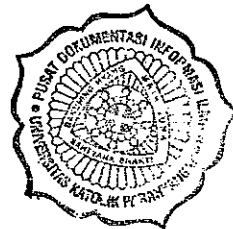


Slide Basis Data dengan Oracle 9i



005. 3
ADH
S

Disusun oleh :
Lucky Adhie, SKom



R/
GC340 SB / PM
13. 9. 05.

Universitas Katolik Parahyangan
Fakultas Matematika Dan Ilmu Pengetahuan Alam
Jurusan Ilmu Komputer
Bandung
Ganjil 2003/2004

DATABASE

Chapter I INTRODUCTION

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Purpose of Database System

- In the early days, database applications were built on top of file systems
- Drawbacks of using file systems to store data:
 - Data redundancy and inconsistency
 - Multiple file formats, duplication of information in different files
 - Difficulty in accessing data
 - Need to write a new program to carry out each new task
 - Data isolation — multiple files and formats
 - Integrity problems
 - Integrity constraints (e.g. account balance > 0) become part of program code
 - Hard to add new constraints or change existing ones

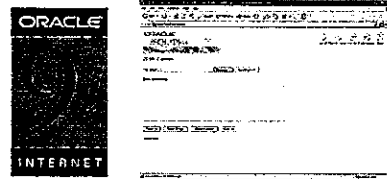
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Purpose of Database Systems (Cont.)

- Drawbacks of using file systems (cont.)
 - Atomicity of updates
 - Failures may leave database in an inconsistent state with partial updates carried out
 - E.g. transfer of funds from one account to another should either complete or not happen at all
 - Concurrent access by multiple users
 - Concurrent access needed for performance
 - Uncontrolled concurrent accesses can lead to inconsistencies
 - E.g. two people reading a balance and updating it at the same time
 - Security problems
- Database systems offer solutions to all the above problems


Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

ORACLE 9i




Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Oracle 9i Application Server

A	PORTALS	
P	Transactional Apps	
A	Business Intelligence	
C	Integration	
H		
E		

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Oracle 9i Database

	Object Relational Data
	Documents
	Multimedia
	Messages

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Database Management System (DBMS)

- Collection of interrelated data
- Set of programs to access the data
- DBMS contains information about a particular enterprise
- DBMS provides an environment that is both *convenient* and *efficient* to use.
- Database Applications:
 - Banking: all transactions
 - Airlines: reservations, schedules
 - Universities: registration, grades
 - Sales: customers, products, purchases
 - Manufacturing: production, inventory, orders, supply chain
 - Human resources: employee records, salaries, tax deductions
- Databases touch all aspects of our lives

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

SQL

- SQL = Structured Query Language
- To access the database
- To modified database
- Contains a large set of operators for partitioning and combining relations

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

SQL

- SQL: widely used non-procedural language
 - E.g. find the name of the customer with customer-id 192-83-7465


```
select customer.customer-name
from customer
where customer.customer-id = '192-83-7465'
```
 - E.g. find the balances of all accounts held by the customer with customer-id 192-83-7465


```
select account.balance
from depositor, account
where depositor.customer-id = '192-83-7465' and
depositor.account-number = account.account-number
```
- Application programs generally access databases through one of
 - Language extensions to allow embedded SQL
 - Application program interface (e.g. ODBC/JDBC) which allow SQL queries to be sent to a database

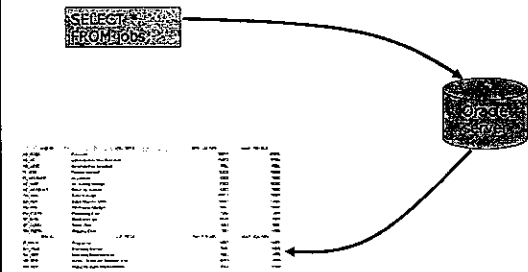
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

SQL Advantages

- Efficient
- Easy to learn and use
- Functionally completed (define, retrieve, and manipulate data in the tables)

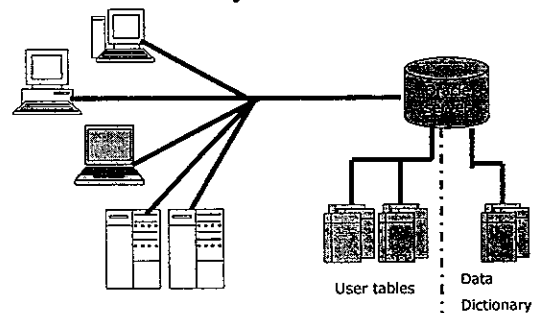
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

SQL Processing



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Relational Database Management System



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

DATABASE

Chapter II Writing SQL Statement & Creating Tables

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Writing SQL Statement

- SQL statement are not case sensitive
- SQL statement can be one or more lines
- Keywords cannot be abbreviated or split across lines
- Clauses are usually placed on separate lines
- Indents are used to enhance readability

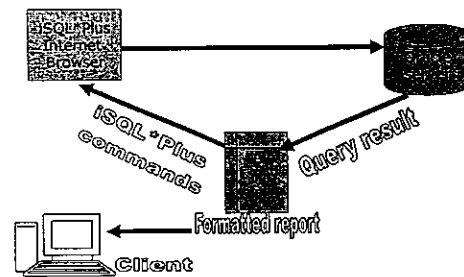
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Column Heading Default

- iSQL*Plus :
 - Default heading justification : Center
 - Default heading display : Uppercase
- SQL Plus :
 - Character and Date column heading are left-justified.
 - Number column heading are right-justified.
 - Default heading display : Uppercase

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

SQL and iSQL*Plus Interaction



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Features of SQL

- Can be used by a range of user, including those with little or no programming experience
- Is a nonprocedural language
- Reduce the amount of time required for creating and maintaining systems
- Is an English-like language

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Features of iSQL*Plus

- Accessed from a browser
- Accepts ad hoc entry statement
- Provides online editing for modifying SQL statement
- Controls environmental setting
- Format query result into a basic report
- Access local and remote database

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.



