;
BEGIN
SELECT salary INTO :emp_salary
FROM employees WHERE employee_id = emp_no;
END;
/

old 2: emp_no NUMBER(6):=&empno;
new 2: emp_no NUMBER(6):=120;
PL/SQL procedure successfully completed.
```

EMP_SALARY	
	8000

Tipe Data Reference

Merupakan salah satu dari **macam-macam tipe data** pada pl/sql yang digunakan untuk mereferensikan tipe data pada **variabel** lain.

Attribut %TYPE

%TYPE adalah attribut yang digunakan untuk mendeklarasikan sebuah variable yang sesuai dengan:

- Definisi sebuah kolom database
- Deklarasi variable lain

Selain itu, %TYPE diawali dengan

- Tabel dan Kolom Database
- Nama dari Variabel yang dideklarasikan

Deklarasi Variabel dengan %TYPE

Syntax:

```
identifier table.column_name%TYPE;
```

Contoh :

```
emp_name employees.last_name%TYPE;
```

```
balance NUMBER(7,2);
```

```
min_balance balance%TYPE := 1000;
```

Pada contoh diatas, kita membuat 3 buah variabel yaitu Variabel **emp_name**, **balance**, dan **min_balance** dimana Variabel **emp_name** memiliki tipe yang sama dengan tipe kolom **last_name** pada tabel **employees**, lalu Variabel **balance** bertipe **Number**, Dan Variabel **min_balance** yang bertipe sama dengan Variabel balance yang memiliki tipe **NUMBER**

Attribut %ROWTYPE

%ROWTYPE digunakan untuk mendeklarasikan sebuah variabel yang sesuai dengan tipe data sejumlah kolom pada tabel atau view di database. Beda halnya dengan **%TYPE** yang hanya bisa mereferensikan satu tipe kolom atau satu variabel saja.

Keuntungan menggunakan %ROWTYPE :

- Kita tidak perlu mengetahui berapa jumlah dan tipe data kolom pada tabel atau view yang ingin kita referensikan
- Adanya kemungkinan terjadinya perubahan jumlah dan tipe data kolom pada tabel atau view pada saat run time
- Cocok digunakan saat menerima data menggunakan SELECT * statement

Syntax :

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


Contoh :

```
SET SERVEROUTPUT ON
DECLARE
Col_Emp pegawai%ROWTYPE;
BEGIN
SELECT * INTO Col_Emp
FROM pegawai
WHERE Employee_id = 1024;
DBMS_OUTPUT.PUT_LINE(Col_Emp.nama_pegawai);
DBMS_OUTPUT.PUT_LINE(Col_Emp.jabatan);
DBMS_OUTPUT.PUT_LINE(Col_Emp.gaji);
END;
```

Pada Contoh diatas kita membuat sebuah variabel Col_Emp yang mempunyai semua tipe data pada kolom tabel employees.

Nb : %ROWTYPE dan %TYPE sama-sama hanya bisa mengambil satu baris data

Soal Praktikum

1. Buatlah sebuah blok PL/SQL yang dapat menampilkan informasi berupa teks :
“BELAJAR” dan “PEMROGRAMAN ORACLE PL/SQL” dengan cara :
 - Tanpa mendefinisikan variabel, dan
 - Dengan mendefinisikan variabel-variabel yang dibutuhkan.
2. Buatlah sebuah blok PL/SQL sederhana sehingga pada layar SQL*Plus akan menampilkan teks berikut : “/* Ini adalah komentar program*/”
3. Perbaiki blok PL/SQL yang mengalami error dibawah ini :

```
Declare
X  CONSTANT INTEGER := 200 ;
BEGIN
X := 100 ;
DBMS_OUTPUT.PUT_LINE(TO_CHAR(X) );
END ;
```

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 10 Percabangan dan Perulangan Pada PL / SQL

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

10

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah percabangan dan perulangan pada PL / SQL

Kompetensi

Mampu memahami perintah-perintah percabangan dan perulangan pada PL / SQL

Percabangan

Percabangan adalah suatu kontrol untuk pengecekan yang dilakukan sebelum statemen-statement dalam sebuah blok PL/SQL dieksekusi. Statement yang dapat digunakan untuk percabangan adalah IF dan CASE, dengan syntax sebagai berikut :

IF-ELSE Statement

Sama seperti bahasa pemrograman yang lain, PL/SQL juga menyediakan Statement IF-ELSE.

Syntax :

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

Contoh :

```
DECLARE
    noBulan integer;
    namaBulan varchar2(9);
BEGIN
    noBulan := 3;
    IF noBulan = 1 then
        namaBulan := 'January';
    ELSIF noBulan = 2 then
        namaBulan := 'Februari';
    ELSIF noBulan = 3 then
        namaBulan := 'Maret';
    ELSIF noBulan = 4 then
        namaBulan := 'April';
    ELSIF noBulan = 5 then
        namaBulan := 'Mei';
```

```

ELSIF noBulan = 6 then
    namaBulan := 'Juni';
ELSIF noBulan = 7 then
    namaBulan := 'Juli';
ELSIF noBulan = 8 then
    namaBulan := 'Agustus';
ELSIF noBulan = 9 then
    namaBulan := 'September';
ELSIF noBulan = 10 then
    namaBulan := 'Oktober';
ELSIF noBulan = 11 then
    namaBulan := 'November';
ELSIF noBulan = 12 then
    namaBulan := 'Desember';
ELSE
    namaBulan := '';
END IF;
DBMS_OUTPUT.PUT_LINE(NAMABULAN);
END;
/

```

SET SERVEROUTPUT ON
SET VERIFY OFF

DECLARE

ANGKA NUMBER :=&n;

BEGIN

IF Angka > 10 THEN

DBMS_OUTPUT.PUT_LINE('ANGKA' || ANGKA || 'LEBIH BESAR DARI 10');

ELSIF ANGKA < 10 THEN

DBMS_OUTPUT.PUT_LINE('ANGKA' || ANGKA || 'LEBIH KECIL DARI 10');

ELSE

DBMS_OUTPUT.PUT_LINE('ANGKA' || ANGKA || 'SAMA DENGAN 10');

END IF;

END;

/

Blok PL/SQL diatas terdapat simbol **ampersand (&)** yang digunakan untuk meminta inputan user. Apabila user memasukkan angka lebih besar dari 10 maka Oracle Server akan mencetak statement DBMS_OUTPUT.PUT_LINE yang pertama. Jika Angka yang diinput lebih kecil dari 10 Maka **Oracle** Server akan mencetak statement DBMS_OUTPUT.PUT_LINE yang kedua. Dan jika tidak ada kondisi yang memenuhi semua kondisi diatas, maka oracle server akan mencetak statement DBMS_OUTPUT.PUT_LINE yang ketiga.

CASE Statement

Selain menggunakan statement IF, **Oracle** juga menyediakan alternatif lain untuk melakukan pemilihan statement, yaitu dengan menggunakan CASE Statement. **CASE** statement hampir sama logikanya dengan menggunakan SWITCH-CASE pada bahasa C.

Syntax :

CASE (ekspresi)

```
WHEN nilai_1 THEN Statemen_1;
WHEN nilai_2 THEN Statemen_2;
.....
WHEN nilai_n THEN Statemen_n
    [ELSE Statemen_lain;]
END CASE;
```

DECLARE

```
noBulan integer;
namaBulan varchar2(9);
```

BEGIN

```
noBulan := 3;
CASE noBulan
    WHEN 1 then namaBulan := 'January';
    WHEN 2 then namaBulan := 'Februari';
    WHEN 3 then namaBulan := 'Maret';
    WHEN 4 then namaBulan := 'April';
    WHEN 5 then namaBulan := 'Mei';
    WHEN 6 then namaBulan := 'Juni';
    WHEN 7 then namaBulan := 'Juli';
    WHEN 8 then namaBulan := 'Agustus';
```

```

        WHEN 9 then namaBulan := 'September';
        WHEN 10 then namaBulan := 'Oktober';
        WHEN 11 then namaBulan := 'November';
        WHEN 12 then namaBulan := 'Desember';
        ELSE namaBulan := "";
    END CASE;
    DBMS_OUTPUT.PUT_LINE(NAMABULAN);
END;
/

```

SET SERVEROUTPUT ON

SET VERIFY OFF

DECLARE

```
input VARCHAR2(1):= upper('&input');
```

```
kota VARCHAR2(50);
```

BEGIN

```
kota:=
```

```
    CASE input
```

```
        WHEN 'A' THEN 'Jakarta'
```

```
        WHEN 'B' THEN 'Padang'
```

```
        WHEN 'C' THEN 'Bandung'
```

```
        ELSE 'Indonesia'
```

```
    END;
```

```
    DBMS_OUTPUT.PUT_LINE ('Input: '|| input|| 'Kota' || kota);
```

```
END;
```

Perulangan

Pengulangan adalah suatu blok yang memperbolehkan statemen-statemen dalam sebuah blok PL/SQL diulang-ulang. Tiga buah jenis struktur pengulangan dalam Oracle, yaitu simple loop, For dan While.

BASIC LOOPS

Basic Loop pada PL/SQL hampir sama logikanya dengan DO-WHILE pada Bahasa Prosedural yang lain, contohnya pada bahasa C. Pada Basic Loop Minimal 1 kali Terjadi looping.

Syntax :

```
LOOP  
    Statement;  
    .....  
END LOOP;
```

Contoh :

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

```
SQL> DECLARE  
2 k integer;  
3 BEGIN  
4 k:=1;  
5 Loop  
6 DBMS_OUTPUT.PUT_LINE('Baris ke-'||TO_CHAR(k));  
7 k:=k+1;  
8 exit when k>5;  
9 END LOOP;  
10 END;  
11 /
```

FOR LOOPS

FOR pada PL/SQL logikanya sama dengan FOR pada Bahasa Prosedural yang lain, contohnya pada bahasa C. Dan kita menggunakan FOR LOOP ketika kita sudah mengetahui batas angka perulangannya

FOR variabel IN indeks_awal .. indeks_akhir LOOP

Statement;

.....

END LOOP

SQL> DECLARE

2 k integer;

3 BEGIN

4 FOR k in 1..5 LOOP

5 DBMS_OUTPUT.PUT_LINE('Baris ke – '||to_char(k));

6 END LOOP;

7 END;

/

DECLARE

countryid locations.country_id%TYPE := 'CA';

loc_id locations.location_id%TYPE;

new_city locations.city%TYPE := 'Montreal';

BEGIN

SELECT MAX(location_id) INTO loc_id

FROM locations

WHERE country_id = countryid;

FOR i IN 1..3 LOOP

INSERT INTO locations(location_id, city, country_id)

VALUES((loc_id + i), new_city, countryid);

END LOOP;

END;

/

WHILE LOOP

Kita menggunakan WHILE LOOPS ketika kita ingin mengecek terlebih dahulu kondisi yang ada, baru kemudian lakukan looping sampai syaratnya tidak terpenuhi.

WHILE kondisi LOOP

