SE-204: Database Management Systems

SQL Practicals

1- Basic SQL SELECT Statements and Restricting and Sorting Data

3-Single-Row Functions

4-Displaying Data from Multiple Tables

5-Aggregating Data Using Group Functions and Subqueries

6-Manipulating Data

7-Creating and Managing Tables

8-Constraints

9- Creating Views

PL/SQL Section

10-Declaring Variables

12-Writing Executable Statements

13-Interacting with the Oracle Server

14-Writing Control Structures

Note: The following software is used: ORACLE.

Tables Used in the Course

EMPLOYEES,
DEPARTMENTS,
JOB_GRADES

Tables Used in the Course

The following main tables are used in this course:

- EMPLOYEES table, which gives details of all the employees
- DEPARTMENTS table, which gives details of all the departments
- JOB_GRADES table, which gives details of salaries for various grades

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
100	Steven	King	SKING	17-JUN-87	AD_PRES	24000	
101	Neena	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	17000	
102	Lex	De Haan	LDEHAAN	13-JAN-93	AD_VP	17000	
103	Alexander	Hunold	AHUNOLD	03-JAN-90	IT_PROG	9000	
104	Bruce	Ernst	BERNST	21-MAY-91	IT_PROG	6000	
107	Diana	Lorentz	DLORENTZ	07-FEB-99	IT_PROG	4200	
124	Kevin	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	
141	Trenna	Rajs	TRAJS	17-OCT-95	ST_CLERK	3500	
142	Curtis	Davies	CDAVIES	29-JAN-97	ST_CLERK	3100	
143	Randall	Matos	RMATOS	15-MAR-98	ST_CLERK	2600	
144	Peter	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	
149	Eleni	Zlotkey	EZLOTKEY	29-JAN-00	SA_MAN	10500	.2
174	Ellen	Abel	EABEL	11-MAY-96	SA_REP	11000	.3
176	Jonathon	Taylor	JTAYLOR	24-MAR-98	SA_REP	8600	.2
178	Kimberely	Grant	KGRANT	24-MAY-99	SA_REP	7000	.15
200	Jennifer	Whalen	JWHALEN	17-SEP-87	AD_ASST	4400	
201	Michael	Hartstein	MHARTSTE	17-FEB-96	MK_MAN	13000	
202	Pat	Fay	PFAY	17-AUG-97	MK_REP	6000	
205	Shelley	Higgins	SHIGGINS	07-JUN-94	AC_MGR	12000	
206	William	Gietz	WGIETZ	07-JUN-94	AC_ACCOUNT	8300	

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

1-Writing Basic SQL SELECT Statements

Objectives

After completing this lesson, you should be able to do the following:

• List the capabilities of SQL SELECT statements
• Execute a basic SELECT statement
• Differentiate between SQL statements and iSQL*Plus commands

Capabilities of SQL SELECT Statements

Projection Selection Join

A SELECT statement retrieves information from the database. Using a SELECT statement, you can do the following:

• Projection: You can use the projection capability in SQL to choose the columns in a table that you want returned by your query. You can choose as few or as many co lumns of the table as you require.

• Selection: You can use the selection capability in SQL to choose the rows in a table that you wantreturned by a query. You can use various criteria to restrict the rows that you see.

• Joining: You can use the join capability in SQL to bring together data that is stored in different tables by creating a link between them. You learn more about joins in a later lesson.

Basic SELECTStatement

SELECT *|{[DISTINCT] column | expression [alias],...} FROM table;

•SELECT identifies what columns

• FROM identifies which table
Selecting All Columns
SELECT * FROM departments;

SELECT department_id, department_name, manager_id, location_ id FROM departments;

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATION_ID	DEPARTMENT_ID
1700	10
1800	20
1500	50

8 rows selected.

Selecting Specific Columns

SELECT department_id, location_id FROM departments;

Arithmetic Expressions

Create expressions with number and date data by using arithmetic operators.

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide

Using Arithmetic Operators

SELECT last_name, salary, salary + 300 FROM employees;

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

LAST_NAME	SALARY	12*(SALARY+100)
King	24000	289200
Kochhar	17000	205200
De Haan	17000	205200
Hunold	9000	109200
Ernst	6000	73200
Lorentz	4200	51600

SELECT last_name, salary, 12*salary+100 FROM employees;

Gietz	8300	100800

20 rows selected.

Using Parentheses

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	

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

Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2
Higgins	AC_MGR	12000	
Gietz	AC_ACCOUNT	8300	

20 rows selected.

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;

LAST_NAME	12*SALARY*COMMISSION_PCT
King	
Kochhar	
Zlotkey	25200
Abel	39600
Taylor	20640
Higgins	
Gietz	

20 rows selected.

Null Values in Arithmetic Expressions

Arithmetic expressions containing a null value evaluate to null.

SELECT last_name, 12*salary*commission_pct FROM employees;

Defining a Column Alias

A column alias:
•Renames a column heading
• Is useful with calculations
• Immediately follows the column name: there can also be the optional AS keyword between the column name and alias
• Requires double quotation marks if it contains spaces or special characters or is case sensitive

NAME	COMM
King	
Kochhar	
Higgins	
Gietz	

20 rows selected.

Name	Annual Salary
King	288000
Kochhar	204000
Higgins	144000
Gietz	99600

20 rows selected.

Using Column Aliases

```
SELECT last_name AS name, commission_pct comm
FROM   employees;
SELECT last_name  "Name",salary*12 "Annual Salary"
FROM   employees;
```

A concatenation operator:
• Concatenates columns or character strings to other columns
• Is represented by two vertical bars (||)
• Creates a resultant column that is a character expression

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
HunoldIT_PROG
GietzAC_ACCOUNT

20 rows selected.

```
SELECT last_name||job_id AS "Employees" FROM  employees;
```

Using the Concatenation Operator

Literal Character Strings

• A literal value is a character, a number, or a date included in the SELECT list.
• Date and character literal values must be enclosed within single quotation marks.
• Each character string is output once for each row returned.

Employee Details
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Gietz is a AC_ACCOUNT

20 rows selected.

MONTHLY
King: 1 Month salary = 24000
Kochhar: 1 Month salary = 17000
De Haan: 1 Month salary = 17000
Hunold: 1 Month salary = 9000
Ernst: 1 Month salary = 6000
Lorentz: 1 Month salary = 4200
Mourgos: 1 Month salary = 5800
Rajs: 1 Month salary = 3500

20 rows selected.

Using Literal Character Strings

```
SELECT last_name ||' is a '||job_id AS "Employee Details"
FROM   employees;
```

DEPARTMENT_ID
90
90
90
60
60
60
50
50

20 rows selected.

Duplicate Rows

The default display of queries is all rows, including duplicate rows.

SELECT department_id FROM employees;

DEPARTMENT_ID
10
20
50
60
80
90
110

8 rows selected.

DEPARTMENT_ID	JOB_ID
10	AD_ASST
20	MK_MAN
20	MK_REP
50	ST_CLERK
50	ST_MAN
60	IT_PROG
80	SA_MAN
80	SA_REP

13 rows selected.