SQL VS iSQL*Plus

SQL

- A Language
- ANSI Standard
- Keyword cannot be abbreviated
- Statement manipulate data and table definitions in the database

iSQL Plus

- An environment
- Oracle proprietary
- Keyword can be abbreviated
- Commands do not allow manipulation of values in the database
- Run on the browser
- Centrally loaded, doesn't have to be implemented on each machine

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

iSQL*Plus

After you log into iSQL*Plus, you can :

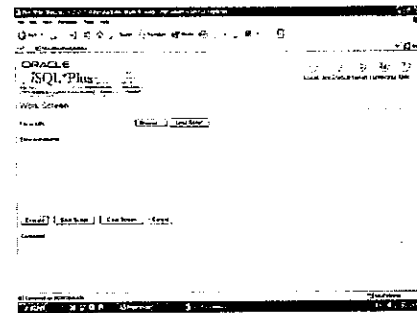
- Describe the table structure
- Edit your SQL statement
- Execute SQL from iSQL*Plus
- Save SQL statement to files and append SQL statement to files
- Execute statements stored in saved files
- Load commands from a text file into the iSQL*Plus edit windows

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Logging into iSQL*Plus

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The iSQL*Plus Environment



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Displaying Table Structure

Use the iSQL*Plus DESCRIBE command to display the structure of a table.

```
DESCRIBE <tablename>
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Database Objects

OBJECT	DESCRIPTION
Table	Basic unit of storage; composed of rows and columns
View	Logically represents subset of data from one or more tables
Sequence	Numeric value generator
Index	Improves the performance of some queries
Synonym	Give alternative names to object

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The ALTER TABLE Statement

Use the ALTER TABLE statement to :

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify, or drop columns.

```
ALTER TABLE table  
ADD (column datatype [DEFAULT expr]  
[, column datatype] ...)
```

```
ALTER TABLE table  
MODIFY (column datatype [DEFAULT expr]  
[, column datatype] ...)
```

```
ALTER TABLE table  
DROP (column)
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Dropping a Table

- All data and structure in the table is deleted
- Any pending transactions are committed
- All indexes are dropped
- You cannot roll back the DROP TABLE statement

```
DROP TABLE table  
Table dropped.
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Changing the Name of an Object

- To change the name of a table, view, sequence, or synonym, you execute the RENAME statement

```
RENAME source table to destination table  
Table renamed.
```

- You must be the owner of the object

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Truncating a Table

- The TRUNCATE TABLE statement :
 - Removes all row from a table
 - Releases the storage space used by that table

```
TRUNCATE TABLE tablename  
Table truncated.
```

- You cannot roll back removal when using TRUNCATE.
- Alternatively, you can remove rows by using the DELETE statement.

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The NOT NULL Example

```
CREATE TABLE mahasiswa (
  npm NUMBER(6),
  nama VARCHAR2 (8) NOT NULL,
  email VARCHAR2 (8) CONSTRAINT mhs_email_nn
  NOT NULL ,
  telp NUMBER(5),
  alamat VARCHAR2 (8));
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The UNIQUE Constraint

■ Mahasiswa

UNIQUE
↓

NPM	NAMA	EMAIL	TELP	ALAMAT
730201	Saya	Iam	11111	Jl. ABCDE
730202	Kamu	you		JL.FGHIJK

↑ NOT ALLOWED

730203	AKU	Iam	11333	Jl. fsdfs
--------	-----	-----	-------	-----------

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

THE UNIQUE Sample

```
CREATE TABLE mahasiswa (
  npm NUMBER(6),
  nama VARCHAR2 (8) NOT NULL,
  email VARCHAR2 (8),
  telp NUMBER(5),
  alamat VARCHAR2 (8),
  CONSTRAINT mhs_email_uk
  UNIQUE (email));
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The PRIMARY KEY Constraint

NPM	NAMA	EMAIL	TELP	ALAMAT
730201	Saya	Iam	11111	Jl. ABCDE
730202	Kamu	you		JL.FGHIJK

↑
PRIMARY KEY

PRIMARY KEY include -> UNIQUE and NOT NULL

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The PRIMARY KEY Example

```
CREATE TABLE mahasiswa (
  npm NUMBER(6),
  nama VARCHAR2 (8) NOT NULL,
  email VARCHAR2 (8),
  telp NUMBER(5),
  alamat VARCHAR2 (8),
  CONSTRAINT mhs_npm_pk PRIMARY KEY (npm)
);
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The FOREIGN KEY Constraint

NPM	NAMA	EMAIL	TELP	ALAMAT
730201	Saya	Iam	11111	Jl. ABCDE
730202	Kamu	you		JL.FGHIJK