```
    Statemen;  
    .....  
END LOOP;
```

Contoh :

```
DECLARE  
    k integer;  
BEGIN  
    k := 1;  
    WHILE k <= 5 loop  
        DBMS_OUTPUT.PUT_LINE('Baris ke – '||to_char(k));  
        k := k+1;  
    END LOOP;  
END;  
/
```

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

Handling Exceptions

Handling Exceptions merupakan salah satu kelebihan dari **PL/SQL** yang digunakan untuk menangani error yang terjadi pada saat run time.

Sebagai contoh, pada saat kita menuliskan sebuah query, dan data tersebut tidak ditemukan di dalam database, maka **oracle** akan membangkitkan eksepsi **NO_DATA_FOUND** dengan kode **ORA-01403** yang berarti data tidak ditemukan. Berikut ini kita akan menuliskan perbedaan dengan menggunakan blok eksepsi dengan yang tidak menggunakan blok eksepsi.

Contoh Yang tidak Menggunakan blok eksepsi :

```
SET SERVEROUTPUT ON
DECLARE
    nama employees.last_name%type;
BEGIN
    SELECT last_name INTO nama FROM employees
    WHERE employee_id = 309;
    DBMS_OUTPUT.PUT_LINE ('nama');
END;
```

Hasilnya:

```
DECLARE
*
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 4
```

Contoh Yang Menggunakan Blok Eksepsi :

```
SET SERVEROUTPUT ON
DECLARE
    nama employees.last_name%type;
BEGIN
    SELECT last_name INTO nama FROM employees
    WHERE employee_id = 309;
    DBMS_OUTPUT.PUT_LINE ('nama');
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
```

```

        DBMS_OUTPUT.PUT_LINE ('Tidak Ditemukan Datanya dalam Database');
    END;

```

Hasilnya :

Tidak Ditemukan Datanya dalam Database
PL/SQL procedure successfully completed.

Berikut ini merupakan nama beserta Code exception di PL/SQL :

Exception	Oracle Error	SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Berikut ini merupakan keterangan dari nama exception di PL/SQL

Exception	Raised when ...
ACCESS_INTO_NULL	Your program attempts to assign values to the attributes of an uninitialized (atomically null) object.
CASE_NOT_FOUND	None of the choices in the WHEN clauses of a CASE statement is selected, and there is no ELSE clause.
COLLECTION_IS_NULL	Your program attempts to apply collection methods other than EXISTS to an uninitialized (atomically null) nested table or <u>varray</u> , or the program attempts to assign values to the elements of an uninitialized nested table or <u>varray</u> .
CURSOR_ALREADY_OPEN	Your program attempts to open an already open cursor. A cursor must be closed before it can be reopened. A cursor FOR loop automatically opens the cursor to which it refers. So, your program cannot open that cursor inside the loop.
DUP_VAL_ON_INDEX	Your program attempts to store duplicate values in a database column that is constrained by a unique index.
INVALID_CURSOR	Your program attempts an illegal cursor operation such as closing an unopened cursor.
INVALID_NUMBER	In a SQL statement, the conversion of a character string into a number fails because the string does not represent a valid number. (In procedural statements, VALUE_ERROR is raised.) This exception is also raised when the LIMIT-clause expression in a bulk FETCH statement does not evaluate to a positive number.
LOGIN_DENIED	Your program attempts to log on to Oracle with an invalid username and/or password.
NO_DATA_FOUND	A SELECT INTO statement returns no rows, or your program references a deleted element in a nested table or an uninitialized element in an index-by table. SQL aggregate functions such as AVG and SUM always return a value or a null. So, a SELECT INTO statement that calls an aggregate function never raises NO_DATA_FOUND. The FETCH statement is expected to return no rows eventually, so when that happens, no exception is raised.
NOT_LOGGED_ON	Your program issues a database call without being connected to Oracle.
PROGRAM_ERROR	PL/SQL has an internal problem.
ROWTYPE_MISMATCH	The host cursor variable and PL/SQL cursor variable involved in an assignment have incompatible return types. For example, when an open host cursor variable is passed to a stored subprogram, the return types of the actual and formal parameters must be compatible.
SELF_IS_NULL	Your program attempts to call a MEMBER method on a null

Exception	Raised when ...
	instance. That is, the built-in parameter SELF (which is always the first parameter passed to a MEMBER method) is null.
STORAGE_ERROR	PL/SQL runs out of memory or memory has been corrupted.
SUBSCRIPT_BEYOND_COUNT	Your program references a nested table or varray element using an index number larger than the number of elements in the collection.
SUBSCRIPT_OUTSIDE_LIMIT	Your program references a nested table or varray element using an index number (-1 for example) that is outside the legal range.
SYS_INVALID_ROWID	The conversion of a character string into a universal rowid fails because the character string does not represent a valid rowid.
TIMEOUT_ON_RESOURCE	A time-out occurs while Oracle is waiting for a resource.
TOO_MANY_ROWS	A SELECT INTO statement returns more than one row.
VALUE_ERROR	An arithmetic, conversion, truncation, or size-constraint error occurs. For example, when your program selects a column value into a character variable, if the value is longer than the declared length of the variable, PL/SQL aborts the assignment and raises VALUE_ERROR. In procedural statements, VALUE_ERROR is raised if the conversion of a character string into a number fails. (In SQL statements, INVALID_NUMBER is raised.)
ZERO_DIVIDE	Your program attempts to divide a number by zero.

Kita tidak perlu menghafal semua nama dari setiap exception handling diatas, kita cukup hanya menghafal beberapa saja dan sisanya kita dapat menggunakan eksepsi **OTHERS**.

DECLARE

empno **NUMBER**;

BEGIN

SELECT employee_id **INTO** empno **FROM** employees ;

DBMS_OUTPUT.PUT_LINE(empno);

EXCEPTION

WHEN NO_DATA_FOUND THEN

DBMS_OUTPUT.PUT_LINE('Data Tidak Ditemukan');

WHEN OTHERS THEN

DBMS_OUTPUT.PUT_LINE('Terjadi Error yang Lain');

END;

Hasil :

Terjadi Error yang Lain

PL/SQL procedure successfully completed.

PRAGMA EXCEPTION INIT

PRAGMA EXCEPTION INIT digunakan agar kita dapat mendefinisikan nama EXCEPTION kita sendiri, tetapi berdasarkan dengan ERROR CODE yang ada.

Contoh :

```
SET SERVEROUTPUT ON  
DECLARE  
    kebanyakan EXCEPTION;  
    PRAGMA EXCEPTION_INIT(keanyakan, -01422);  
    emp employees.employee_id%type;  
BEGIN  
    SELECT employee_id INTO emp FROM Employees;  
    DBMS_OUTPUT.PUT_LINE(EMP);  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        DBMS_OUTPUT.PUT_LINE('Hasil tidak ditemukan');  
    WHEN kebanyakan THEN  
        DBMS_OUTPUT.PUT_LINE('Hasilnya kebanyakan');  
END;
```

Pada mulanya error code -01422 merupakan code exception dari TOO_MANY_ROWS, tetapi dengan menggunakan PRAGMA_EXCEPTION_INIT, kita merubah TOO_MANY_ROW menjadi KEBANYAKAN. Yang sebelumnya, KEBANYAKAN telah dideklarasikan terlebih dahulu dengan menjadi sebuah EXCEPTION.

SQLCODE dan SQLERRM

SQLCODE dan **SQLERRM** biasa digunakan pada EXCEPTION OTHERS untuk mengetahui error apa yang terjadi. **SQLCODE** untuk menampilkan kode eksepsi, sedangkan **SQLERRM** digunakan untuk menampilkan keterangan eksepsi.

Contoh :

```
SET SERVEROUTPUT ON  
DECLARE  
    emp employees%rowtype;  
BEGIN  
    SELECT * into emp FROM EMPLOYEES WHERE employee_id > 100;  
EXCEPTION  
    WHEN no_data_found THEN  
        DBMS_OUTPUT.PUT_LINE('No Data Found');  
    WHEN OTHERS THEN  
        DBMS_OUTPUT.PUT_LINE('ERROR');
```

```

DBMS_OUTPUT.PUT_LINE('error code ='||SQLCODE);
DBMS_OUTPUT.PUT_LINE('error message ='||SQLERRM);
END;

```

Hasilnya :

```

ERROR
error code =-1422
error message =ORA-01422: exact fetch returns more than requested number of rows
PL/SQL procedure successfully completed.

```

RAISE APPLICATION ERROR

Pada PRAGMA EXCEPTION INIT kita hanya dapat mendefinisikan sendiri nama dari eksepsi tersebut dan masih harus sesuai dengan ERROR CODEnya. Tetapi dengan menggunakan RAISE APPLICATION ERROR, kita dapat menentukan sendiri nama beserta code dari eksepsi yang kita inginkan.

Contoh :

SET SERVEROUTPUT ON

```

DECLARE
    name VARCHAR2(100);
    sal NUMBER;
BEGIN
    SELECT last_name, salary INTO name, sal FROM employees WHERE
    employee_id = 120;
    IF sal > 5000 THEN
        RAISE_APPLICATION_ERROR (-20000,'Gaji Kebanyakan');
    END IF;
    DBMS_OUTPUT.PUT_LINE(name||' '||sal);
END;

```

Hasilnya :

```

DECLARE *
ERROR at line 1:
ORA-20000: Gaji Kebanyakan
ORA-06512: at line 8

```

Soal Praktikum

1. Buatlah Deret Bilangan Ganjil dengan menggunakan PL/SQL. Jumlah Berdasarkan deret inputan user.
2. Buatlah dalam blok PL/SQL yang dapat mencari dan menampilkan nama pegawai yaitu : Asfianti(catatan : gunakan tabel Pegawai milik user sales);
3. Buat Satu PL/SQL Menggunakan Percabangan (Kasus Bebas)
4. Buatlah Handling Exception masing-masing 1 Kasus untuk Handling Exception Sederhana, Pragma Exception, SQLCODE dan SQLERRM, dan Raise Application Error.

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 11 Procedure dan Function Pada PL / SQL

Fakultas
Ilmu Komputer

Program Studi
Teknik Informatika

Tatap Muka

11

Kode MK
87043

Disusun Oleh
Tim Dosen

Abstract

Memahami tentang perintah-perintah
procedure dan function pada PL / SQL

Kompetensi

Mampu memahami perintah-perintah
procedure dan function pada PL / SQL

PROCEDURE

Procedure dan function adalah sebuah blok PL/SQL yang dapat berdiri sendiri dan disimpan sebagai suatu objek di dalam *database* untuk melakukan tugas-tugas spesifik tertentu. Hal ini akan membuat kode yang dibuat lebih bersifat modular sehingga mudah untuk di-maintain.

Procedure adalah suatu blok PL/SQL yang menyimpan sekumpulan perintah yang tidak disertai dengan pengembalian nilai. Dengan kata lain, procedure hanya melakukan proses tertentu saja.

Prosedur merupakan subprogram PL/SQL yang berdiri sendiri. Kalau kita punya pekerjaan rutin dan command-commandnya pun itu-itu saja, kita bisa menyimpan command-command tersebut dan memanggilnya kapan saja kita mau. Itulah filosofi dari prosedur.

User yang membuat prosedur harus punya privilege “create procedure”.