Eliminating Duplicate Rows

Eliminate duplicate rows by using the DISTINCT keyword in the SELECT clause.
SELECT DISTINCT department_id FROM employees;

SQL and *i*SQL*Plus Interaction

SQL statements iSQL*Plus Oracle Internet server Browser
Query results iSQL*Plus commands Formatted report Client
SQL and iSQL*Plus SQL is a command language for communication with the Oracle server f rom any tool or application. Oracle SQL contains many extensions.
iSQL*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle server for execution and contains its own command language.
Features of SQL
• Can be used by a range of users, including those with little or no programming experience
• Is a nonprocedural language
• Reduces the amount of time required for creating and maintaining systems
• Is an English-like language
Features of iSQL*Plus
• Accessed from a browser
• Accepts ad hoc entry of statements
• Provides online editing for modifying SQL statements
• Controls environmental settings
• Formats query results into a basic report
• Accesses local and remote databases

SQL Statements versus *i*SQL*Plus Commands

SQL iSQL*Plus
• A language
• An environment
• ANSI standard
• Oracle proprietary
• Keyword cannot be
• Keywords can be abbreviated abbreviated
• Statements manipulate
• Commands do not allow data and table definitions manipulation of values in in the database the database
• Runs on a browser
• Centrally loaded, does not have to be implemented on each machine SQL iSQL*Plus statements commands
SQL and iSQL*Plus (continued)

Paper-Based Questions

For questions 2–4, circle either True or False.

Practice 1

1. Initiate an iSQL*Plus session using the user ID and password provided by the instructor.

2.iSQL*Plus commands access the database.

True/False

3. The following SELECT statement executes successfully:

SELECT last_name, job_id, salary AS Sal FROM employees;

True/False

4. The following SELECT statement executes successfully:

SELECT * FROM job_grades;

True/False

5. There are four coding errors in this statement. Can you identif y them?

SELECT employee_id, last_name sal x 12 ANNUAL SALARY
FROM employees;

6. Show the structure of the DEPARTMENTS table. Select all data from the table.

7. Show the structure of the
EMPLOYEES table. Create a query to display the last name, job code, hire
date, and employee number for each employee, with employee number appearing first. Save your SQL statement to a file named lab1_7.sql.

8. Run your query in the file lab1_7.sql.

9. Create a query to display unique job codes from the EMPLOYEES table.

If you have time, complete the following exercises:

10. Copy the statement from lab1_7.sql into the iSQL*Plus Edit window. Name the column headings Emp #,Employee,Job, and Hire Date, respectively. Run your query again.

Employee and Title
King, AD_PRES
Kochhar, AD_VP
De Haan, AD_VP
Hunold, IT_PROG
Ernst, IT_PROG
Lorentz, IT_PROG
Mourgos, ST_MAN
Rajs, ST_CLERK
Davies, ST_CLERK
Gietz, AC_ACCOUNT

20 rows selected.

THE_OUTPUT
100,Steven,King,SKING,515.123.4567,AD_PRES,,17-JUN-87,24000,,90
101,Neena,Kochhar,NKOCHHAR,515.123.4568,AD_VP,100,21-SEP-89,17000,,90
102,Lex,De Haan,LDEHAAN,515.123.4569,AD_VP,100,13-JAN-93,17000,,90
103,Alexander,Hunold,AHUNOLD,590.423.4567,IT_PROG,102,03-JAN-90,9000,,60
104,Bruce,Ernst,BERNST,590.423.4568,IT_PROG,103,21-MAY-91,6000,,60
107,Diana,Lorentz,DLORENTZ,590.423.5567,IT_PROG,103,07-FEB-99,4200,,60
124,Kevin,Mourgos,KMOURGOS,650.123.5234,ST_MAN,100,16-NOV-99,5800,,50
141,Trenna,Rajs,TRAJS,650.121.8009,ST_CLERK,124,17-OCT-95,3500,,50
206,William,Gietz,WGIETZ,515.123.8181,AC_ACCOUNT,205,07-JUN-94,8300,,110

20 rows selected.

11. Display the last name concatenated with the job ID, separated by a comma and space, and name the column Employee and Title.

If you want an extra challenge, complete the following exercise:

12. Create a query to display all the data from the EMPLOYEES table. Separate each column by a comma. Name the column THE_OUTPUT.

Restricting and Sorting Data Objectives

After completing this lesson, you should be able to do the following:

• Limit the rows retrieved by a query

• Sort the rows retrieved by a query

Limiting Rows Using a Selection

EMPLOYEES "retrieve all employees in department 90"

Limiting Rows Using a Selection

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.

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Using theWHERE Clause

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

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 last_name, job_id, department_id FROM   employees
WHERE  last_name = 'Goyal';
```

Comparison Conditions

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

Example

... WHERE hire_date='01-JAN-95'
... WHERE salary>=6000
... WHERE last_name='Smith'

An alias cannot be used in the WHERE clause.

Note: The symbol != and ^= can also represent the *not equal to* condition.

LAST_NAME	SALARY
Matos	2600
Vargas	2500

Using Comparison Conditions

```
SELECT last_name, salary FROM   employees
WHERE  salary <= 3000;
```

Using the Comparison Conditions

Other Comparison Conditions

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

Using the 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

The BETWEEN Condition

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

8 rows selected.

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);
```

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 first_name FROM  employees WHERE first_name LIKE 'S%';
```

LAST_NAME
Kochhar
Lorentz
Mourgos

EMPLOYEE_ID	LAST_NAME	JOB_ID
149	Zlotkey	SA_MAN
174	Abel	SA_REP
176	Taylor	SA_REP
178	Grant	SA_REP

Using the LIKE Condition

• You can combine pattern-matching characters.

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

• You can use the ESCAPE identifier to search for the

LAST_NAME	JOB_ID	COMMISSION_PCT
King	AD_PRES	
Kochhar	AD_VP	
De Haan	AD_VP	

Actual % and _ symbols.

LAST_NAME	MANAGER_ID
King	

Gietz	AC_ACCOUNT	

16 rows selected.

Using theNULLConditions

Test for nulls with the IS NULL operator.

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

Logical Conditions

Meaning	Operator
AND	Returns TRUE if *both* component conditions are true
OR	Returns TRUE If *either* component

	condition is true
NOT	**Returns TRUE if the following condition is false**

Using the AND Operator

AND requires both conditions to be true.

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

Using the OR Operator

OR

requires either condition to be true.

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

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

10 rows selected.

Using the NOT Operator

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

Rules of Precedence

Order Evaluated Operator

1 Arithmetic operators

2 Concatenation operator

3 Comparison conditions

4 IS [NOT] NULL , LIKE , [NOT] IN

5 [NOT] BETWEEN

6 NOT logical condition

7 AND logical condition
8 OR logical condition
Override rules of precedence by using parentheses.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = 'SA_REP'
OR     job_id = 'AD_PRES'
AND    salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

Use parentheses to force priority.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  (job_id = 'SA_REP'
OR     job_id = 'AD_PRES')
AND    salary > 15000;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91
De Haan	AD_VP	90	13-JAN-93

20 rows selected.

ORDER BY Clause

• Sort rows with the ORDER BY clause
– ASC : ascending order (the default order)
– 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;
```

Syntax

SELECT *expr* FROM *table* [WHERE *condition(s)*] [ORDER BY {*column*, *Expr* } [ASC|DESC]];

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Zlotkey	SA_MAN	80	29-JAN-00
Mourgos	ST_MAN	50	16-NOV-99
Grant	SA_REP		24-MAY-99
Lorentz	IT_PROG	60	07-FEB-99
Vargas	ST_CLERK	50	09-JUL-98
Taylor	SA_REP	80	24-MAR-98
Matos	ST_CLERK	50	15-MAR-98
Fay	MK_REP	20	17-AUG-97
Davies	ST_CLERK	50	29-JAN-97
Abel	SA_REP	80	11-MAY-96
King	AD_PRES	90	17-JUN-87

20 rows selected.

Sorting in Descending Order

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

EMPLOYEE_ID	LAST_NAME	ANNSAL
144	Vargas	30000
143	Matos	31200
142	Davies	37200
141	Rajs	42000
107	Lorentz	50400
200	Whalen	52800
124	Mourgos	69600
104	Ernst	72000
202	Fay	72000
178	Grant	84000
206	Gietz	99600

```
ORDER BY hire_date DESC;
```

100	King	288000

20 rows selected.