↙

NPM	JURUSAN	FAKULTAS	KODE
730201	Saya	Iam	73
730202	Kamu	you	73

FOREIGN KEY

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

THE FOREIGN KEY Example

```
CREATE TABLE jurusan (  
  npm NUMBER(6),  
  jurusan VARCHAR2 (8) NOT NULL,  
  fakultas VARCHAR2 (8) NOT NULL,  
  kode NUMBER(5) NOT NULL,  
  CONSTRAINT jrs_npm_fk FOREIGN KEY (npm)  
  REFERENCES mahasiswa(npm)  
);
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

FOREIGN KEY Constraint Keyword

- FOREIGN KEY : Defines the column in the child table at the table constraint level
- REFERENCES : identifies the table and column in the parent table
- ON DELETE CASCADE : Deletes the dependent rows in the child table when a row in the parent table is deleted
- ON DELETE SET NULL : Converts dependent foreign key values to null

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The CHECK Constraint

- Defines a condition that each row must satisfy
- Not Allowed :
 - CURRVAL, NEXTVAL, LEVEL, and ROWNUM pseudocolumns

```
CREATE TABLE emp (  
  salary NUMBER(7),  
  CONSTRAINT emp_sal_min  
  CHECK (salary > 0))
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Adding a Constraint Syntax

Use the ALTER TABLE statement to :
Add or drop a constraint, but not modify its structure

- Enable or disable constraints
- Add a NOT NULL constraint by using the MODIFY clause

```
ALTER TABLE table  
ADD CONSTRAINT constraint_type (column)
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Dropping a Constraint

```
ALTER TABLE mahasiswa  
DROP CONSTRAINT mns_email_idx
```

```
ALTER TABLE mahasiswa  
DROP PRIMARY KEY CASCADE
```

```
ALTER TABLE mahasiswa  
DISABLE CONSTRAINT mns_email_uk
```

```
ALTER TABLE mahasiswa  
ENABLE CONSTRAINT mns_email_uk
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

DATABASE

Chapter IV Writing SQL SELECT Statement

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Capabilities of SQL SELECT Statement

- Projection : To choose the columns in table that you want returned by your query
- Selection : To choose the row in a table that you want returned by your query
- Joining : To bring together data that is stored in different tables by creating a link between them.

Basic SELECT Statement

```
SELECT [DISTINCT] column | Expression | Alias  
FROM table
```

- SELECT identifies what columns
- FROM identifies which table

Selecting All Column

```
SELECT  
FROM department
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	205	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	202	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	106	1700
110	Accounting	203	1700
120	Treasury	205	1700

Selecting Specific Columns

```
SELECT department_id and location_id  
FROM departments
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
30	1700
40	2400
50	1500
60	1400
70	2700
80	2500
90	1700
100	1700
110	1700
120	1700

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators

OPERATOR	Description
+	Add
-	Subtract
*	Multiply
/	Devide

Using Arithmetic Operators

```
SELECT last_name, salary, salary * 30
FROM employees;
```

LAST_NAME	SALARY	SALARY*30
King	24000	720000
Neenan	17000	510000
De Haan	17000	510000
Hunold	9000	270000
Giet	9000	270000
Austin	4900	147000
Pavane	4900	147000
Ullrich	4200	126000
Crowley	12000	360000
Fay	9000	270000
Chen	8200	246000

Operator Precedence

- Multiplication and division take priority over addition and subtraction
- Operators of the same priority are evaluated from left to right.
- Parentheses are used to force prioritized evaluation and to clarify statements.

Operator Precedence

```
SELECT last_name, salary, 12 * salary + 20
FROM employees;
```

LAST_NAME	SALARY	12*SALARY+20
King	24000	288000
Neenan	17000	204000
De Haan	17000	204000
Hunold	9000	108000
Giet	9000	108000
Austin	4900	58800
Pavane	4900	58800
Ullrich	4200	50400
Crowley	12000	144000
Fay	9000	108000
Chen	8200	98400
Sims	7100	85200
Liman	7100	85200

Using Parentheses

```
SELECT last_name, salary, (12 * salary + 100)
FROM employees;
```

LAST_NAME	SALARY	(12*SALARY+100)
King	24000	288000
Neenan	17000	204000
De Haan	17000	204000
Hunold	9000	108000
Giet	9000	108000
Austin	4900	58800
Pavane	4900	58800
Ullrich	4200	50400
Crowley	12000	144000
Fay	9000	108000
Chen	8200	98400
Sims	7100	85200
Liman	7100	85200

Defining a Null Value

- A null is a value that is unavailable, unassigned, unknown, or inapplicable
- A null is not the same as zero or a blank space

```
SELECT last_name, job_id, salary, commission_pct
FROM employees;
```

- Commission_pct column include null

Null Values in Arithmetic Expressions

- Arithmetic expressions containing a null value evaluate to null.

```
SELECT last_name, 12 * salary * commission_pct
FROM employees;
```

Eliminating Duplicate Rows

- Eliminate duplicate rows by using the **DISTINCT** keyword in the **SELECT** clause

```
SELECT DISTINCT department_id  
FROM employees
```

DATABASE

Chapter V RESTRICTING AND SORTING DATA

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Limiting Row using a Selection

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD PRES	90
101	De Haan	AD VP	90
102	De Haan	AD VP	90
103	Hunold	IT PRG	90
104	Baer	IT PRG	90
105	Whalen	IT PRG	90
106	Whalen	IT PRG	90

"Retrieve all Employees
In department 90"

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD PRES	90
101	De Haan	AD VP	90
102	De Haan	AD VP	90

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Limiting the Rows Selected

- Restrict the rows returned by using the WHERE clause.

```
SELECT DISTINCT column/expression (alias)
FROM table
[WHERE condition(s)]
```

- The WHERE clause follows the FROM clause.