```
SQL> GRANT CREATE PROCEDURE TO nama_user;
```

Syntax untuk membuat procedure :

CREATE digunakan untuk membuat procedure yang baru

REPLACE digunakan untuk mengganti isi procedure yang telah dibuat.

```
CREATE [OR REPLACE] PROCEDURE nama_procedure
```

```
(parameter_1 tippedata, parameter_2 tippedata, ...) AS
```

```
variabel-variabel_lokal
```

```
BEGIN
```

```
    Statemen;
```

```
    .....
```

```
END;
```

Untuk menjalankan prosedur, jalankan:

1. Di SQLPlus

```
SQL> exec Nama_Procedure;
```

atau

```
SQL> execute PROC_REFRESH_MYTAB;
```

2. Di block PL/SQL, tulis saja nama prosedur tersebut

```
DECLARE
```

```
BEGIN
```

```
Nama_Procedure;  
END;  
/
```

Contoh :

```
CREATE OR REPLACE PROCEDURE cetak AS  
    d varchar(50);  
BEGIN  
    d :=q'(Coba's lagi)';  
    DBMS_OUTPUT.PUT_LINE(d);  
END;  
/
```

Jalankan dengan perintah SQL>exec cetak;

Execute Procedure

EXECUTE nama_procedure(Parameter1,Parameter2,...);

- Parameter terbagi menjadi 3 yaitu parameter masukan, keluaran, dan parameter masukan/keluaran.
- Parameter masukan dideklarasikan sebagai data untuk pengolahan procedure dan ditandai dengan IN.

Contoh :

```
CREATE OR REPLACE PROCEDURE tambah (a IN INTEGER) AS  
    hasil INTEGER(5);  
BEGIN  
    hasil := a + 1;  
    DBMS_OUTPUT.PUT_LINE('Hasil akhir = ' || hasil);  
END;  
/
```

Jalankan dengan

EXECUTE tambah(9);

Parameter pada Procedure

Parameter pada procedure digunakan sebagai penghubung data antara procedure dengan si pemanggil procedure. Perlu kita ketahui lebih dahulu, parameter yang terdapat

pada Procedure dinamakan **Formal Parameter**. Sedangkan Parameter yang terdapat pada si pemanggil Procedure adalah **Actual Parameter**.

Tipe Parameter pada Procedure ada 3 :

1. **IN** parameter, Merupakan Tipe parameter yang didefinisikan pada aktual parameter untuk kemudian ditangkap oleh formal parameter. Kita tidak perlu menuliskan IN untuk mendefinisikan parameter tersebut, karena parameter IN telah didefinisikan secara DEFAULT oleh Oracle.
2. **OUT** parameter. Merupakan tipe parameter pada procedure yang nilainya dapat digunakan oleh si pemanggil procedure dan bisa dibilang OUT parameter merupakan kebalikan dari IN parameter.
3. **IN OUT** parameter. Tipe parameter yang digunakan untuk mengirimkan sebuah nilai ke procedure yang kemudian akan diproses dan dikembalikan kepada si pemanggil procedure.

Contoh IN Parameter

```
CREATE OR REPLACE PROCEDURE raise_salary
(id      IN employees.employee_id%TYPE,
percent IN NUMBER)
IS
BEGIN
    UPDATE employees
    SET    salary = salary * (1 + percent/100)
    WHERE employee_id = id;
END raise_salary;
/

EXECUTE raise_salary(176,10)
```

Contoh OUT Parameter

```
CREATE OR REPLACE PROCEDURE query_emp
(id      IN employees.employee_id%TYPE,
name     OUT employees.last_name%TYPE,
salary  OUT employees.salary%TYPE) IS
BEGIN
    SELECT last_name, salary INTO name, salary
    FROM    employees
    WHERE   employee_id = id;
END query_emp;

DECLARE
    emp_name employees.last_name%TYPE;
    emp_sal  employees.salary%TYPE;
BEGIN
    query_emp(171, emp_name, emp_sal); ...
END;
```

Nb: Sesuaikan urutan nilai parameter antara aktual parameter dengan Formal Parameter baik menggunakan IN atau OUT parameter.

Contoh IN OUT Parameter

Pertama kita buat terlebih dahulu Procedure `format_phone` yang akan mengubah format karakter

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

Setelah itu kita buat pemanggil procedurenya

```
DECLARE
phone VARCHAR2(21):='234234ASDA';
BEGIN
format_phone(phone);
DBMS_OUTPUT.PUT_LINE(phone);
END;
```

Perhatikan sebelumnya value dari Phone adalah '234234ASDA' . Lalu setelah kita panggil PROCEDURE `format_phone` dengan memasukkan **variabel** `phone` ke dalam parameternya, maka secara otomatis akan mengambil value tersebut (IN parameter) yang kemudian akan diproses dan dikembalikan lagi nilainya(OUT Parameter) kedalam bentuk yang berbeda menjadi '(234)234-ASDA' .

PASSING PARAMETERS

Ada berbagai macam cara dalam melakukan passing parameter.

1. Positional

Passing Parameter secara Positional adalah passing dengan menyesuaikan urutan pada formal parameter dengan aktual parameter, seperti yang sudah kita lakukan sebelumnya

2. Named

Kalau yang ini merupakan passing Parameter dengan cara menspesifikkan nama variable formal parameter di parameter aktual dengan menggunakan bantuan '`=>`' .
contoh :

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

3. **Combination**

Combination merupakan kombinasi dari Passing secara Positional dengan Named contoh :

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

Procedure dalam Procedure

```
CREATE OR REPLACE PROCEDURE cetak (x IN INTEGER) AS
```

```
    J INTEGER;
```

```
BEGIN
```

```
    FOR J IN 1..X LOOP
```

```
        DBMS_OUTPUT.PUT_LINE(TO_CHAR(J));
```

```
    END LOOP;
```

```
END;
```

```
/
```

```
CREATE OR REPLACE PROCEDURE panggil AS
```

```
BEGIN
```

```
    cetak(10);
```

```
END;
```

```
/
```

```
EXECUTE panggil;
```

Latihan Procedure

Contoh Latihan :

```
SQL> CREATE OR REPLACE PROCEDURE FIBO AS
```

```
2 K INT;
```

```
3 I INT;
```

```
4 J INT;
```

```
5 BEGIN
```

```
6 K:=1;
```

```
7 I:=1;
```

```
8 DBMS_OUTPUT.PUT_LINE(TO_CHAR(K));
```

```
9 DBMS_OUTPUT.PUT_LINE(TO_CHAR(I));
```

```
10 LOOP;
```

```

11 J:=K+I;
12 DBMS_OUTPUT.PUT_LINE(TO_CHAR(J));
13 K:=I;
14 I:=J;
15 EXIT WHEN J>20;
16 END LOOP;
17 END;
18 /

```

```

CREATE TABLE MHS (NAMA VARCHAR2 (20), NPM VARCHAR2 (8) PRIMARY KEY,
KELAS VARCHAR2 (5))

```

```

INSERT INTO MHS VALUES ('CHOIGI KIM', '50408042', '3GG09');
INSERT INTO MHS VALUES ('TAEYEON KIM', '50408009', '3GG09');
INSERT INTO MHS VALUES ('YURI KWON', '50408021', '3GG09');

```

```

SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE CETAK AS
NAMA VARCHAR2 (20);
NPM VARCHAR2 (8);
KELAS VARCHAR2 (5);
BEGIN
SELECT * INTO NAMA, NPM, KELAS FROM MHS WHERE NPM = '50408042';
DBMS_OUTPUT.PUT_LINE ('NAMA : ' || NAMA);
DBMS_OUTPUT.PUT_LINE ('NPM : ' || NPM);
DBMS_OUTPUT.PUT_LINE ('KELAS : ' || KELAS);
SELECT * INTO NAMA, NPM, KELAS FROM MHS WHERE NPM = '50408009';
DBMS_OUTPUT.PUT_LINE ('NAMA : ' || NAMA);
DBMS_OUTPUT.PUT_LINE ('NPM : ' || NPM);
DBMS_OUTPUT.PUT_LINE ('KELAS : ' || KELAS);
SELECT * INTO NAMA, NPM, KELAS FROM MHS WHERE NPM = '50408021';
DBMS_OUTPUT.PUT_LINE ('NAMA : ' || NAMA);
DBMS_OUTPUT.PUT_LINE ('NPM : ' || NPM);
DBMS_OUTPUT.PUT_LINE ('KELAS : ' || KELAS);
END;
/

```

```

CREATE TABLE MHS (NAMA VARCHAR2 (20), NPM VARCHAR2 (8) PRIMARY KEY,
KELAS VARCHAR2 (5));
SQL> CREATE OR REPLACE PROCEDURE tambah (npm varchar2,nama
varchar2,kelas varchar2) AS
2 begin
3 execute immediate 'truncate table mhs';
4 INSERT INTO MHS VALUES(npm,nama,kelas);
5 commit;
6 end;
7 /

```

```

CREATE OR REPLACE PROCEDURE Naik_Gaji ( id employees.employee_id%TYPE,
Tambahan NUMBER)
IS
BEGIN
UPDATE Employees
SET salary = salary + Tambahan
WHERE employee_id = id;
END;
/

```

```

CREATE OR REPLACE PROCEDURE Potong_Gaji(id NUMBER, Potong NUMBER)
IS
BEGIN
IF Potong > 2000 THEN
RAISE_APPLICATION_ERROR (-20000, ' Tidak Boleh >2000 ');
ELSE
UPDATE employees
SET Salary = Salary - Potong
WHERE employee_id = id;
END IF;
END;
/
EXECUTE Potong_Gaji (120, 3000)

```


FUNCTION

Stored Function merupakan sebuah **blok PL/SQL** yang dapat mengembalikan sebuah nilai. Stored Function juga dapat disimpan dalam sebuah schema object, sehingga dapat digunakan secara berulang-ulang.

Berikut Perbedaan **Stored Procedure** dengan Stored Function

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain RETURN clause in the header	Must contain a RETURN clause in the header
Can return values (if any) in output parameters	Must return a single value
Can contain a RETURN statement without a value	Must contain at least one RETURN statement

Syntax Function :

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

Contoh :

```
SET SERVEROUTPUT ON  
CREATE OR REPLACE FUNCTION Lihat_Gaji (ID  
Employees.employee_id %TYPE)  
RETURN NUMBER  
IS  
Gaji NUMBER;  
BEGIN  
SELECT Salary INTO Gaji FROM Employees  
WHERE employee_id = id;  
RETURN Gaji;  
END;  
/  
EXECUTE DBMS_OUTPUT.PUT_LINE (Lihat_Gaji(100))
```

Fungsi diatas adalah fungsi yang digunakan untuk melihat gaji employee berdasarkan employee_id.

Selain dengan cara diatas, kita juga dapat menggunakan **SQL statement** untuk mendapatkan nilai dari sebuah function.

contoh :