Sorting by Column Alias

```
SELECT employee_id, last_name, salary*12 annsal
FROM   employees
ORDER BY annsal;
```

Sorting by Column Aliases

You can use a column alias in the ORDER BY clause. The slide example

sorts the data by annual salary.

Higgins	110	12000
Gietz	110	8300
Grant		7000

20 rows selected.

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Hartstein	20	13000
Fay	20	6000
Mourgos	50	5800
Rajs	50	3500

Sorting by Multiple Columns

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

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

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

Practice 1

1. Create a query to display the last name and salary of employees earning more than $12,000.

Place your SQL statement in a text file named lab2_1.sql . Run your query.

2. Create a query to display the employee last name and department number for employee number 176.

3. Modify lab2_1.sql to display the last name and salary for all employees whose salary is not in the range of $5,000 and $12,000. Place your SQL statement in a text file named lab2_3.sql

4. Display the employee last name, job ID, and start date of employees hired between February 20, 1998, and May 1, 1998. Order the query in ascending order by start date.

5. Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.

6. Modify lab2_3.sql to list the last name and salary of employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary , respectively. Resave lab2_3.sql as lab2_6.sql . Run the statement in lab2_6.sql .

7. Display the last name and hire date of every employee who was hired in 1994.

8. Display the last name and job title of all employees who do not have a manager.

9. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

If you have time, complete the following exercises:

10. Display the last names of all employees where the third letter of the name is an *a.*

11. Display the last name of all employees who have an *a* and an *e* in their last name.

If you want an extra challenge, complete the following exercises:

12. Display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to $2,500, $3,500, or $7,000.

13. Modify lab2_6.sql to display the last name, salary, and commission for all employees whose commission amount is 20%. Resave lab2_6.sql as lab2_13.sql . Rerun the statement in lab2_13.sql.

2-Single-Row Functions

Objectives

After completing this lesson, you should be able to do the following:

• Describe various types of functions available in SQL

• Use character, number, and date functions in SELECT statements

• Describe the use of conversion functions

SQL Functions

Input Output Function

Function performs action arg 1

arg 2 Result value arg n

Note:

Most of the functions described in this lesson are specific to Oracle Corporation's version of SQL.

Two Types of SQL Functions

Functions

Single-row functions

Single-Row Functions

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 which can be a column or an expression
function_name[(arg1, arg2,...)]
Single-Row Functions

Single-Row Functions

Character
General
Number
Single-row functions
Conversion
Date

Character Functions

Character functions
Case-manipulation functions
Character-manipulation functions
LOWER
CONCAT
UPPER
SUBSTR
INITCAP
LENGTH
INSTR
LPAD | RPAD
TRIM
REPLACE

Note: The functions discussed in this lesson are only some of the available functions.

Case Manipulation Functions

These functions convert case for character strings.

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

Using Case Manipulation Functions

Display the employee number, name, and department number for employee Higgins:

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

no rows selected

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  LOWER(last_name) = 'higgins';
```

Character-Manipulation Functions

These functions manipulate character strings:

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*****

EMPLOYEE_ID	NAME	JOB_ID	LENGTH(LAST_NAME)	Contains 'a'?
174	EllenAbel	SA_REP	4	0
176	JonathonTaylor	SA_REP	6	2
178	KimberelyGrant	SA_REP	5	3
202	PatFay	MK_REP	3	2

TRIM('H' FROM 'HelloWorld') elloWorld

EMPLOYEE_ID	NAME	LENGTH(LAST_NAME)	Contains 'a'?
102	LexDe Haan	7	5
200	JenniferWhalen	6	3
201	MichaelHartstein	9	2

Using the Character-Manipulation Functions

```
SELECT employee_id, CONCAT(first_name, last_name) NAME, job_id,
LENGTH (last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
```

Number Functions

• ROUND : Rounds value to specified decimal
ROUND(45.926, 2) 45.93
• TRUNC : Truncates value to specified decimal
TRUNC(45.926, 2) 45.92
• MOD : Returns remainder of division
MOD(1600, 300) 100

ROUND(45.923,2)	ROUND(45.923,0)	ROUND(45.923,-1)
45.92	46	50

Using the ROUND Function

SELECT ROUND(45.923,2), ROUND(45.923,0),ROUND(45.923,-1)
FROM DUAL;
DUAL is a dummy table you can use to view results from functions and calculations.

TRUNC(45.923,2)	TRUNC(45.923)	TRUNC(45.923, 2)
45.92	45	0

Using the TRUNC Function

LAST_NAME	SALARY	MOD(SALARY,5000)
Abel	11000	1000
Taylor	8600	3600
Grant	7000	2000

SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-2) FROM DUAL;

Using the MOD Function

Calculate the remainder of a salary after it is divided by 5000 for all employees whose job title is sales representative.
SELECT last_name, salary, MOD(salary, 5000)
FROM employees

LAST_NAME	HIRE_DATE
Gietz	07-JUN-94
Grant	24-MAY-99

WHERE job_id = 'SA_REP';

Working with Dates

• Oracle database stores dates in an internal numeric format: century, year, month, day, hours, minutes, seconds.
• The default date display format is DD-MON-RR.
– Allows you to store 21st century dates in the 20th century by specifying only the last two digits of the year.
– Allowa you to store 20th century dates in the 21st century in the same way.
SELECT last_name, hire_date FROM employees

WHERE last_name like 'G%';

SYSDATE
08-MAR-01

Working with Dates

SYSDATE is a function that returns:
•Date
•Time

Example Display the current date using the DUAL table.
SELECT SYSDATE FROM DUAL;

Arithmetic with Dates

•Add or subtract a number to or from a date for a resultant date value.
•Subtract two dates to find the number of days between those dates.
•Add hours to a date by dividing the number of hours by 24.

Operation	Result	Description
date + number	Date	Adds a number of days to a date
date - number	Date	Subtracts a number of days from a date
date - date	Number of days	Subtracts one date from another
date + number/24	Date	Adds a number of hours to a date

LAST_NAME	WEEKS
King	716.227563
Kochhar	598.084706
De Haan	425.227563

Using Arithmetic Operators with Dates

SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM employees
WHERE department_id = 90;

Note:SYSDATE is a SQL function that returns the current date and time. Your results may differ from the example. If a more current date is subtracted from an older date, the difference is a negative number.

Date Functions

EMPLOYEE_ID	HIRE_DATE	TENURE	REVIEW	NEXT_DAY(	LAST_DAY(
107	07-FEB-99	25.0548529	07-AUG-99	12-FEB-99	28-FEB-99
124	16-NOV-99	15.7645303	16-MAY-00	19-NOV-99	30-NOV-99
143	15-MAR-98	35.7967884	15-SEP-98	20-MAR-98	31-MAR-98
144	09-JUL-98	31.9903368	09-JAN-99	10-JUL-98	31-JUL-98
149	29-JAN-00	13.3451755	29-JUL-00	04-FEB-00	31-JAN-00
176	24-MAR-98	35.5064658	24-SEP-98	27-MAR-98	31-MAR-98
178	24-MAY-99	21.5064658	24-NOV-99	28-MAY-99	31-MAY-99

7 rows selected.

Using Date Functions

•MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')
19.6774194
•ADD_MONTHS ('11-JAN-94',6) '11-JUL-94'
•NEXT_DAY ('01-SEP-95','FRIDAY') '08-SEP-95'
• LAST_DAY('01-FEB-95') '28-FEB-95'

EMPLOYEE_ID	HIRE_DATE	ROUND(HIR	TRUNC(HIR
142	29-JAN-97	01-FEB-97	01-JAN-97
202	17-AUG-97	01-SEP-97	01-AUG-97

Using Date Functions

Assume SYSDATE = '25-JUL-95':

- ROUND(SYSDATE,'MONTH') 01-AUG-95
- ROUND(SYSDATE ,'YEAR') 01-JAN-96
- TRUNC(SYSDATE ,'MONTH') 01-JUL-95
- TRUNC(SYSDATE ,'YEAR') 01-JAN-95

Example

Compare the hire dates for all employees who started in 1997. Display the employee number, hire date, and month started using the ROUND and TRUNC functions.

SELECT employee_id, hire_date, ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH') FROM employees WHERE hire_date LIKE '%97';

Practice 3, Part 1

This practice is designed to give you a variety of exercises using different functions available for character,number, and date data types.

Complete questions 1 through 5 of Practice 3, found at the end of this lesson.

Conversion Functions

Data-type conversion
Implicit data-type conversion

Explicit data-type conversion

Note: Although implicit data-type conversion is available, it is recommended that you do explicit data type conversion to ensure the reliability of your SQL statements.

Implicit Data-Type Conversion

For assignments, the Oracle server can automatically convert the following:

From	To
VARCHAR2 or CHAR	NUMBER

VARCHAR2 or CHAR	**DATE**
NUMBER	**VARCHAR2**
DATE	**VARCHAR2**
VARCHAR2 or CHAR	**DATE**

Note: CHAR to NUMBER conversions succeed only if the character string represents a valid number.

Explicit Data-Type Conversion

TO_NUMBER
TO_DATE
NUMBER
DATE
CHARACTER
TO_CHAR
TO_CHAR

Function	**Purpose**
TO_CHAR(*number* | *date* ,[*fmt*], VARCHAR2 *[nlsparams*])	Converts a number or date value to a character string with format model *fmt*.
TO_NUMBER(*char,[fmt],* *[nlsparams]*)	Converts a character string containing digits to a number in the format specified by the optional format model *Fmt* . The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for number conversion.
TO_DATE(c*har* ,[*fmt*],[*nlsparams*])	Converts a character string representing a date to a date value according to the *fmt* specified. If *fmt* is omitted, the format is DD-MON-YY.The nlsparams parameter has the same purpose in this function as in the TO_CHAR function for date conversion.

Note: The list of functions mentioned in this lesson includes only some of the

EMPLOYEE_ID	MONTH
205	06/94

available conversion functions.

Using the TO_CHAR Function with Dates

TO_CHAR(date, ' format_model ')
The format model:
•Must be enclosed in single quotation marks and is case sensitive
•Can include any valid date format element

•Has an fm element to remove padded blanks or suppress leading zeros
• Is separated from the date value by a comma

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

Elements of the Date Format Model

YYYY Full year in numbers
YEAR Year spelled out
MM Two-digit value for month
MONTH Full name of the month
MON Three-letter abbreviation of the
Month Three-letter abbreviation of the
DY day of the week
DAY Full name of the day of the week
DD Numeric day of the month

Elements of the Date Format Model

• Time elements format the time portion of the date.
HH24:MI:SS AM 15:45:32 PM
• Add character strings by enclosing them in double quotation marks.
DD "of" MONTH 12 of OCTOBER
• Number suffixes spell out numbers.
ddspth fourteenth

Using the TO_CHAR Function with Dates

SELECT last_name, TO_CHAR(hire_date, 'fmDD Month YYYY') HIREDATE FROM employees;