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the WHERE Clause

```
SELECT employee_id, last_name, job_id, department_id
FROM employees
WHERE department_id = 90;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD PRES	90
101	De Haan	AD VP	90
102	De Haan	AD VP	90

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Character Strings and Dates

- Character strings and date values are enclosed in single quotation marks.
- Character values are case sensitive, and date values are format sensitive.
- The default date format is DD-MON-RR.

```
SELECT employee_id, job_id, department_id
FROM employees
WHERE last_name = 'Whalen';
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Comparison Condition

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<> or !=	Not equal to

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using Comparison Condition

```
SELECT last_name, salary
FROM employees
WHERE salary < 2000;
```

LAST NAME	SALARY
Baida	2900
Tobias	2800
Himuro	2600

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Other Comparison Condition

Operator	Meaning
BETWEEN ... AND ...	Between two values (inclusive)
IN (set)	Match any of a list of values
LIKE	Match a character pattern
IS NULL	Is a null value

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using BETWEEN Condition

- Use the BETWEEN condition to display rows based on a range of values.

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```

Lower limit Upper limit

LAST NAME	SALARY
Khao	3100
Baida	2900
Tobias	2800
Himuro	2600

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the IN Condition

- Use the IN membership condition to test for values in a list.

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201);
```

EMPLOYEE ID	LAST NAME	SALARY	MANAGER ID
101	Kochhar	17000	100
102	De Haan	17000	100
108	Greenberg	12000	101
114	Raphaely	11000	100

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the LIKE Condition

- Use the LIKE condition to perform wildcard searches of valid search string values.
- Search conditions can contain either literal characters or numbers :
 - % denotes zero or many characters.
 - _ denotes one character.

```
SELECT last_name
FROM employees
WHERE last_name LIKE 'S%';
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the LIKE Condition

- You can combine pattern-matching characters.
- You can use the ESCAPE identifier to search for the actual % and _ symbols

```
SELECT last_name
FROM employees
WHERE last_name LIKE 'S%';
```

```
SELECT last_name, job_id
FROM employees
WHERE job_id LIKE '%SA%' ESCAPE '%';
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the NULL Conditions

- Test for nulls with the IS NULL operator.

```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL
```

LAST_NAME	MANAGER_ID
King	

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Logical Conditions

Operator	Meaning
AND	Return TRUE if <i>both</i> component condition are true
OR	Return TRUE if <i>either</i> component condition is true
NOT	Return TRUE if <i>the</i> following condition is false

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using AND, OR , NOT

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary > 10000
AND job_id LIKE 'MAN%'
```

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary > 10000
OR job_id LIKE 'MAN%'
```

```
SELECT last_name, job_id
FROM employees
WHERE job_id NOT IN ('PROG', 'STUDENT', 'SALER')
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Rules of Precedence

Order Evaluated	Operator
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN [NOT] BETWEEN
5	[NOT] BETWEEN
6	NOT Logical condition
7	AND Logical condition
8	OR Logical condition

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

ORDER BY Clause

- Sort rows with the ORDER BY clause
 - ASC : ascending order, default
 - DESC : descending order
- The ORDER BY clause comes last in the SELECT statement.

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Sorting in Descending Order

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date DESC
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Sorting by Column Alias

```
SELECT employee_id, last_name AS L, first_name AS F  
FROM employees  
ORDER BY L;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Sorting by Multiple Columns

- The order of ORDER BY list is the order of sort.