```
SELECT employee_id,last_name, Lihat_gaji(employee_id) FROM employees
```

Contoh lain :

```
CREATE OR REPLACE FUNCTION Naik_Gaji (Salary NUMBER) RETURN  
NUMBER
```

```
IS
```

```
Gaji_Sekarang NUMBER;
```

```
BEGIN
```

```
Gaji_Sekarang := Salary + 1000 ;
```

```
RETURN Gaji_Sekarang;
```

```
END;
```

```
/
```

```
SELECT Employee_id, Naik_Gaji(salary) FROM Employees
```

```
WHERE Department_id = 20;
```

EMPLOYEE_ID	NAIK_GAJII(SALARY)
201	14000
202	8900

Berikut tempat-tempat yang dapat kita gunakan sebuah function pada SQL Statement :

1. Pada **SELECT**_list
2. Pada Clausa WHERE atau HAVING
3. Pada Clausa CONNECT BY, START WITH, ORDER BY, dan **GROUP BY**
4. Pada VALUES clause dalam **INSERT**_query
5. Pada SET clause dalam **UPDATE** query

Drop Function

Kita dapat menghapus fungsi yang telah kita buat dengan menggunakan perintah DROP:

Syntax :

```
DROP FUNCTION nama_function
```

Return Value

Sebuah function akan mengembalikan nilai sesuai dengan tipe data yang ditentukan. Untuk memberikan nilai kedalam function setelah diolah, maka dalam function dikenal dengan RETURN.

Contoh

```
CREATE OR REPLACE FUNCTION nama_function
(parameter1,parameter2,...)
RETURN tipe_data AS
variable1 tipe_data;
...
BEGIN
statement;
RETURN nilai_yang_dikembalikan
END;
```

Function without Parameter

```
CREATE OR REPLACE FUNCTION tulis
RETURN VARCHAR2 AS
BEGIN
RETURN 'Hello Hello';
END;
atau
CREATE OR REPLACE FUNCTION tulis
RETURN VARCHAR2 AS hsl VARCHAR2(20);
BEGIN
hsl:= 'Hello Hello';
RETURN hsl;
END;
```

Function with Parameter

```
CREATE OR REPLACE FUNCTION isprime(bil INTEGER) RETURN BOOLEAN AS
PRIMA BOOLEAN:=TRUE;
J INTEGER;
BEGIN
IF bil <= 1 THEN
PRIMA := FALSE;
```

```

END IF;
FOR J IN 2..(bil/2) LOOP
    IF MOD(bil,J) = 0 THEN
        PRIMA:=FALSE;
        EXIT;
    END IF;
END LOOP;
RETURN PRIMA;
END;
CREATE OR REPLACE FUNCTION pangkat (bil INTEGER, n INTEGER)
RETURN INTEGER AS
HASIL INTEGER(10);
I INTEGER;
BEGIN
    HASIL := 1;
    FOR I IN 1..n LOOP
        HASIL := HASIL * bil;
    END LOOP;
    RETURN HASIL;
END;
/

DECLARE
    H INTEGER;
BEGIN
    H := pangkat(2, 3);
    DBMS_OUTPUT.PUT_LINE('Hasil = ' || TO_CHAR(H));
END;
/

```

Nested Function

```

CREATE OR REPLACE FUNCTION kuadrat(X NUMBER) RETURN NUMBER AS
HASIL NUMBER(10);
BEGIN
    HASIL :=X*X;
    RETURN HASIL;
END;

```

```

CREATE OR REPLACE FUNCTION determinan (a NUMBER, b NUMBER, c NUMBER)
RETURN NUMBER AS D NUMBER(10);
BEGIN
    D:=kuadrat(b)-(4*a*c);
    RETURN D;
END;

SET SERVEROUTPUT ON
DECLARE D NUMBER (10);
BEGIN
    D:=determinan(1,1-6);
    DBMS_OUTPUT.PUT_LINE('Nilai Determinan = ' || TO_CHAR(D));
END;

```

Soal Praktikum

Apa output dari contoh program di bawah ini :

Create or replace procedure keliling_lingkaran as

R number(5);

K number(10);

Begin

R := 21;

K := 2*((22/7)*R);

Dbms_output.put_line ('Keliling Lingkaran = ' || K);

End;

Execute keliling_lingkaran;

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 12 Trigger Pada PL / SQL

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

12

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah trigger pada PL / SQL

Kompetensi

Mampu memahami perintah-perintah trigger pada PL / SQL

TRIGGER

Trigger adalah sebuah obyek dalam database yang berupa prosedur dalam bentuk kode-kode sql yang merespon setiap kali terdapat proses modifikasi dalam table. Proses modifikasi itu adalah berupa Data Manipulation Language(DML) yaitu insert, update dan delete.

Tujuan dari trigger adalah untuk menjaga integritas dan konsistensi data. Biasanya diimplementasikan pada tabel-tabel yang memiliki relasi. Trigger dapat dieksekusi sebelum (before Event) atau sesudah (after Event) modifikasi.

Agar user dapat membuat object trigger maka harus diberi privileges trigger. Caranya login sebagai user sys dan berikan perintah :

```
GRANT CREATE Trigger to nama_user;
```

Trigger biasanya digunakan untuk **menampilkan komentar** apabila terjadi suatu aksi pada table. Aksi tersebut adalah statement Dynamic Modeling Language (DML), dan dilaksanakan pada kejadian tertentu. Selain kejadian secara umum juga kejadian yang menggunakan kondisi. Kelebihan trigger di Oracle kita juga bisa membuat kode dan pesan error sendiri yang tidak ada dalam Oracle. Dalam memberi kode error harus di atas 20000, karena itu kode yang dibolehkan, sebab kode di 20000 sudah dipakai oleh Oracle. Trigger akan memicu perubahan di table lain jika ada aksi tertentu pada suatu table.

Untuk menampilkan pesan apabila transaksi terjadi menggunakan perintah `dbms_output.put_line('...kalimat pesan...');` sedangkan untuk menampilkan pesan error adalah `raise_application_error(-kode_error,'...kalimat pesan error...');`

Secara default pesan tidak akan ditampilkan karena **serveroutput** masih off. Untuk melihat status server output dengan perintah `show serveroutput;` dan untuk merubah status ke on adalah `set serveroutput on;`

Trigger terdapat 2 jenis :

1. Application Trigger

Yang dimana trigger tersebut akan teraktivasi ketika terjadi suatu event pada sebuah aplikasi tertentu.

2. Database Trigger

Yang dimana trigger tersebut akan teraktivasi ketika terjadi suatu data event (operasi DML (**INSERT, UPDATE, DELETE**)) atau system event (logon or shutdown) pada

sebuah schema atau database dan untuk selanjutnya kita akan membahas tentang trigger ini.

Syntax Trigger

```
CREATE OR REPLACE TRIGGER nama_trigger (BEFORE | AFTER)
(INSERT|UPDATE|UPDATE OF nama_kolom|DELETE) ON nama_tabel
FOR EACH ROW
WHEN (kondisi)
DECLARE
    variable1;
    ...
BEGIN
    statemen1;
    ...
END;
```

Event

- **BEFORE INSERT**: dieksekusi sebelum statement Insert
- **BEFORE INSERT FOR EACH ROW** : dieksekusi sebelum setiap baris baru dibuat.
- **AFTER INSERT** : diaktivasi setelah statement insert sukses dilakukan
- **AFTER INSERT FOR EACH ROW** : diaktivasi setelah setiap baris baru dibuat.
- **BEFORE UPDATE** : diaktivasi sebelum statement Update
- **BEFORE UPDATE FOR EACH ROW** : diaktivasi sebelum setiap baris baru diubah.
- **AFTER UPDATE** : diaktivasi setelah statement update sukses dilakukan
- **AFTER UPDATE FOR EACH ROW** : diaktivasi setelah setiap baris baru diubah.
- **BEFORE DELETE** : diaktivasi sebelum statement Delete
- **BEFORE DELETE FOR EACH ROW** : diaktivasi sebelum setiap baris baru dihapus.
- **AFTER DELETE** : diaktivasi setelah statement Delete sukses dilakukan.
- **AFTER DELETE FOR EACH ROW** : diaktivasi setelah setiap baris baru dihapus.

Contoh :

```
*/Buat table stok terlebih dahulu
CREATE OR REPLACE TRIGGER stok2
BEFORE INSERT ON STOK
FOR EACH ROW
BEGIN
```



```

        DBMS_OUTPUT.PUT_LINE('Trigger BEFORE INSERT dilaksanakan');
    END;
/

```

Untuk melihat apakah trigger tersebut dieksekusi, dapat dilakukan proses insert terhadap tabel STOK.

Mendapatkan Nilai Kolom

- Oracle menyediakan 2 buah alias yaitu :new dan :old.
- Alias :new digunakan untuk mengambil nilai kolom dari baris yang akan dimasukkan ke dalam sebuah tabel. Alias :old digunakan untuk mengambil nilai kolom dari baris yang tersimpan didalam tabel.
- Setiap proses manipulasi data (INSERT, UPDATE, dan DELETE) memiliki alias yang berbeda beda.

- **Proses Insert**

Pada proses insert, di dalamnya hanya terdapat alias :new yang berfungsi untuk mengambil nilai kolom dari baris yang akan dimasukkan.

INSERT INTO STOK (KODE, NAMA_BARANG, JUMLAH)

VALUES ('0007','Komputer', 5)

:new.KODE : 0007

:new.NAMA_BARANG : Komputer

:new.JUMLAH : 5

- **Proses Update**

Dalam Proses update alias :new akan digunakan untuk mengambil nilai kolom pada data yang baru, sedangkan alias : old akan mengambil nilai kolom dari data yang sebelumnya.

UPDATE STOK SET JUMLAH = 10 WHERE KODE = '0007'

:new.JUMLAH : 10

:old.JUMLAH : 5 (data sebelumnya)

- **Proses Delete**

Namun dalam proses delete, hanya dikenal alias :old sebagai penampung data kolom pada nilai yang dihapus.

DELETE FROM STOK WHERE KODE = '0007';

:old.KODE : 0007

:old.NAMA_BARANG : Komputer

:old.JUMLAH : 10

Contoh :

```
CREATE OR REPLACE TRIGGER backup
BEFORE UPDATE ON Employees
BEGIN
  RAISE_APPLICATION_ERROR(-20001,'Tabel Employees Tidak Dapat Dirubah');
END;
```

Trigger diatas berfungsi untuk melarang terjadinya proses UPDATE pada tabel employees dan Ketika kita mencoba untuk mengUpdate tabel employees