```
SELECT   last_name, TO_CHAR(hire_date, 'fmDdspth "of" Month YYYY
fmHH:MI:SS AM') HIREDATE FROM    employees;
```

Notice that the month follows the format model specified: in oth er words, the first letter is capitalized and the rest are lowercase.

Using the TO_CHAR Function with Numbers

TO_CHAR(number, ' format_model ')
These are some of the format elements you can use
with the TO_CHAR function to display a number value
as a character:9 Represents a number 0
Forces a zero to be displayed $ Places a floating dollar sign
L Uses the floating local currency symbol. Prints a decimal point, Prints a thousand indicator

Element	Description	Example	Result
9	Numeric position (number of 9s determine display width)	999999	1234
0	Display leading zeros	099999	001234
$	Floating dollar sign	$999999	$1234
L	Floating local currency symbol	L999999	FF1234
.	Decimal point in position specified	999999.99	1234.00
,	Comma in position specified	999,999	1,234
MI	Minus signs to right (negative values)	999999MI	1234-
PR	Parenthesize negative numbers	999999PR	<1234>
EEEE	Scientific notation (format must specify four Es)	99.999EEEE	1.234E+03
V	Multiply by 10 *n* times (*n* = number of 9s after V)	9999V99	123400
B	Display zero values as blank, not 0	B9999.99	1234.00

SALARY
$3,000.00

Using the TO_CHAR Function with Numbers

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

Using the TO_NUMBER And TO_DATE Functions

•Convert a character string to a number format using the TO_NUMBER function: TO_NUMBER(char [,'format_model'])
•Convert a character string to a date format using the TO_DATE function: TO_DATE(char [, 'format_model'])
•These functions have an fx modifier. This modifier specifies the exact matching for the character argument and date format model of a TO_DATE function.

Example

Display the names and hire dates of all the employees who joined on May 24, 1999. Because the fx modifier is used, an exact match is required and the

spaces afte r the word “May” are not recognized.
SELECT last_name, hire_date FROM employees
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY')

Example of RR Date Format

To find employees hired prior to 1990, use the RR format, which produces the same results whether the command is run in 1999 or now:

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM   employees
WHERE  hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

SELECT last_name, TO_CHAR(hire_date, 'DD -Mon-yyyy')
FROM employees
WHERE TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
no rows selected

Nesting Functions

- **Single-row functions can be nested to any level.**
- **Nested functions are evaluated from deepest level to the least deep level.**

F3(F2(F1(col,arg1),arg2),arg3)
Step 1 = Result 1
Step 2 = Result 2
Step 3 = Result 3
Nesting Functions

Single-row functions can be nested to any depth. Nested functions are evaluated from the innermost level to the outermost level. Some examples follow to show you the flexibility of these functions.

LAST_NAME	NVL(TO_CHAR(MANAGER_ID),'NOMANAGER')
King	No Manager

Nesting Functions

```
SELECT last_name,
NVL(TO_CHAR(manager_id), 'No Manager')
FROM   employees
WHERE  manager_id IS NULL;
```

Example

Display the date of the next Friday that is six months from the hire date. The

resulting date should appear as Friday, August 13th, 1999. Order the results by hire date.