```
SELECT last_name, department_id AS D, salary  
FROM employees  
ORDER BY D, salary DESC;
```

- You can sort by a column that is not in the SELECT list.

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

DATABASE

Chapter VI SINGLE ROW FUNCTION

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

SQL Functions

Function are very powerful feature of SQL and can be used to do the following :

- Perform calculation on data
- Modify individual data items
- Manipulate output for groups of rows
- Format dates and numbers for display
- Convert column data types

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Two Types of SQL Functions

- Single-row functions operate on single rows only and return one result per row.
- Multiple-row functions can manipulate groups of rows to give one result per group of rows

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Single-Row Function

- Single row functions :
 - Manipulate data items
 - Accept arguments and return one value
 - Act on each row returned
 - Return one result per row
 - May modify the data type
 - Can be nested
 - Accept arguments witch can be a column or an expression

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Character Functions

LOWER(<i>column/expression</i>)	Convert alpha character values to lowercase
UPPER(<i>column/expression</i>)	Convert alpha character values to uppercase
INITCAP(<i>column/expression</i>)	Convert alpha character values to uppercase for the 1 st letter of each word, all other in lowercase
CONCAT(<i>column1/expression1</i> , <i>column2/ expression2</i>)	Concatenates the first character value to the second character value; equivalent to concatenation operator ()
SUBSTR(<i>column/expression</i> , <i>m[,n]</i>)	Returns specified characters from character value string at character position <i>m</i> , <i>n</i> character long

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Character Functions

LOWER(<i>column/expression</i>)	Convert alpha character values to lowercase
UPPER(<i>column/expression</i>)	Convert alpha character values to uppercase
INITCAP(<i>column/expression</i>)	Convert alpha character values to uppercase for the 1 st letter of each word, all other in lowercase
CONCAT(<i>column1/expression1</i> , <i>column2/ expression2</i>)	Concatenates the first character value to the second character value; equivalent to concatenation operator ()
SUBSTR(<i>column/expression</i> , <i>m[,n]</i>)	Returns specified characters from character value string at character position <i>m</i> , <i>n</i> character long

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.



Case Manipulation Functions

Function	Result
LOWER('SQL Course')	sql course
UPPER('SQL Course')	SQL COURSE
INITCAP('SQL Course')	Sql Course

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Character-Manipulation Functions

Function	Result
CONCAT('Hello','World')	HelloWorld
SUBSTR('HelloWorld', 1, 5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000
RPAD(salary,10,'*')	24000*****
TRIM('H' FROM 'HelloWorld')	elloWorld

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Number Function

FUNCTION	Purpose
ROUND(column expression, n)	Rounds the column expression, or value to n decimal places, or, if n is omitted, no decimal places.
TRUNC(column expression, n)	Truncate the column, expression, or value to n decimal places, or if n is omitted then n defaults zero
MOD (m, n)	Returns the remainder of m divided by n

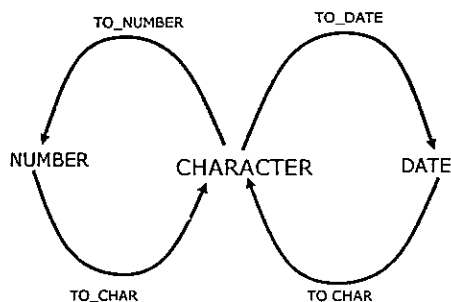
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Dates Functions

Function	Description
MONTHS_BETWEEN (date1, date2)	Number of months between two dates
ADD_MONTHS (date, n)	Add calendar months to date
NEXT_DAY(date, 'char')	Next day of the date specified
LAST_DAY(date)	Last day of the month
ROUND(date[, 'fmt'])	Round date
TRUNC(date [, 'fmt'])	Truncate date

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Data Type Conversion



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

General Function

Function	Description
NVL (expr1, expr2)	Convert a null value to an actual value
NVL2 (expr1, expr2, expr3)	If expr1 is not null, NVL2 return expr2, If expr1 is null, NVL2 return expr3
NULLIF (expr1, expr2)	Compares two expressions and return null if they are equal or 1 st expression if not equal
COALESCE(expr1, expr2, ..., exprn)	Returns the first non-null expression list

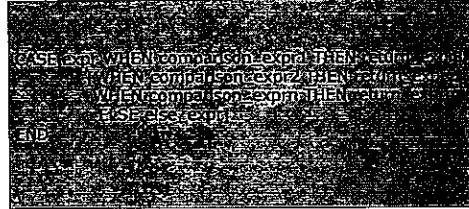
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Conditional Expression

- CASE expression
- DECODE function

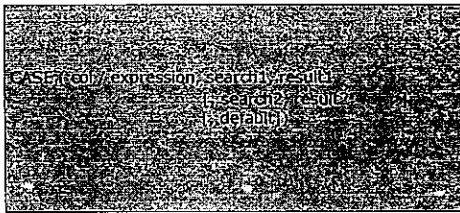
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The CASE Expressions



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The DECODE Function



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

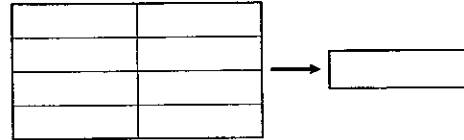
DATABASE

Chapter VIII AGGREGATING DATA USING GROUP FUNCTIONS

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

What Are Group Functions

Group functions operate on sets of rows to give one result per group



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Types of group Functions

Function	Description
AVG (([DISTINCT][ALL] n)	Average value of n, ignoring null values
COUNT (([*][DISTINCT][ALL] expr))	Number of rows, where expr evaluates to something other than null(count all selected rows using *, including duplicates and rows with nulls
MAX(([DISTINCT][ALL] expr)	Maximum value of expr, ignoring null values
MIN (([DISTINCT][ALL] expr)	Minimum value of expr, ignoring null values
STDDEV (([DISTINCT][ALL] x)	Standard deviation of n, ignoring null values
SUM(([DISTINCT][ALL] n)	Sum values of n, ignoring null values
VARIANCE (([DISTINCT][ALL] n)	Variance of n, ignoring null values

Group Functions Syntax

```
SELECT (column) group_function (column)
FROM table
WHERE condition
(GROUP BY column)
ORDER BY column
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the AVG, SUM, MAX, MIN Functions

```
SELECT AVG(salary), MAX(salary),
MIN(salary), SUM(salary)
FROM emp;
WHERE 0 <= emp.sal <= 5000;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the COUNT Functions

```
SELECT COUNT(*)
FROM emp;
WHERE department_id = 50;
```

- COUNT (expr) returns the number of rows with non-null values for the expr.
- Display the number of department values in the EMPLOYEES table, excluding the null values.

```
SELECT COUNT(commission_pct)
FROM emp;
WHERE department_id = 50;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the DISTINCT Keyword

- COUNT (DISTINCT expr) returns the number of distinct non-null values of the expr.
- Display the number of distinct department values in the EMPLOYEES table.

```
SELECT COUNT(DISTINCT department_id)
FROM employees;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Group Functions and Null Values

Group functions ignore null values in the column.

```
SELECT AVG(commission_pct)
FROM employees;
```

The NVL function forces group functions to include null values.