```
UPDATE employees
SET salary = 1000
WHERE employee_id= 100
```

Hasilnya :

ERROR at line 1:

ORA-20001: Tabel Employees Tidak Dapat Dirubah

ORA-06512: at "HR.BACKUP_SALARY", line 2

ORA-04088: error during execution of trigger 'HR.BACKUP_SALARY'

```
CREATE OR REPLACE TRIGGER backup
BEFORE UPDATE OF SALARY ON employees
BEGIN
  RAISE_APPLICATION_ERROR(-20001,'Data Salary Pada Tabel Employees Tidak
  Dapat Dirubah');
END;
```

Trigger diatas akan aktif ketika kita melakukan operasi UPDATE pada Kolom Salary di Tabel employees.

IF-ELSE TRIGGER

```
CREATE OR REPLACE TRIGGER backup2
BEFORE UPDATE OR DELETE OR INSERT ON employees
BEGIN
  IF UPDATING THEN
```

```

        RAISE_APPLICATION_ERROR(-20001, 'Tabel Emp Tidak Dapat DiRubah');
ELSIF DELETING THEN
        RAISE_APPLICATION_ERROR(-20002, 'Tabel Emp Tidak Dapat DiHapus');
ELSE
        RAISE_APPLICATION_ERROR(-20003, 'Tabel Emp Tidak Dapat DiTambah');
END IF;
END;
/

```

```

CREATE OR REPLACE TRIGGER Konfirmasi
AFTER DELETE OR INSERT OR UPDATE ON Employees
BEGIN
IF DELETING THEN
        DBMS_OUTPUT.PUT_LINE ('Data Telah di Delete');
ELSIF UPDATING THEN
        DBMS_OUTPUT.PUT_LINE ('Data Telah di Update');
ELSIF INSERTING THEN
        DBMS_OUTPUT.PUT_LINE ('Data Telah di Insert');
END IF;
END;
/

```

Trigger diatas akan menampilkan pesan ketika kita melakukan operasi DML(INSERT, UPDATE, DELETE) pada tabel employees

Alias Di Dalam Trigger

Di dalam trigger dikenal istilah alias atau **referensi**, yaitu sejenis **variabel** yang menyimpan nilai dari suatu kolom di dalam tabel. Alias tersebut terbagi menjadi dua yaitu :new dan :old. Alias :new untuk menyimpan nilai terbaru sedangkan alias :old untuk menyimpan nilai lama.

Ketersediaan Alias Di dalam Trigger

- Pada statement UPDATE, terdapat alias :new dan :old
- Pada statement DELETE, hanya terdapat alias :old
- Pada statement INSERT, hanya terdapat alias :new

```

CREATE OR REPLACE TRIGGER backup_sal
AFTER UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
    INSERT INTO backup_salary(emp_id, old_salary, new_salary, backupdate)
    VALUES (: old.employee_id, : old.salary, :new.salary, SYSDATE);
END;
/

```

Trigger diatas berfungsi untuk memasukkan data employee_id yang lama, salary yang lama, salary yang baru dan tanggal perubahan ke tabel backup_salary. Trigger tersebut teraktivasi Ketika kita mengUpdate salary pada Tabel Employees

```

CREATE OR REPLACE TRIGGER Validasi
BEFORE INSERT ON Employees
FOR EACH ROW
BEGIN
    IF :new.salary < 2000 THEN
        RAISE_APPLICATION_ERROR (-20000,'Gaji Minimal $2000');
    END IF;
END;
/

```

Trigger diatas berfungsi untuk memastikan kalau data salary pada tabel employees yang dimasukkan minimal \$2000

Membuat Trigger dalam DDL Statement

Trigger yang dapat dipasang pada DDL stament yaitu **DROP**, **ALTER**, dan **CREATE**.

Syntax :

```

CREATE [OR REPLACE] TRIGGER trigger_name
Timing
[ddl_event1 [OR ddl_event2 OR ...]]
ON {DATABASE|SCHEMA}
trigger_body

```

Contoh :

```
CREATE OR REPLACE TRIGGER BackupData
BEFORE DROP ON SCHEMA
BEGIN
    RAISE_APPLICATION_ERROR(-20005,' Tidak diperbolehkan untuk
    menghapus table pada schema ini');
END;
```

Trigger diatas akan mencegah terjadinya pengeksekusian statement DROP pada schema.

MANAGING TRIGGER

Mengaktifkan TRIGGER

Syntax :

```
ALTER TRIGGER trigger_name ENABLE
```

Menonaktifkan Trigger

Syntax :

```
ALTER TRIGGER trigger_name DISABLE
```

Mengaktifkan atau Menonaktifkan Semua Trigger

Syntax :

```
ALTER TABLE table_name DISABLE | ENABLE
ALL TRIGGERS
```

Menghapus TRIGGER

Syntax :

```
DROP TRIGGER nama_trigger
```

Trigger on System Event

Setelah sebelumnya kita telah mempelajari bagaimana membuat **trigger** dalam **database**, Kali ini kita akan mempelajari tentang bagaimana caranya membuat trigger dalam sistem.

Di dalam system terdapat Event-event Yang Mungkin terjadi yaitu :

Event	Possible Value
AFTER SERVERERROR	Trigger akan diaktifkan ketika terjadinya server error
AFTER LOGON	Trigger akan diaktifkan ketika ada user yang masuk ke database
BEFORE LOGOFF	Trigger akan diaktifkan ketika user ingin keluar dari database
AFTER STARTUP	Trigger akan dijalankan ketika database dijalankan
BEFORE SHUTDOWN	Trigger akan dijalankan ketika database ingin di shut down

Kita dapat menggunakan event-event trigger diatas pada SCHEMA maupun **DATABASE** tetapi tidak untuk event ShutDown dan StartUp yang hanya dapat dijalankan pada DATABASE saja.

Syntax :

```
CREATE [OR REPLACE] TRIGGER trigger_name  
timing  
[database_event1 [OR database_event2 OR ...]]  
ON {DATABASE|SCHEMA}  
trigger_body
```

Contoh :

Kita akan membuat sebuah trigger yang digunakan untuk mengetahui kapan dan siapa user yang login dan logoff pada sebuah schema. Hal pertama yang kita lakukan adalah membuat sebuah table History yang nantinya akan menampung data tentang siapa yang login dan logoff dari schema.

```
CREATE TABLE History (  
    Users Varchar2(20),  
    LogDate Date,  
    Action VARCHAR2(20)  
);
```

Setelah itu kita buat 2 buah trigger yaitu trigger logon_trig yang digunakan untuk memasukkan detail data user yang logon ke dalam tabel history dan yang kedua adalah

trigger logoff_trig yang digunakan untuk memasukkan detail data user yang logoff dari skema.

```
CREATE OR REPLACE TRIGGER logon_trig
AFTER LOGON ON SCHEMA
BEGIN
INSERT INTO History VALUES (USER, SYSDATE, 'Logging on');
END;
/
```

```
CREATE OR REPLACE TRIGGER logoff_trig
BEFORE LOGOFF ON SCHEMA
BEGIN
INSERT INTO History VALUES (USER, SYSDATE, 'Logging off');
END;
/
```

USERS	LOGDATE	ACTION
HR	03-03-2012	Logging on
HR	03-03-2012	Logging off

Karena contoh diatas kita menspesifikkan trigger tersebut pada schema (ON SCHEMA), trigger logon dan logoff_trig hanya akan diaktifkan ketika kita login dan logoff pada schema tempat kita membuat trigger login dan logoff_trigger, misalkan kita membuat trigger tersebut pada schema yang dimiliki oleh hr, maka data yang ada adalah data yang keluar dan masuk pada skema hr.

Sedangkan jika kita menspesifikkan trigger tersebut pada DATABASE (ON DATABASE), trigger tersebut akan aktif terhadap schema yang dimiliki oleh semua user. Namun untuk dapat membuat sebuah trigger pada database diperlukan sebuah ADMINISTER DATABASE TRIGGER privilege.

Berikut **Privilege** yang digunakan dalam trigger :

- CREATE/ALTER/DROP (ANY) TRIGGER privilege, yang digunakan untuk membuat trigger pada schema
- ADMINISTER DATABASE TRIGGER privilege, yang digunakan untuk membuat trigger dalam database
- EXECUTE privilege, yang digunakan apabila trigger yang kita buat diperuntukkan untuk object yang tidak kita miliki

Manfaat Trigger

Dengan adanya trigger kita dapat menggunakannya untuk :

- **Security**

Dengan adanya trigger kita dapat membatasi siapa saja yang dapat mengakses object dalam database sesuai dengan nilai yang telah dideklarasikan oleh trigger tersebut.

- **Auditing**

Dengan adanya trigger kita dapat melacak aliran data yang terjadi di dalam database

- **Data integrity**

Trigger dapat menjaga integritas data

- **Referential integrity**

Ketika Oracle Server hanya dapat membuat standard referential integrity rules, Trigger dapat mengimplementasikan nonstandard functionality

- **Table replication**

Trigger dapat menyalin Table ke dalam bentuk replika

- **Computing derived data automatically**

Triggers compute derived data values automatically

- **Event logging**

Trigger dapat membuat history logging secara transparan

Soal Praktikum

1. Buatlah trigger untuk menambahkan jumlah pasok barang yang ada pada tabel barang setiap kali dilakukan pemasukan (*insert*) data pada tabel pasok, dimana jumlah yang ditambahkan ke dalam stok barang tersebut adalah jumlah pasok pada saat pemasukan!
2. Buatlah trigger untuk mengurangi jumlah stok barang yang ada pada tabel barang setiap kali dilakukan penghapusan (*delete*) data pada tabel pasok, dimana jumlah yang digunakan untuk pengurangan stok barang tersebut adalah jumlah pasok dari baris yang dihapus pada tabel pasok!

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 13 Cursor Pada PL / SQL

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

13

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang perintah-perintah cursor pada PL / SQL

Kompetensi

Mampu memahami perintah-perintah cursor pada PL / SQL

Cursor

Pada block PL/SQL kita tidak bisa **menampilkan beberapa baris dengan menggunakan perintah SELECT** secara langsung. Untuk mengatasi hal tersebut, maka kita dapat menggunakan sebuah kursor. Berbeda dengan variabel skalar, cursor dapat menampung banyak nilai berupa baris atau record. Nilai-nilai yang disimpan dalam sebuah kursor kemudian dapat dimanipulasi sehingga dapat digunakan sesuai kebutuhan.

Cursor sendiri terdiri dari 2 tipe :

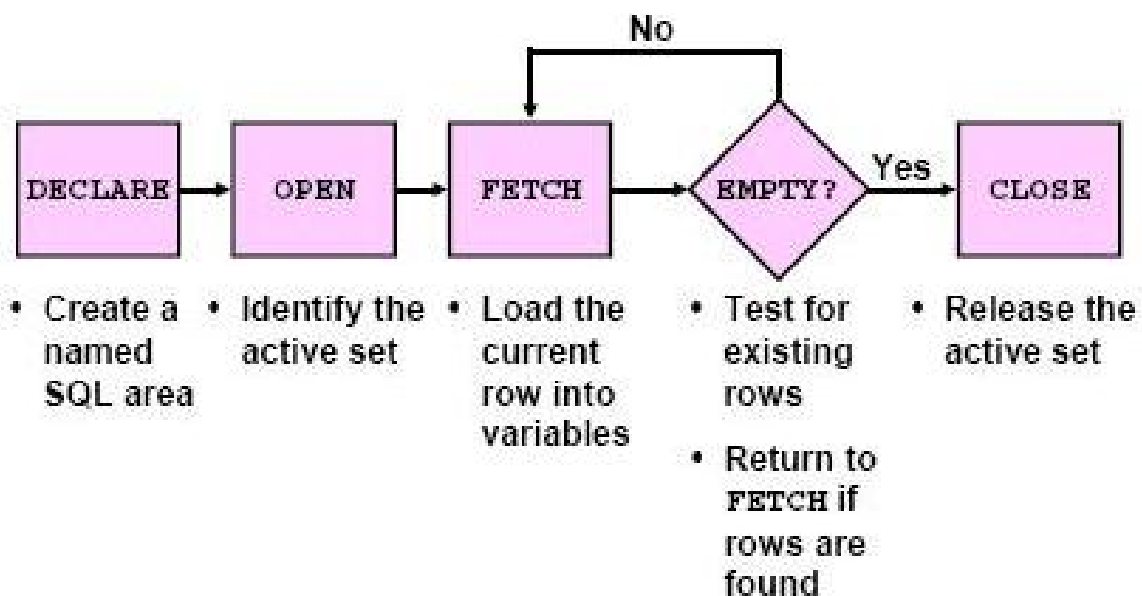
1. **Implicit Cursor**

Merupakan Cursor yang dideklarasikan dan dikelola secara otomatis oleh PL/SQL untuk semua operasi DML SQL dan PL/SQL SELECT Statement

2. **Explicit Cursor**

Merupakan Cursor yang dikelola dan dideklarasikan oleh Programmer

Nb : Sebelum memulai mempelajari CURSOR ada baiknya mempelajari metode **LOOPING** terlebih dahulu.



Dalam Membuat cursor terdapat 4 tahap yang harus kita lakukan yaitu :

1. **DECLARE**

DECLARE digunakan untuk mendeklarasikan cursor yang akan kita gunakan

2. **OPEN**

OPEN digunakan untuk mengaktifkan cursor yang akan kita gunakan

3. FETCH

FETCH digunakan untuk mengambil data dari cursor, dan kemudian data tersebut kita tampung ke dalam satu atau beberapa variabel.

4. CLOSE

CLOSE digunakan untuk menonaktifkan atau menutup cursor yang kita gunakan

Sintax Deklarasi Cursor :