```
SELECT   TO_CHAR(NEXT_DAY(ADD_MONTHS
(hire_date, 6), 'FRIDAY'),
'fmDay, Month DDth, YYYY')
"Next 6 Month Review"
FROM      employees
ORDER BY  hire_date;
```

General Functions

These functions work with any data type and pertain to using null value.

- **NVL (expr1, expr2)**
- **NVL2 (expr1, expr2, expr3)**
- **NULLIF (expr1, expr2)**
- **COALESCE (expr1, expr2, ..., exprn)**

Function	Description
NVL	Converts a null value to an actual value
NVL2	If expr1 is not null, NVL2 returns expr2.
NULLIF	If expr1 is null, NVL2 returns. The argument can have any data type. expr3 expr1
COALESCE	Compares two expressions and returns null if they are equal, or the first expression if they are not equal Returns the first non-null expression in the expression list

NVL Function

- **Converts a null to an actual value**
- **Data types that can be used are date, character, and number.**
- **Data types must match:**
  - **NVL(commission_pct,0)**
  - **NVL(hire_date,'01-JAN-97')**
  - **NVL(job_id,'No Job Yet')**

NVL Conversions for Various Data Types

Data Type	Conversion Example
NUMBER	NVL(*number_column* ,9)
DATE	NVL(*date_column*, '01-JAN-95')
CHAR or VARCHAR2	NVL(*character_column* , 'Unavailable')

Vargas	2500		
Zlotkey	10500	.2	151200
Abel	11000	.3	171600
Taylor	8600	.2	123840

LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
King	24000	0	288000
Kochhar	17000	0	204000
De Haan	17000	0	204000
Hunold	9000	0	108000
Ernst	6000	0	72000
Lorentz	4200	0	50400
Mourgos	5800	0	69600
Rajs	3500	0	42000
Davies	3100	0	37200
Matos	2600	0	31200
Vargas	2500	0	30000
Zlotkey	10500	.2	151200
Abel	11000	.3	171600

20 rows selected.

LAST_NAME	SALARY	COMMISSION_PCT	AN_SAL

20 rows selected.

Using the NVL Function

SELECT last_name, salary, NVL(commission_pct, 0),
(salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;

The NVL Function

To calculate the annual compensation of all employees, you need to multiply the monthly salary by 12 and then add the commission percentage to it.

SELECT last_name, salary, commission_pct, (salary*12) + (salary*12*commission_pct) AN_SAL FROM employees;

LAST_NAME	SALARY	COMMISSION_PCT	INCOME
Zlotkey	10500	.2	SAL+COMM
Abel	11000	.3	SAL+COMM
Taylor	8600	.2	SAL+COMM
Mourgos	5800		SAL
Rajs	3500		SAL
Davies	3100		SAL
Matos	2600		SAL
Vargas	2500		SAL

8 rows selected.

Using the NVL2 Function

SELECT last_name, salary, commission_pct,
NVL2(commission_pct,'SAL+COMM', 'SAL') income

FIRST_NAME	expr1	LAST_NAME	expr2	RESULT
William	7	Gietz	5	7
Shelley	7	Higgins	7	
Pat	3	Fay	3	
Michael	7	Hartstein	9	7
Jennifer	8	Whalen	6	8
Kimberely	9	Grant	5	9
Jonathon	8	Taylor	6	8
Ellen	5	Abel	4	5
Eleni	5	Zlotkey	7	5
Peter	5	Vargas	6	5
Randall	7	Matos	5	7
Curtis	6	Davies	6	
Trenna	6	Rajs	4	6
Kevin	5	Mourgos	7	5

FROM employees WHERE department_id IN (50, 80);

20 rows selected.

Using the NULLIF Function

SELECT first_name, LENGTH(first_name) "expr1",
last_name, LENGTH(last_name) "expr2",
NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM employees;

Note:

The NULLIF function is logically equivalent to the following CASE expression. The CASE expression is discussed in a subsequent page:
CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END

Practice 2, Part 1 (continued)

5. Write a query that displays the employee's last names with the first letter capitalized and all other letters lowercase and the length of the names, for all employees whose name starts with *J* , *A* , or *M* . Give each column an appropriate label. Sort the results by the employees' last names.

Practice 2, Part 2

6. For each employee, display the employee's last name, and calculate the number of months between today and the date the employee was hired. Label the column MONTHS_WORKED . Order your results by the number of months employed. Round the number of months up to the closest whole number.

Note:

Your results will differ.

Gietz earns $8,300.00 monthly but wants $24,900.00.

20 rows selected.

Gietz	$$$$$$$$$$8300

20 rows selected.

Practice 3, Part 2 (continued)

7. Write a query that produces the following for each employee:
< employee last name> earns <salary> monthly but wants <3 times salary >. Label the column Dream Salaries . If you have time, complete the following exercises:
8. Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with $. Label the column SALARY .
9. Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear similar to "Monday, the Thirty-First of July, 2000."

EMPLOYEE_AND_THEIR_SALARIES
King **********
Kochhar **********
De Haan **********
Hartstei **********
Higgins **********
Abel **********
Zlotkey **********
Hunold *********
Taylor ********
Gietz ********
Grant *******

10. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY . Order the results by the day of the week starting with Monday.
If you want an extra challenge, complete the following exercises:
11. Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, put "No Commission." Label the column COMM.
12. Create a query that displays the employees' last names and indic ates the amounts of their annual salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES_AND_THEIR_SALARIES.
13. Using the DECODE function, write a query that displays the grade of all employee s based on the value of the column JOB_ID , as per the following

data:
Job Grade
AD_PRES
A *ST_MAN*
B *IT_PROG*
C *SA_REP*
D *ST_CLERK*
E None of the above 0
14. Rewrite the statement in the preceding question using the CASE syntax.

3-Displaying Data from Multiple Tables

Objectives

After completing this lesson, you should be able to do the following:
• Write SELECT statements to access data from more than one table using equality and nonequality joins
• View data that generally does not meet a join condition by using outer joins
• Join a table to itself by using a self join

Obtaining Data from Multiple Tables

EMPLOYEES DEPARTMENTS

Obtaining Data from Multiple Tables

Sometimes you need to use data from more than one table. In the example, the report displays data fromtwo separate tables.
• Employee IDs exist in the EMPLOYEES table.
• Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
• Location IDs exist in the DEPARTMENTS table.
To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables and access data from both of them.

Cartesian Products

•A Cartesian product is formed when:
–A join condition is omitted
–A join condition is invalid
–All rows in the first table are joined to all rows in the second table
•To avoid a Cartesian product, always include a valid join condition in a WHERE clause.
Cartesian Products

Generating a Cartesian Product

EMPLOYEES
(20 rows)
DEPARTMENTS
(8 rows)
Cartesian product:
20x8=160 rows

SELECT last_name, department_name dept_name
FROM employees, departments;

Types of Joins

Oracle Proprietary SQL: 1999
Compliant Joins: Joins (8i and prior):
• Equijoin
•Cross joins
•Nonequijoin
•Natural joins
•Outer join
Using clause

•Self join
•Full or two sided outer joins
• Arbitrary join conditions for outer joins

Joining Tables Using Oracle Syntax

Use a join to query data from more than one table.

```
SELECT
table1.column, table2.column
FROM
table1, table2
WHERE
table1.column1 = table2.column2;
```

•Write the join condition in the WHERE clause.
• Prefix the column name with the table name when the same column name appears in more than one table.

What Is an Equijoin?

EMPLOYEES DEPARTMENTS

Foreign key Primary key

Equijoins

To determine an employee's department name, you compare the value in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an *equijoin* , that is, values in the DEPARTMENT_ID column on both tables must be equal. Frequently, this type of join involves primary and foreign key complements.

Note: Equijoins are also called *simple joins* or *inner joins*

.

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
205	Higgins	110	110	1700
206	Gietz	110	110	1700

19 rows selected.

Retrieving Records with Equijoins

```
SELECT employees.employee_id, employees.last_name,
employees.department_id, departments.department_id,
departments.location_id
FROM   employees, departments
WHERE  employees.department_id = departments.department_id;
```

Additional Search Conditions Using the AND Operator

EMPLOYEES DEPARTMENTS

```
SELECT last_name, employees.department_id,
department_name
FROM   employees, departments
```

WHERE employees.department_id = departments.department_id
AND   last_name = 'Matos';

Qualifying Ambiguous Column Names

• Use table prefixes to qualify column names that are in multiple tables.
• Improve performance by using table prefixes.
• Distinguish columns that have identical names but reside in different tables by using column aliases.

Using Table Aliases

•Simplify queries by using table aliases
•Improve performance by using table prefixes

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

Joining More than Two Tables

EMPLOYEES
DEPARTMENTS
LOCATIONS
To join n tables together, you need a minimum of n -1 join conditions. For example, to join three tables, a minimum of two joins is required.