```
SELECT AVG(NVL(commission_pct,0))
FROM employees;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Creating Groups of Data

Department_id	Salary
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2500
50	2600



Department_id	Salary
10	4400
20	9500
50	3500

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the GROUP BY Clause

All column in the SELECT list that are not in group functions must be in the GROUP BY clause.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

The GROUP BY column does not have to be in the SELECT list.

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Grouping Multiple Columns

```
SELECT department_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Illegal Queries Using Group Functions

Any column or expression in the SELECT list that is not an aggregate function must be in the GROUP BY clause.

```
SELECT department_id, COUNT(last_name)
FROM employees;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Illegal Queries Using Group Functions

- You cannot use the WHERE clause to restrict groups.
- You use the HAVING clause to restrict groups.
- You cannot use group function in WHERE clause.

```
SELECT department_id, AVG(salary)
FROM employee
GROUP BY department_id
HAVING AVG(salary) > 3000;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Excluding Group Result: The HAVING Clause

Use the HAVING clause to restrict groups :

- Rows are grouped
- The group functions applied
- Groups matching the HAVING clause are displayed

```
SELECT (column), group_function (column)
FROM (table)
WHERE (condition)
GROUP BY (group-by-expression)
HAVING (group-condition)
ORDER BY (column);
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using HAVING Clause

```
SELECT dept_id, SUM(salary) FROM emp
FROM employee
WHERE dept_id NOT LIKE 'SA%'
GROUP BY dept_id
HAVING SUM(salary) > 10000
ORDER BY SUM(salary)
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

DATABASE

Chapter IX Subqueries

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Subquery Syntax

```
SELECT column_list  
FROM table  
WHERE operator  
      (SELECT column_list  
      FROM table)
```

- The subquery (inner query) executes once before the main query
- The result of the subquery is used by the main query (outer query)

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using a Subquery

```
SELECT last_name  
FROM employees  
WHERE salary  
      (SELECT salary  
      FROM employees  
      WHERE last_name = 'Abel')
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Guidelines for Using Subqueries

- Enclose subqueries in parentheses
- Place subquery on the right side of comparison condition
- The ORDER BY clause in subquery is not needed unless you are performing Top-N analysis
- Use single-row operators with single row subqueries and use multiple-row operators with multiple-row subqueries

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Types of Subqueries

- Single-row subquery
- Multiple-row subquery

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Single-Row Subqueries

- Return only one row
- Use single-row comparison operators
=, >, >=, <, <=, <>
- Group function can be used in subqueries (return one row)

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

What is Wrong with this Statement ?

```
SELECT employee_id, last_name
FROM employees
WHERE salary < (SELECT MIN(salary)
                FROM employees
                GROUP BY department_id)
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators
IN, ANY, ALL

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using The ANY Operator

```
SELECT employee_id, last_name, job_id, salary
FROM employee
WHERE salary < ANY
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG'
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using The ALL Operator

```
SELECT employee_id, last_name, job_id, salary
FROM employee
WHERE salary < ALL
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG'
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Null Values in Subqueries

```
SELECT emp.last_name
FROM employee emp
WHERE emp.employee_id NOT IN
      (SELECT mgr.manager_id
       FROM employees mgr
       WHERE manager_id IS NOT NULL)
```

- Whenever null values are likely to be part of results set of subquery, do not use the NOT IN operator
- The NOT IN operator is equivalent to <> ALL

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Creating a Table by Using a Subquery Syntax

- Create a table and insert rows by combining the CREATE TABLE statement and the AS subquery option

```
CREATE TABLE table
(column column)
AS subquery;
```

- Match the number of specified columns to the number of subquery columns
- Define columns with column names and default values.

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Example of Creating Table Using a Subquery

```
CREATE TABLE dept30  
AS  
SELECT employee_id, last_name,  
       salary * 12 ANNUAL,  
       hire_date  
FROM   employees  
WHERE  department_id = 80;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

DATABASE

Chapter X Manipulating Data

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Data Manipulation Language

- A DML statement is executed when you :
 - Add new rows to a table
 - Modify existing rows in a table
 - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit work.

The INSERT Statement Syntax

- Add new rows to a table by using INSERT statement.

```
INSERT INTO table (column [, column ...])  
VALUES (value [, value] ...)
```

- Only one row is inserted at a time with this syntax

Copying Rows from Another Table

- Write your INSERT statement with subquery.

```
INSERT INTO emp  
SELECT FROM EMPLOYEES
```

- Do not use the VALUES clause.
- Match the number of columns in the INSERT clause to those in subquery.

The UPDATE Statement Syntax

- Modify existing rows with the UPDATE statement

```
UPDATE table  
SET column = value [, column = value] ...  
WHERE condition
```

- Update more than one row at a time, if required.

Updating Rows in a Table

- Specific row or rows are modified if you specify the WHERE clause.

```
UPDATE employees  
SET department_id = 70  
WHERE employee_id = 103
```

- All rows in the table are modified if you omit the where clause

```
UPDATE copy_emp  
SET department_id = 100
```



Updating Two Columns with a Subquery

Update employee 114's job and salary to match that of employee 205

```
UPDATE employees
SET job_id=(SELECT job_id
            FROM employees
            WHERE employee_id=205),
    salary=(SELECT salary
            FROM employees
            WHERE employee_id=205)
WHERE employee_id=114
```

Updating Row Based on Another Table

Use subqueries in UPDATE statements to update rows in a table based on values from another table

```
UPDATE emp
SET department_id=(SELECT department_id
                  FROM employees
                  WHERE employee_id=101)
WHERE emp_id=102
```