```
CURSOR cursor_name IS  
select_statement;
```

Contoh :

```
DECLARE  
CURSOR cur_emp IS  
SELECT employee_id, last_name FROM employees WHERE department_id = 30
```

Cursor diatas kita deklarasikan dengan menggunakan nama cur_emp dan cursor tersebut menunjuk atau berisi data employee_id, last_name dari tabel employees yang mempunyai department id = 30

Contoh OPEN CURSOR :

```
DECLARE  
CURSOR cur_emp IS  
SELECT employee_id, last_name FROM employees WHERE department_id = 30;  
BEGIN  
OPEN cur_emp;
```

Contoh FETCH CURSOR :

```
DECLARE  
CURSOR cur_emp IS  
SELECT employee_id, last_name FROM employees WHERE department_id = 30;  
empID employees.employee_id%TYPE;  
Name employees.last_name%TYPE;  
BEGIN  
OPEN cur_emp;  
FETCH cur_emp INTO empID, Name ;
```

Contoh Lengkap :

```
SET SERVEROUTPUT ON
DECLARE
CURSOR emp_cursor IS
SELECT employee_id, last_name, salary FROM Employees
WHERE department_id = 20;
empId employees.employee_id%type;
name employees.last_name%type;
Sal employees.salary%type;
BEGIN
OPEN emp_cursor;
LOOP
FETCH emp_cursor INTO empID, Name, Sal ;
EXIT WHEN emp_cursor%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(empID||' '||Name||' '||Sal);
END LOOP;
DBMS_OUTPUT.PUT_LINE('jumlah Data '||' '||emp_cursor%ROWCOUNT);
CLOSE emp_cursor;
END;
/
```

Cursor diatas dideklarasikan dengan nama emp_cur yang berisi data employee_id, last_name dari tabel employees dengan department_id =30. lalu dibuat 3 buah variabel baru yang bernama empID, Name, dan Sal dengan **tipe data reference**. Karena employee_id pada department id = 30 terdapat beberapa employee, maka kita gunakan sebuah metode **LOOPING** yang akan secara otomatis akan menunjuk data satu persatu yang mempunyai department_id =30.

Berhenti ketika Cursor tersebut tidak menemukan data yang mempunyai department id = 30. Setelah itu datanya kita FETCH dan kemudian ditampung ke dalam variabel empId, Name, dan Sal. Lalu kita tampilkan hasilnya dengan menggunakan DBMS_OUTPUT.PUT_LINE. Dan terakhir cursor tersebut ditutup.

CURSOR Dengan RECORD

```
SET SERVEROUTPUT ON
DECLARE
CURSOR emp_cursor IS
SELECT employee_id, last_name, Salary FROM employees
WHERE department_id = 30;
emp_record emp_cursor%ROWTYPE;
BEGIN
OPEN emp_cursor;
LOOP
FETCH emp_cursor INTO emp_record;
EXIT WHEN emp_cursor%NOTFOUND;
DBMS_OUTPUT.PUT_LINE( emp_record.employee_id|| ' ||emp_record.last_name||'
'||emp_record.salary);
END LOOP;
DBMS_OUTPUT.PUT_LINE('Jumlah Data Yang Diambil:
'||emp_cursor%ROWCOUNT);
CLOSE emp_cursor;
END;
/
```

CURSOR dengan FOR

```
SET SERVEROUTPUT ON
DECLARE
CURSOR emp_cur IS
SELECT employee_id, Last_name FROM employees
WHERE department_id = 30;
BEGIN
FOR emp_rec IN Emp_cur
LOOP
DBMS_OUTPUT.PUT_LINE(emp_rec.employee_id || emp_rec.last_name);
END LOOP;
END;
/
```

Dengan menggunakan metode pengulangan FOR, kita tidak perlu lagi mendeklarasikan emp_rec karena **record** tersebut secara implisit telah dideklarasikan.

Selain itu, kita juga tidak perlu mendeklarasikan OPEN, FETCH, EXIT, dan CLOSE seperti saat kita menggunakan metode BASIC LOOP karena perintah tersebut juga secara implisit telah dideklarasikan. Dan Pada contoh diatas, emp_record akan berisi semua data pada Emp_Cur

CURSOR dengan FOR SUBQUERY

```
SET SERVEROUTPUT ON
BEGIN
FOR emp_record IN (SELECT employee_id, last_name, salary FROM employees
WHERE salary>5000)
LOOP
DBMS_OUTPUT.PUT_LINE(emp_record.employee_id || ' ' || emp_record.last_name ||
' ' || emp_record.salary);
END LOOP;
END;
/
```

Dengan menggunakan FOR **SUBQUERY** kita tidak perlu lagi mendeklarasikan cursor.

UPDATE dengan CURSOR

Salah satu kemampuan cursor yaitu cursor dapat menunjuk sekaligus mengUpdate banyak data di dalam database. Untuk melakukan proses tersebut kita membutuhkan keyword UPDATE dan keyword WHERE CURRENT OF.

Keyword Update, digunakan untuk mendefinisikan apa tujuan dari cursor tersebut dibuat, sedangkan WHERE CURRENT OF digunakan untuk menunjukan data yang mana yang ingin di UPDATE. Satu hal yang perlu kita ketahui, keyword UPDATE disini bukan hanya sebatas untuk merubah data menjadi data yang baru, tetapi juga termasuk UPDATE untuk menghilangkan atau menghapus data.

contoh **UPDATE**:

EMPLOYEE_ID	SALARY	DEPARTMENT_ID
201	13000	20
202	7900	20

data sebelum di update

```

DECLARE
CURSOR emp_cursor IS
SELECT employee_id FROM employees
WHERE department_id = 20

FOR UPDATE OF Salary;
empId employees.employee_id%TYPE;
BEGIN
FOR emp_upd IN emp_cursor
LOOP
UPDATE employees
SET salary = 5000
WHERE CURRENT OF emp_cursor;
END LOOP;
COMMIT;
END;
/

```

EMPLOYEE_ID	SALARY	DEPARTMENT_ID
201	5000	20
202	5000	20

data setelah di update

Penjelasan :

Cursor diatas digunakan untuk merubah salary pada tabel employees menjadi 5000, dimana salary yang dirubah adalah karyawan yang employee_id nya terdapat pada department_id = 20

contoh **DELETE** :

```

DECLARE
CURSOR emp_cursor IS
SELECT employee_id FROM employees
WHERE department_id = 20
FOR UPDATE OF Salary;
empId employees.employee_id%TYPE;
BEGIN

```

```

FOR emp_upd IN emp_cursor
LOOP
DELETE FROM employees
WHERE CURRENT OF emp_cursor;
END LOOP;
COMMIT;
END;
/

```

Contoh Soal Praktikum

1. Buatlah statement **PL/SQL** dengan menggunakan **cursor** utk menampilkan employee_id, department_id, salary, dan rata-rata salary per departemen maksimal 10 baris. Dan Tampilkan juga berapa rows yg ditampilkan.