```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

Nonequijoins

EMPLOYEES JOB_GRADES

Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB_GRADES table.

Retrieving Records with Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary BETWEEN j.lowest_sal AND j.highest _sal;
```

Outer Joins

DEPARTMENTS EMPLOYEES

There are no employees in department 190.

SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;

Outer Joins Syntax

• You use an outer join to also see rows that do not meet the join condition.
• The outer join operator is the plus sign (+).

```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column(+) = table2.column;
SELECT table1.column, table2.column
FROM table1, table2
 WHERE table1.column = table2.column(+);
```

Using Outer Joins

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id(+) = d.department_id;
```

Self Joins

EMPLOYEES (WORKER) EMPLOYEES (MANAGER)
MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

Joining a Table to Itself

```
SELECT worker.last_name || ' works for '
|| manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id;
```

Use a join to query data from more than one table.

```
SELECT table1.column, table2.column
FROM table1
[CROSS JOIN table2 ] | [NATURAL JOIN table2 ] | [JOIN table2
USING ( column_name )] | [JOIN table2 ON( table1.column_name
= table2.column_name )] | [LEFT|RIGHT|FULL OUTER JOIN
table2 ON ( table1.column_name = table2.column_name )];
```

Creating Cross Joins

•The CROSS JOIN clause produces the cross-

product of two tables.
• This is the same as a Cartesian product between the two tables.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments;
```

Creating Cross Joins

The example in the slide gives the same results as the following:
SELECT last_name, department_name FROM employees, departments;

Creating Natural Joins

• The NATURAL JOIN clause is based on all columns in the two tables that have the same name.
• It selects rows from the two tables that have equal values in all matched columns.
• If the columns having the same names have different data types, then an error is returned.

Note: The join can happen only on columns having the same names and data types in both the tables. If the columns have the same name, but different data types, then the NATURAL JOIN syntax causes an error.

Retrieving Records with Natural Joins

```
SELECT department_id, department_name,
location_id, city
FROM   departments
NATURAL JOIN locations;
```

```
SELECT department_id, department_name,
departments.location_id, city
FROM   departments, locations
WHERE  departments.location_id = locations.location_id;
```

```
SELECT  department_id, department_name,
location_id, city
FROM    departments
NATURAL JOIN locations
WHERE   department_id IN (20, 50);
```

Creating Joins with the USING Clause

• If several columns have the same names but the data types do not match, the NATURAL JOIN

clause can be modified with the USING clause to specify the columns that should be used for an equijoin.
Note: Use the USING clause to match only one column when more than one column matches.
• Do not use a table name or alias in the referenced columns.
• The NATURAL JOIN and USING clauses are mutually exclusive.

For example, this statement is valid:
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE location_id = 1400;
This statement is invalid because the
LOCATION_ID
is qualified in the where clause:
SELECT l.city, d.department_name
FROM locations l JOIN departments d USING (location_id)
WHERE d.location_id = 1400;
ORA-25154: column part of USING clause cannot have qualifier
The same restriction applies to NATURAL
joins also. Therefore columns that have the same name in both tables have to be used without any qualifiers.

Retrieving Records with the USING Clause

SELECT e.employee_id, e.last_name, d.location_id
FROM employees e JOIN departments d
USING (department_id);

SELECT employee_id, last_name,
employees.department_id, location_id
FROM employees, departments
WHERE employees.department_id = departments.department_id;

Creating Joins with the ON Clause

•The join condition for the natural join is basically an equijoin of all columns with the same name.
•To specify arbitrary conditions or specify columns

to join, the ON clause is used.
•Separates the join condition from other search conditions.
•The ON clause makes code easy to understand.

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

INNER versus OUTER Joins

• In SQL: 1999, the join of two tables returning only matched rows is an inner join.
• A join between two tables that returns the results of the inner join as well as unmatched rows left (or right) tables is a left (or right) outer join.
• A join between two tables that returns the results of an inner join as well as the results of a left and right join is a full outer join.
Joins: Comparing SQL: 1999 to Oracle Syntax

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
LEFT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

Example of
LEFT OUTER JOIN

This query retrieves all rows in the EMPLOYEES table, which is the left table even if there is no match in the DEPARTMENTS table.

This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id (+) = e.department_id;
```

RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

Example of
RIGHT OUTER JOIN

This query retrieves all rows in the DEPARTMENTS table, which is the right table even if there is no match in the EMPLOYEES table.
This query was completed in earlier releases as follows:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE d.department_id = e.department_id  (+);
```

FULL OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON (e.department_id = d.department_id);
```

Additional Conditions

```
SELECT e.employee_id, e.last_name, e.department_id,
d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149;
```

Higgins	110	Accounting
Gietz	110	Accounting

19 rows selected.

Practice 3, Part 1

1. Write a query to display the last name, department number, and department name for all employees.
2. Create a unique listing of all jobs that are in department 30. Include the location of department 90 in the output.
3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

4. Display the employee last name and department name for all employees who have an *a* (lowercase) in their last names. Place your SQL statement in a text file named lab4_4.sql.

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.
6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee ,
Emp# , Manager , and Mgr# , respectively.
Place your SQL statement in a text file named lab4_6.sql.

7. Modify lab4_6.sql to display all employees including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab4_7.sql . Run the query in lab4_7.sql.
If you have time, complete the following exercises:
8. Create a query that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label.

9. Show the structure of the JOB_GRADES table. Create a query that displays the name, job, department name, salary, and grade for all employees.
If you want an extra challenge, complete the following exercises:
10. Create a query to display the name and hire date of any employee hired after employee

Davies.
11. Display the names and hire dates for all employees who were hire d before their managers, along with their manager's names and hire dates. Label the columns Employee , Emp Hired , Manager , and Mgr Hired , respectively.

4-Aggregating Data Using Group Functions

Objectives

After completing this lesson, you should be able to do the following:

•Identify the available group functions
•Describe the use of group functions
•Group data using the GROUP BY clause
•Include or exclude grouped rows by using the HAVING clause

What Are Group Functions?

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

EMPLOYEES

The maximum salary in the EMPLOYEES table.

Types of Group Functions

• AVG
•COUNT
•MAX
•MIN
•SUM

Group Functions Syntax

```
SELECT [ column ,] group_function(column), ...
FROM table
```

[WHERE condition]
[GROUP BY column]
[ORDER BY column];

Using the AVG and SUM Functions

You can use AVG and SUM for numeric data.

```
SELECT AVG(salary), MAX(salary),
MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

Using the MIN and MAX Functions

You can use MIN and MAX for any data type.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM employees;
```

```
SELECT MIN(last_name), MAX(last_name)
FROM   employees;
```

Note: AVG , SUM , VARIANCE , and STDDEV functions can be used only with numeric data types.

COUNT(*)
5

Using the COUNT Function COUNT(*)

returns the number of rows in a table.

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

Using the COUNT Function

• 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 employees
WHERE department_id = 80;
```

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

Using the DISTINCT Keyword

• COUNT(DISTINCT expr) returns the number of distinct nonnull values of the expr .

• Display the number of distinct department values in the EMPLOYEES table.

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

Group Functions and Null Values

Group functions ignore null values in the column.

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

Using the NVL Function with Group Functions

The NVL function forces group functions to include null values.

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

Creating Groups of Data

EMPLOYEES
4400
9500
The average salary 3500 in EMPLOYEES
Table 6400 for each department.

Creating Groups of Data: GROUP BY Clause Syntax

```
SELECT column , group_function(column)
FROM table
[WHERE condition ]
[GROUP BY group_by_expression ]
[ORDER BY column ];
```

Divide rows in a table into smaller groups by using the

Using the GROUP BY Clause

All columns 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;
```

Using the GROUP BY Clause

Using the GROUP BY Clause

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

```
SELECT   AVG(salary)
```

FROM employees
GROUP BY department_id;

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

Grouping by More Than One Column

EMPLOYEES
Add up the salaries in the
EMPLOYEES table
for each job,
grouped by
department.