The DELETE Statement

You can remove existing rows from a table using the DELETE statement

```
DELETE FROM table
WHERE condition;
```

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause

```
DELETE FROM departments
WHERE department_name = 'Finance'
```

- All rows in the table are deleted if you omit the WHERE clause

```
DELETE FROM emp;
```

Deleting Rows Based on Another Table

Use Subqueries in DELETE statements to remove rows from a table based on values from another table.

```
DELETE FROM employees
WHERE department_id=
(SELECT department_id
 FROM departments
 WHERE department_name LIKE '%Finance%')
```

Using a Subquery in an INSERT Statement

```
INSERT INTO
  (SELECT employee_id, job_id, salary, manager_id,
   last_name, email, phone_number,
   hire_date, department_id)
FROM emp VALUES
  (9009, 'SAVP', 12000, 1000, 'TAYLOR',
   'TAYLOR@ORACLE.COM', 515.1234,
   TO_DATE('07-JUN-99', 'DD-MON-RR'),
   (SELECT department_id
    FROM employees
    WHERE emp_id=5000))
```

The MERGE Statement

- Provides the ability to conditionally update or insert data into a data base table
- Performs an UPDATE if row exists, and an INSERT if it is a new row
 - Avoids separate updates
 - Increase performance and ease of use
 - Is useful in data warehousing applications

The MERGE Statement Syntax

```
MERGE INTO table_name table_alias  
USING (table/view/subquery) alias  
ON (join_condition)  
WHEN MATCHED THEN  
  UPDATE SET  
    col1 = col1_val,  
    col2 = col2_val  
WHEN NOT MATCHED  
  THEN INSERT (column_id)  
    VALUES (column_values)
```

Merging Rows

```
MERGE INTO copy_emp c  
USING employee e  
ON (e.employee_id = c.employee_id)  
WHEN MATCHED THEN  
  UPDATE SET  
    c.first_name = e.first_name,  
    c.last_name = e.last_name  
WHEN NOT MATCHED  
  THEN INSERT VALUES (e.employee_id,  
    e.first_name)
```

DATABASE

Chapter XI Creating View

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Why Use Views ?

- To restrict data access
- To make complex query easy
- To provide data independence
- To present different views of the same data

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of Tables	One	One or More
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operation through a view	Yes	Not always

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Creating a View

- You embed a subquery within the CREATE VIEW Statement.

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW view_name  
(alias [, alias] ...)  
(Subquery)  
[WITH CHECK OPTION | CONSTRAINT constraint] |  
[WITH READ ONLY | CONSTRAINT constraint];
```

- The subquery can contain complex SELECT syntax

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Creating a View

- Create a view, EMPVU80, that contains detail of employees in department 80

```
CREATE VIEW empvu80  
AS SELECT employee_id, last_name, salary  
FROM employees  
WHERE department_id = 80;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Creating a View

- Create a view by using a column aliases in the subquery

```
CREATE VIEW salvu80 (ID, NUMBER, NAME, ANN, SALARY)  
AS SELECT employee_id, last_name, salary * 12  
FROM employees  
WHERE department_id = 80;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Modifying a View

- You can Modify the view by using CREATE OR REPLACE VIEW clause.

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Creating a Complex View

Create a complex view that contains group functions to display values from two tables.

```
CREATE OR REPLACE VIEW emp_dept AS
SELECT department_name, MIN(salary)
      AS min_salary, AVG(salary)
      AS avg_salary, department_id
FROM employees
WHERE department_id < 10
GROUP BY department_name;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Rules for Performing DML Operation on a View

- You can perform DML operation on simple views.
- You cannot remove row if the view contains the following :
 - Group functions
 - A GROUP BY clause
 - The DISTINCT keyword
 - The pseudocolumn ROWNUM keyword
 - Columns defined by expressions
 - NOT NULL columns in the base tables that are not selected by the view

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay within domain of the view by using the WITH CHECK OPTION clause.

```
CREATE OR REPLACE VIEW emp_dept2 AS
SELECT *
FROM emp_dept
WHERE department_id = 20
WITH CHECK OPTION CONSTRAINT emp_dept2_c;
```

- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Denying DML Operation

- You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition
- Any attempt to perform a DML on any row in the view results in an Oracle server error

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Denying DML Operation

```
CREATE OR REPLACE VIEW emp_dept3 AS
SELECT employee_number, employee_name, job_title
FROM employees
WHERE department_id = 10
WITH READ ONLY;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database

```
DROP VIEW view1;
DROP VIEW empv180;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Inline Views

- An inline view is a subquery with an alias (or correlation name) that you can use within a SQL statement.
- A named subquery in the FROM clause of the main query is an inline view
- An inline view is not a schema object

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Top-N analysis

- Top-N queries ask for the n largest or smallest values of a column. For example
 - What are the ten best selling product?
 - What are the ten worst selling product?
- Both largest values and smallest values sets are considered Top-N queries.