```

SET SERVEROUTPUT ON

```

```

DECLARE

```

```

emp employees.employee_id%type;

```

```

dep employees.department_id%type;

```

```

sal employees.salary%type;

```

```

avsal employees.salary%type;

```

```

CURSOR emp_cursor

```

```

IS

```

```

SELECT employee_id, salary, department_id,(select avg(salary) FROM employees

```

```

WHERE department_id= b.department_id) FROM employees b;

```

```

BEGIN

```

```

OPEN emp_cursor;

```

```

LOOP

```

```

FETCH emp_cursor INTO emp, sal, dep, avsal;

```

```

dbms_output.put_line('emp id: ' || emp || ' ' || 'department id : ' || dep || ' ' || 'salary : '

```

```

|| sal || ' ' || 'average salary : ' || avsal);

```

```

EXIT WHEN emp_cursor%ROWCOUNT = 10;

```

```

END LOOP;

```

```

dbms_output.put_line('jumlah baris= ' || emp_cursor%ROWCOUNT);

```

```

CLOSE emp_cursor;

```

```

end;

```

```

/

```


2. Tampilkanlah first name, last name, dan salary dari tabel employees dengan menggunakan cursor yang memiliki sebuah parameter. Kemudian Tampilkan data karyawan tersebut dimana karyawan tersebut memiliki salary yang nilainya lebih kecil dari salary dari parameter cursor.

DECLARE

CURSOR cur_emps(salary_param NUMBER) **IS**

SELECT first_name, last_name, salary

FROM employees **WHERE** salary < salary_param;

f_name employees.first_name%**type**;

l_name employees.last_name%**type**;

gaji employees.salary%**type**;

BEGIN

OPEN cur_emps(2500);

LOOP

FETCH cur_emps **INTO** f_name, l_name, gaji;

EXIT WHEN cur_emps%NOTFOUND;

DBMS_OUTPUT.PUT_LINE(f_name || ' ' || l_name || ' memiliki gaji: ' || gaji);

END LOOP;

CLOSE cur_emps;

END;

/

Hasil jika nilai parameter=2500

Soal Praktikum

1. Buatlah sebuah cursor yang digunakan untuk mengUPDATE salary karyawan menjadi 2 kali gaji awal dimana id karyawan tersebut adalah 109. Kemudian tampilkan pesan 'Data Telah diUpdate' ketika data telah berhasil diUPDATE (**gunakan PARAMETER!**) dan tampilkan pesan 'Karyawan tidak ditemukan' ketika employee_id nya tidak terdapat dalam **database**.

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta



MODUL PERKULIAHAN

Pemrogramman Sistem Basis Data (Oracle)

Pertemuan 14 Application Builder

Fakultas

Ilmu Komputer

Program Studi

Teknik Informatika

Tatap Muka

14

Kode MK

87043

Disusun Oleh

Tim Dosen

Abstract

Memahami tentang pembuatan aplikasi dengan Application Builder pada Oracle

Kompetensi

Mampu memahami tentang pembuatan aplikasi dengan Application Builder pada Oracle

Application Builder

Kali ini kita akan membahas cara membangun *Application Builder* dengan Oracle 11g XE. Oracle menyediakan fasilitas *Application Builder* untuk membuat aplikasi *database* tanpa menggunakan *Coding*.

Langkah 1

Buatlah sebuah *workspace* dan tabel baru, dalam hal ini saya menggunakan tabel yang saya buat di tutorial sebelumnya. Untuk lebih jelasnya dapat dilihat pada tutorial MEMBUAT TABLE DI ORACLEXE 11.

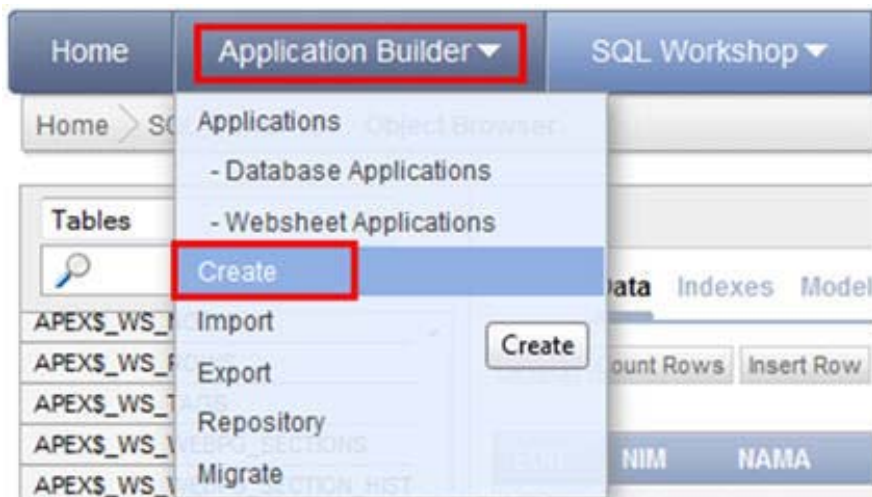


Table	Data	Indexes	Model	Constraint
Query	Count Rows	Insert Row		
EDIT	NIM	NAMA	ALAMAT	
	13001	Septiano	Palembang	
	13002	Wijaya	Plaju	
	13003	Andrei	Bukit	
row(s) 1 - 3 of 3				

Langkah 2

Selanjutnya adalah membuat *application builder* dari tabel yang kita buat.

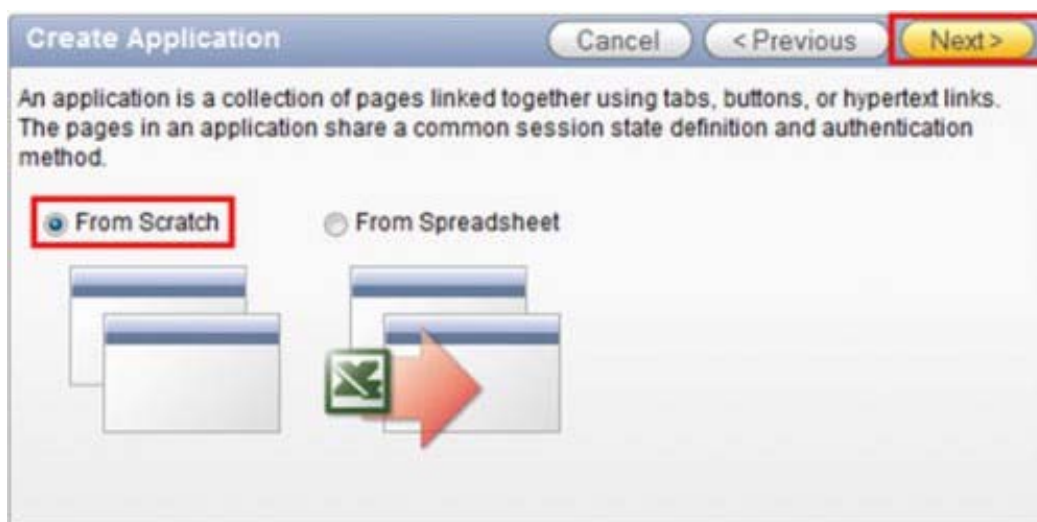
- Klik menu *Application Builder*-> *Create*.



Pada dialog dialik pilih *Database*->klik *Next*



Klik pilihan *Form Scratch* -> klik *next*.



Masukkan nama *Application* (unik) yang akan anda buat ->klik *next*.

Create Application

Cancel < Previous Next >

Enter an application name and an unique application ID. Then, select an application creation method and a schema.

Name APLIKASI_NASRUL

Application 104

Create Application: ☒ From scratch ☐ Based on existing application design model

Schema DB_NASRUL

Langkah 3

Klik pilhan *Report and Form*, untuk membuat aplikasi dalam bentuk *form* dan laporan. *Browse* tabel yang akan dibuatkan aplikasinya -> klik *Next*->klik *Add Page*.

Create Application

Cancel < Previous

Add pages to your application by selecting a page type and clicking Add Page.