Using theGROUP BY Clause on Multiple Columns

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

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;
SELECT department_id, COUNT(last_name)
*****
ERROR at line 1:
ORA-00937: not a single-group group function

SELECT department_id, count(last_name)
FROM employees
GROUP BY department_id;

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

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 functions in the WHERE clause.

```
SELECT department_id, AVG(salary)
FROM   employees
WHERE  AVG(salary) > 8000
GROUP BY department_id;
WHERE  AVG(salary) > 8000
*
ERROR at line 3:
ORA-00934: group function is not allowed here
```

Illegal Queries Using Group Functions (continued)

```
SELECT department_id, AVG(salary)
FROM   employees
HAVING  AVG(salary) > 8000
GROUP BY department_id;
```

Excluding Group Results

EMPLOYEES
The maximum salary per department when it is greater than $10,000.

Excluding Group Results: The HAVING Clause

Use the HAVING clause to restrict groups:
1. Rows are grouped.
2. The group function is applied.
3. Groups matching the HAVING clause are displayed.

```
SELECT column , group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression ]
[HAVING group_condition]
[ORDER BY column ];
```

DEPARTMENT_ID	AVG(SALARY)
20	9500
80	10033.3333
90	19333.3333
110	10150

Using the HAVING Clause

SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary)>10000;
Using the HAVING Clause

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

JOB_ID	PAYROLL
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

HAVING max(salary)>10000;

Using the HAVING Clause

SELECT job_id, SUM(salary) PAYROLL
FROM employees
WHERE job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING SUM(salary) > 13000
ORDER BY SUM(salary);

MAX(AVG(SALARY))
19333.3333

.

Nesting Group Functions

Display the maximum average salary.
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;

Paper-Based Questions

For questions 1 through 3, circle either True or False.

Note:

Column aliases are used for the queries.

Practice 4

Determine the validity of the following three statements. Circle either True or False.

1. Group functions work across many rows to produce one result per group.

True/False

2. Group functions include nulls in calculations.

True/False

3. The WHERE clause restricts rows prior to inclusion in a group calculation

True/False

4. Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum , Minimum , Sum , and Average , respectively. Round your results to the nearest whole number. Place your SQL statement in a text file named lab5_6.sql.

5. Modify the query in lab5_4.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab5_4.sql To lab5_5.sql. Run the statement in lab5_5.sql.

6. Write a query to display the number of people with the same job.

7. Determine the number of managers without listing them. Label the column Number of Managers .

Hint: Use the MANAGER_ID column to determine the number of managers.

8. Write a query that displays the difference between the highest a nd lowest salaries. Label the column DIFFERENCE.

If you have time, complete the following exercises:

9. Display the manager number and the salary of the lowest paid employee for that

manager.
Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is less than $6,000. Sort the output in descending order of salary.

10. Write a query to display each department's name, location, number of employees, and the average salary for all employees in that department. Label the columns Name , Location , Number of People , and Salary , respectively. Round the average salary to two decimal places.
If you want an extra challenge, complete the following exercises:
11. Create a query that will display the total number of employees a nd, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.
12. Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

Subqueries Objectives

After completing this lesson, you should be able to do the following:

• Describe the types of problem that subqueries can solve
• Define subqueries
• List the types of subqueries
• Write single-row and multiple-row subqueries

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?
Main Query: Which employees have salaries greater?
than Abel's salary?
Subquery: ?
What is Abel's salary?

Subquery Syntax

```
SELECT select_list
FROM table
WHERE expr operator
 (SELECT  select_list
FROM table );
```

•The subquery (inner query) executes once before the main query.
•The result of the subquery is used by the main
output is used to complete the query condition for the main or outer query.

LAST_NAME
King
Kochhar
De Haan
Hartstein
Higgins

Using a Subquery

SELECT last_name
FROM employees
11000
WHERE salary >
(SELECT salary
FROM employees
WHERE last_name = 'Abel');

Types of Subqueries

• Single-row subquery Main query returns
ST_CLERK Subquery
• Multiple-row subquery Main query
ST_CLERK returns Subquery
SA_MAN

LAST_NAME	JOB_ID
Rajs	ST_CLERK
Davies	ST_CLERK
Matos	ST_CLERK
Vargas	ST_CLERK

Single-Row Subqueries

•Return only one row
•Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

<> **Not equal to**

Example

Display the employees whose job ID is the same as that of employee 141.

```
SELECT last_name, job_id FROM   employees WHERE  job_id =
(SELECT job_id FROM   employees WHERE  employee_id = 141);
```

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

Executing Single-Row Subqueries

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
ST_CLERK
(SELECT job_id
FROM   employees
WHERE  employee_id = 141)
AND    salary >
2600
(SELECT salary
FROM   employees
```

LAST_NAME	JOB_ID	SALARY
Vargas	ST_CLERK	2500

```
WHERE  employee_id = 143);
```

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM   employees
2500
WHERE  salary =
(SELECT MIN(salary)
```

DEPARTMENT_ID	MIN(SALARY)
10	4400
20	6000

```
FROM   employees);
```

	7000

7 rows selected.

The HAVING Clause with Subqueries

•The Oracle Server executes subqueries first.
•The Oracle Server returns results into the HAVING clause of the main query.

```
SELECT   department_id, MIN(salary)
FROM     employees
GROUP BY department_id
2500
HAVING   MIN(salary) >
(SELECT MIN(salary)
FROM employees
WHERE department_id = 50);
```

Example

Find the job with the lowest average salary.

```
SELECT   job_id, AVG(salary)
FROM     employees
GROUP BY job_id
HAVING   AVG(salary) = (SELECT   MIN(AVG(salary))
FROM     employees
GROUP BY job_id);
```

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM   employees
WHERE  salary =
(SELECT MIN(salary)
FROM   employees
GROUP BY department_id);
ERROR at line 4:
ORA-01427: single-row subquery returns more than
one row
```

Will This Statement Return Rows?

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
(SELECT job_id
FROM   employees
WHERE  last_name = 'Haas');
```

no rows selected

Multiple-Row Subqueries

•Return more than one row
•Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (SELECT MIN(salary)
FROM employees
GROUP BY department_id);

Example

Find the employees who earn the same salary as the minimum salar y for each department. The inner query is executed first, producing a query result. The main query block is then processed and uses the values returned by the inner query to complete its sear ch condition. In fact, the main query would look like the following to the Oracle Server:

SELECT last_name, salary, department_id
FROM employees
WHERE salary IN (2500, 4200, 4400, 6000, 7000, 8300, 8600, 1 7000);

Using the ANY Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
9000, 6000, 4200
WHERE  salary < ANY
(SELECT salary
FROM   employees
WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
9 00 0 ,  6 0 00 ,   42 0 0
(SELECT salary
FROM   employees
WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