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Performing Top-N Analysis

- The high-level structure of a Top-N analysis query is :

```
SELECT (column list), ROWNUM
FROM (SELECT (column list)
      FROM table
      ORDER BY (Top-N column))
WHERE ROWNUM <= N;
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Example of Top-N Analysis

- To display the top three earner names and salaries from the EMPLOYEES table :

```
SELECT ROWNUM AS RANK, last_name, salary
FROM (SELECT last_name, salary FROM emp_ee)
ORDER BY salary DESC
WHERE ROWNUM <= 3;
```

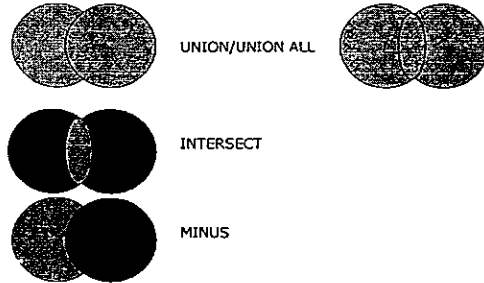
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

DATABASE

Chapter XII Using SET Operator

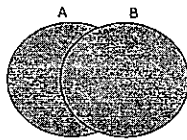
Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The SET Operators



Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The UNION Operator



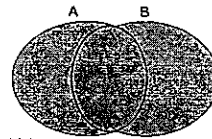
- The Union operator returns results from both queries after eliminating duplications.

Example :

```
SELECT employee_id, job_id  
FROM employees  
UNION  
SELECT employee_id, job_id  
FROM job_history
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The UNION ALL Operator



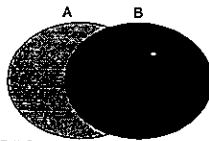
- The Union ALL operator returns results from both queries including all duplications.

Example :

```
SELECT employee_id, job_id  
FROM employees  
UNION ALL  
SELECT employee_id, job_id  
FROM job_history
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The MINUS Operator



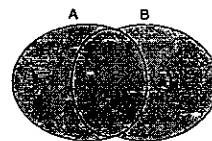
- The INTERSECT operator returns the first query that are not present in the second query

Example :

```
SELECT employee_id, job_id  
FROM employees  
MINUS  
SELECT employee_id, job_id  
FROM job_history
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The UNION ALL Operator



- The Union ALL operator returns results from both queries including all duplications.

Example :

```
SELECT employee_id, job_id  
FROM employees  
UNION ALL  
SELECT employee_id, job_id  
FROM job_history
```

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

SET Operator Guidelines

- The expressions in the `SELECT` lists must match in number and data type
- Parentheses can be used to alter the sequence of execution
- The `ORDER BY` clause :
 - Can appear only at the very end of the statement
 - Will accept the column name, aliases from the first `SELECT` statement, or positional

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

DATABASE

Chapter XIII Hierarchical Retrieval

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

Hierarchical Queries

```
SELECT LEVEL, column_expr  
FROM table  
WHERE condition(s)  
START WITH condition(s1)  
CONNECT BY PRIOR condition(s2)
```

WHERE condition :

```
expr1 condition operator expr2
```

Walking the Tree (1)

Starting Point

- Specifies the condition that must be met
- Accepts any valid condition

START WITH column1 = value

- Using the EMPLOYEES table, start with the employee whose last name is Kochhar

START WITH last_name = Kochhar

Walking the Tree (2)

CONNECT BY PRIOR column1 = column2

Walk from the top down, using
Employees table

CONNECT BY PRIOR employee_id = manager_id

DIRECTION

- Top down -> Column1 = Parent Key
Column2 = Child Key
- Bottom up -> Column1 = Child Key
Column2 = Parent Key

Walking The tree : from the Bottom Up

```
SELECT employee_id, last_name, job_id, manager_id  
FROM employees  
START WITH employee_id = 10  
CONNECT BY PRIOR manager_id = employee_id
```

Walking The tree : from the Top Down

```
SELECT last_name, empno, report_to ||>  
PRIOR last_name ||>|| Top Down  
FROM employees  
START WITH last_name = King  
CONNECT BY PRIOR employee_id = manager_id
```

Ranking Rows with LEVEL Pseudocolumn

LEVEL

VALUE	LEVEL
1	A Root node
2	A child of root node
3	A child of a child and so on

Formatting Hierarchical Reports Using LEVEL and LPAD

```
COLUMN emp_id, emp_name, emp_sal FORMAT A20  
SELECT emp_id, emp_name, emp_sal, emp_level  
FROM emp_hierarchy  
CONNECT BY PRIOR emp_id = manager_id
```

Pruning Branches

- Use WHERE Clause to eliminate a node
 - WHERE last_name != 'Higgins'
- Use the CONNECT BY clause to eliminate a branch.
 - CONNECT BY PRIOR
employee_id = manager_id
AND last_name != 'Higgins'

DATABASE

Chapter XIV
Other Data Base Object

Copyrights © Lucky Adhie S.Kom, UNPAR, 2004.

The CREATE SEQUENCE Statement Syntax

```
CREATE SEQUENCE sequence
  [INCREMENT BY n]
  [START WITH n]
  [MAXVALUE n | NOMAXVALUE]
  [MINVALUE n | NOMINVALUE]
  [CYCLE | NOCYCLE]
  [CACHE | NOCACHE]
```

EXAMPLE

```
CREATE SEQUENCE dept_deptid_seq
  INCREMENT BY 10
  START WITH 120
  MAXVALUE 9999
  NOCYCLE
  NOCACHE
```

Confirming Sequence

- Verify your sequence values in the USER_SEQUENCES data dictionary table

```
SELECT sequence_name, min_value, max_value,
       increment_by, last_number
FROM user_sequences
```

- The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified

NEXTVAL and CURRVAL

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtain the current sequence value
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

Using a Sequence

```
INSERT INTO departments (department_id,
                        department_name, location_id)
VALUES (dept_deptid_seq.NEXTVAL,
       'Support', 2500)
```

```
SELECT dept_deptid_seq.CURRVAL
FROM dual
```