Add Page

Select Page Type:

☐ Blank ☐ Report ☐ Form ☒ Report and Form

☐ Tabular Form ☐ Master Detail ☐ Chart

Action: Add a report with an edit form on a second page

Table Name MHS2

Implementation Interactive

- Klik *Create* untuk langsung konfirmasi akhir, Atau klik *Next* untuk lanjut ke langkah selanjutnya. Klik pilihan *One Level of Tabs*-> klik *Next*

Create Application

Cancel < Previous Next >

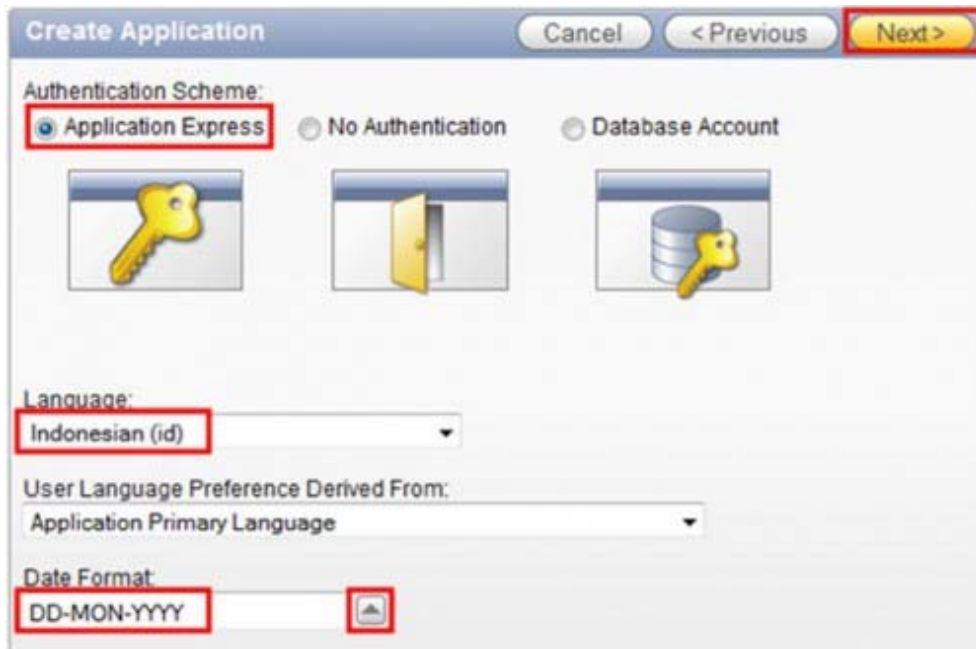
Application: 104

Name: APLIKASI_NASRUL

Tabs:

☐ No Tabs ☒ One Level of Tabs ☐ Two Levels of Tabs

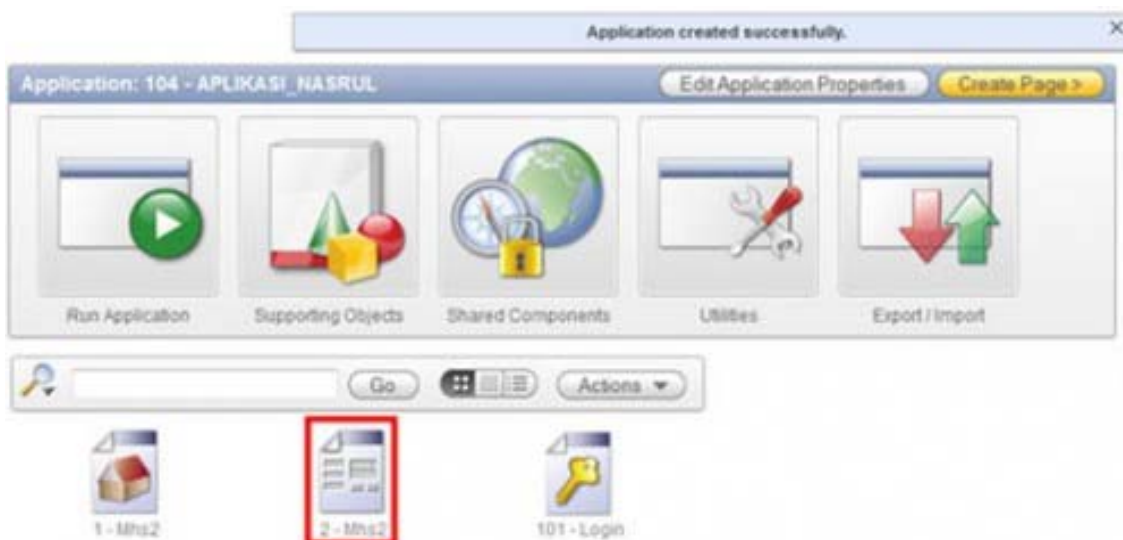
- Selanjutnya pilih No ->klik *Next*. Pada pilihan *Application Express*-> pilih bahasa yang anda kehendaki ->*browse* Data Format-> klik *Next*.



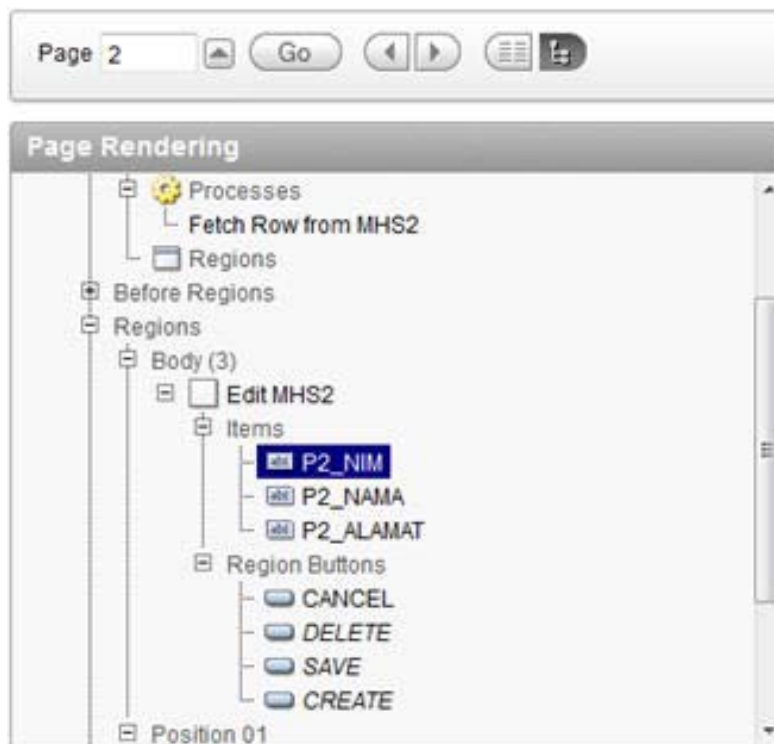
- Pada pemilihan tema tampilan, pilih tema yang anda inginkan ->klik *Next*. Klik *Create* untuk konfirmasi akhir.

Langkah 4

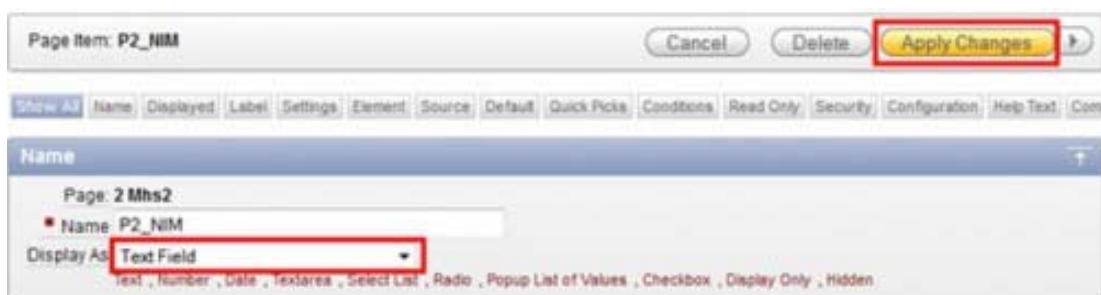
Klik *icon Form* dengan nama tabel yang kita buatkan aplikasinya.



- Double klik *field* NIM untuk diedit agar dapat ditampilkan dalam *Form*.



- Ubah *Display Field* NIM menjadi *Text Field* ->klik *Apply Changes*.



Kemudian pilih run

Langkah 5

Jika diminta login, masukkan *username* dan *password* yang anda gunakan untuk masuk ke dalam *Workspace*



- *Application Builder* yang kita buat telah selesai.

Langkah 6

Klik menu nama_tabel untuk melihat isi data.

The screenshot shows the 'Mhs2' application interface. At the top, there is a red button labeled 'Mhs2'. Below it, there are two buttons labeled 'Mhs2', with the first one highlighted by a red box. The main part of the interface is a form titled 'Edit MHS2'. The form contains three input fields: 'Nim', 'Nama', and 'Alamat', each with a corresponding label to its left.

- Klik *button Create* untuk menambahkan data ke dalam *form*

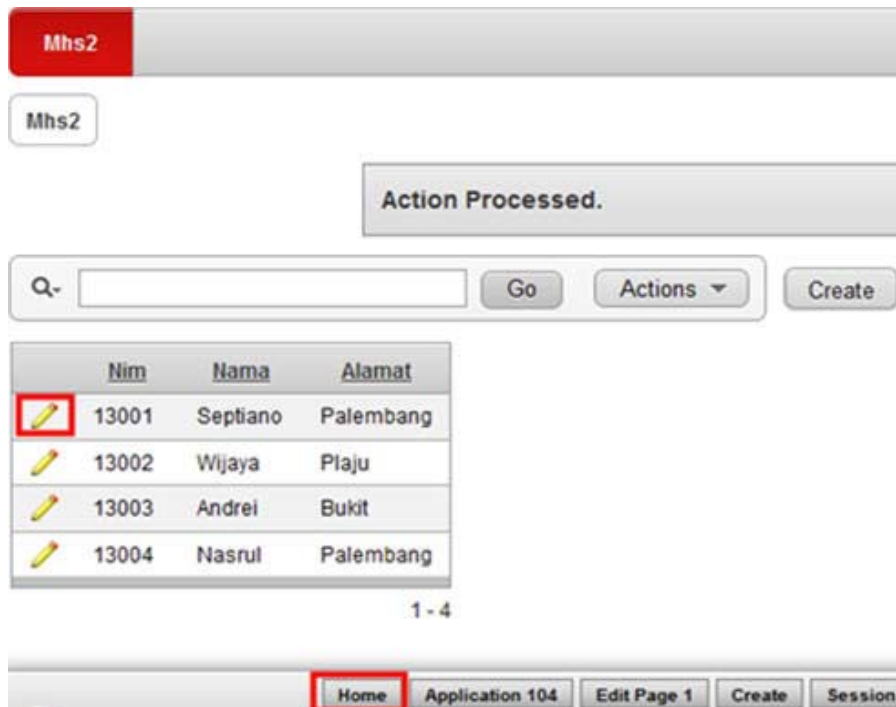
The screenshot shows the 'Mhs2' application interface. At the top, there is a red button labeled 'Mhs2'. Below it, there is a button labeled 'Mhs2'. The main part of the interface is a table with three columns: 'Nim', 'Nama', and 'Alamat'. The table contains three rows of data. To the right of the table, there is a 'Create' button highlighted by a red box. Below the table, there is a pagination indicator '1 - 3'.

	Nim	Nama	Alamat
	13001	Septiano	Palembang
	13002	Wijaya	Plaju
	13003	Andrei	Bukit

- Masukkan data baru -> klik *Create*

The screenshot shows the 'Mhs2' application interface. At the top, there is a red button labeled 'Mhs2'. Below it, there is a button labeled 'Mhs2'. The main part of the interface is a form titled 'Edit MHS2'. The form contains three input fields: 'Nim', 'Nama', and 'Alamat', each with a corresponding label to its left. The 'Nim' field contains the value '13004', the 'Nama' field contains the value 'Nasrul', and the 'Alamat' field contains the value 'Palembang'. To the right of the form, there are two buttons: 'Cancel' and 'Create', with the 'Create' button highlighted by a red box.

- Klik *icon* pensil untuk mengedit data, atau Klik menu *Home* untuk kembali ke halaman awal Oracle *Express Edition*.



Soal Praktikum

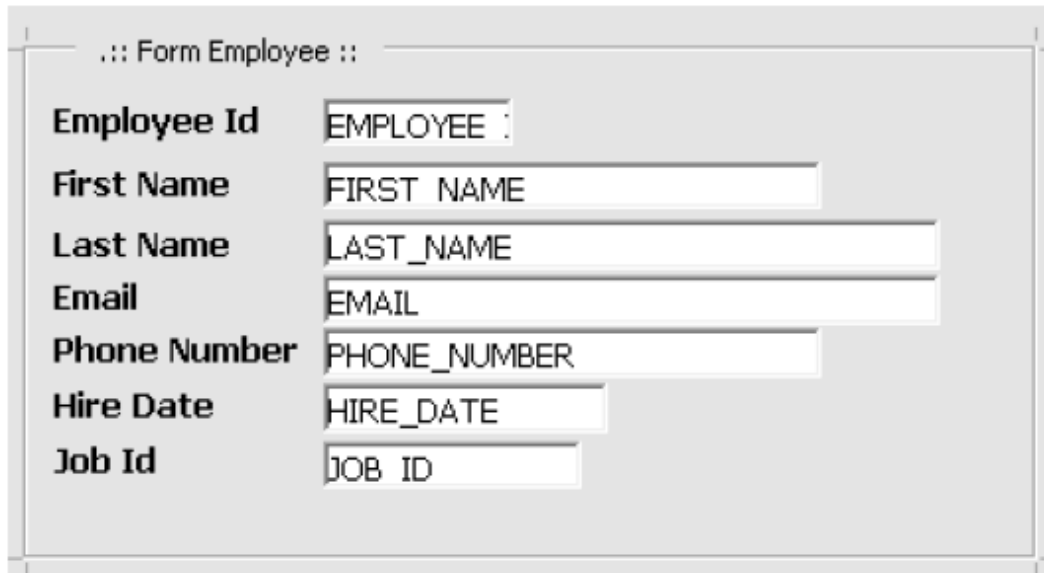
Soal 1

Lengkapi dan perbaikilah PL/SQL di bawah ini, apa output yang akan ditampilkan?

```
DECLARE
    x employees.employee_id%****;
    y employees.first_name%****;
BEGIN
    select employee_id, first_name into x,y
    from employees
    where employee_id=:***** and
    first_name=:*****;
    message('data ada');
EXCEPTION WHEN no_data_found THEN
    message('data tidak ada');
END;
```

Soal 2

Buatlah sebuah form dengan menggunakan Base Table yang mengarah ke table Employees, gunakan hanya field employee_id, first_name, last_name, email, phone_number, hire_date dan job_id. Sehingga akan menghasilkan tampilan di bawah ini



Label	Placeholder Text
Employee Id	EMPLOYEE :
First Name	FIRST_NAME
Last Name	LAST_NAME
Email	EMAIL
Phone Number	PHONE_NUMBER
Hire Date	HIRE_DATE
Job Id	JOB ID

Daftar Pustaka

Bambang Sutejo, Sukses Sertifikasi OCP, PT. Elex Media Komputindo, 2010, Jakarta