Null Values in a Subquery

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
(SELECT mgr.manager_id
FROM   employees mgr);
no rows selected
```

Practice 5

1. Write a query to display the last name and hire date of any employee in the same department as Zlotkey. Exclude Zlotkey.
2. Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in ascending order of salary.
3. Write a query that displays the employee numbers and last names of all employees who work in a department with any employee whose last name contains a *u*. Place your SQL statement in a text file named
lab6_3.sql. Run your query.
4. Display the last name, department number, and job ID of all employees whose department location ID is 1700.
5. Display the last name and salary of every employee who reports to King.
6. Display the department number, last name, and job ID for every employee in the Executive department.

If you have time, complete the following exercises:

7. Modify the query in lab6_3.sql to display the employee numbers, last names, and salaries of all employees who earn more than the average salary and who work in a department with any employee with a *u* in their name. Resave lab6_3.sql to lab6_7.sql . Run the statement in lab6_7.sql.

6-Manipulating Data Objectives

After completing this lesson, you should be able to do the following:

- **Describe each DML statement**
- **Insert rows into a table**
- **Update rows in a table**
- **Delete rows from a table**
- **Merge rows in a table**
- **Control transactions**

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 of work.

Adding a New Row to a Table

New row DEPARTMENTS
Insert a new row into the DEPARMENTS table.

The INSERT Statement Syntax

• Add new rows to a table by using the INSERT statement.

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

•Only one row is inserted at a time with this syntax.

Adding a New Row to a Table (continued)

Note: This statement with the VALUES clause adds only one row at a time to a table.

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

Inserting New Rows

• Insert a new row containing values for each column.

•List values in the default order of the columns in the table.

•Optionally, list the columns in the INSERT clause.

```
INSERT INTO departments(department_id, department_name,
manager_id, location_id)
VALUES     (70, 'Public Relations', 100, 1700);
1 row created.
```

•Enclose character and date values within single quotation marks.

Inserting Rows with Null Values

• Implicit method: Omit the column from the column list.

```
INSERT INTO departments (department_id,
department_name    )
VALUES (30, 'Purchasing');
1 row created.
```

•Explicit method: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO departments
VALUES (100, 'Finance', NULL, NULL);
```

1 row created.

EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
113	Popp	AC_ACCOUNT	12-MAR-01	

Inserting Special Values

The SYSDATE function records the current date and time.

```
INSERT INTO employees (employee_id,
first_name, last_name,
email, phone_number,
hire_date, job_id, salary,
commission_pct, manager_id,
department_id)
VALUES (113, 'Louis', 'Popp', 'LPOPP', '515.124.4567',
SYSDATE, 'AC_ACCOUNT', 6900, NULL, 205, 100);
```

1 row created.

Confirming Additions to the Table

```
SELECT employee_id, last_name, job_id, hire_date, commission _pct
FROM   employees
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_P
114	Den	Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	AC_ACCOUNT	11000	

```
WHERE  employee_id = 113;
```

Inserting Specific Date Values

- **Add a new employee.**

```
INSERT INTO employees
VALUES     (114,
'Den', 'Raphealy',
'DRAPHEAL', '515.127.4561',
TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
'AC_ACCOUNT', 11000, NULL, 100, 30);
```

1 row created.

- **Verify your addition.**

```
INSERT INTO departments
(department_id, department_name, location_id)
VALUES     (&department_id, '&department_name',&location);
```

40

Human Resources

2500

1 row created.

Copying Rows from Another Table

- **Write your INSERT statement with a subquery.**

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM   employees
WHERE  job_id LIKE '%REP%';
```

4 rows created.

•Do not use the VALUES clause.

• Match the number of columns in the INSERT clause to those in the subquery.

```
INSERT INTO copy_emp SELECT *
FROM employees;
```

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.

Note: In general, use the primary key to identify a single row. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

LAST_NAME	DEPARTMENT_ID
King	110
Kochhar	110
De Haan	110
Hunold	110
Ernst	110
Lorentz	110
Mourgos	110

Gietz	110

22 rows selected.

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 = 113;
```

1 row updated.

• All rows in the table are modified if you omit the WHERE clause.

```
UPDATE  copy_emp
SET     department_id = 110;
```

22 rows updated.

Updating Two Columns with a Subquery

Update employee 114's job and department 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;
```

1 row updated.

Updating Rows Based on Another Table

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

```
UPDATE  copy_emp
SET     department_id  =  (SELECT department_id
FROM employees
WHERE employee_id = 100)
WHERE   job_id        =  (SELECT job_id
FROM employees
WHERE employee_id = 200);
```

1 row updated.

Updating Rows:

Integrity Constraint Error

```
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110;
UPDATE employees
*
ERROR at line 1:
ORA-02291: integrity constraint (HR.EMP_DEPT_FK)
violated - parent key not found
```

Integrity Constraint Error

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
100	Finance		
80	Sales	149	2500

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
70	Public Relations	100	1700
30	Purchasing		
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

Removing a Row from a Table

DEPARTMENTS
Delete a row from the
DEPARTMENTS
table.

The DELETE Statement

You can remove existing rows from a table by using
The DELETE statement.

```
DELETE [FROM] table
[WHERE condition ];
```

Note: If no rows are deleted, a message “
0 rows deleted .” is returned:

Deleting Rows from a Table

•Specific rows are deleted if you specify the WHERE clause.

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

1 row deleted.

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

```
DELETE FROM  copy_emp;
```

22 rows deleted.

Example

Remove rows identified in the WHERE clause.

```
DELETE FROM  employees
WHERE        employee_id = 114;
```

1 row deleted.

```
DELETE FROM  departments
WHERE        department_id IN (30, 40);
```

2 rows deleted.

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 '%Public%');
```

1 row deleted.

Deleting Rows:

Integrity Constraint Error

```
DELETE FROM departments
WHERE       department_id = 60;
DELETE FROM departments
*
```

ERROR at line 1:
ORA-02292: integrity constraint (HR.EMP_DEPT_FK) violated - child record found
Integrity Constraint Error

The following statement works because there are no employees in department 70:

```
DELETE FROM  departments WHERE       department_id = 70;
1 row deleted.
```

Using a Subquery in an

INSERT Statement

```
INSERT INTO
(SELECT employee_id, last_name,
email, hire_date, job_id, salary,
department_id
FROM   employees
WHERE  department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
TO_DATE('07-JUN-99', 'DD-MON-RR'),
'ST_CLERK', 5000, 50);
```

1 row created.

EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
124	Mourgos	KMOURGOS	16-NOV-99	ST_MAN	5800	50
141	Rajs	TRAJS	17-OCT-95	ST_CLERK	3500	50
142	Davies	CDAVIES	29-JAN-97	ST_CLERK	3100	50
143	Matos	RMATOS	15-MAR-98	ST_CLERK	2600	50
144	Vargas	PVARGAS	09-JUL-98	ST_CLERK	2500	50
99999	Taylor	DTAYLOR	07-JUN-99	ST_CLERK	5000	50

6 rows selected.

Using a Subquery in an INSERT Statement

•Verify the results

```
SELECT employee_id, last_name, email, hire_date,
job_id, salary, department_id
FROM   employees
WHERE  department_id = 50;
```

Using the WITH CHECK OPTION Keyword on DML Statements

•A subquery is used to identify the table and columns of the DML statement.

•The WITH CHECK OPTION keyword prohibits you from changing rows that are not in the subquery.

```
INSERT INTO  (SELECT employee_id, last_name, email,
hire_date, job_id, salary
FROM   employees
```

WHERE department_id = 50 WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',
TO_DATE('07-JUN-99', 'DD-MON-RR'),
'ST_CLERK', 5000);
INSERT INTO
*****
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation

The MERGE Statement

•Provides the ability to conditionally update or insert data into a database table
•Performs an
UPDATE
if the row exists and an

INSERT

if it is a new row:

–Avoids separate updates
–Increases performance and ease of use
–Is useful in data warehousing applications

MERGE

Statement Syntax

You can conditionally insert or update rows in a table by using the MERGE statement.

```
MERGE INTO table_name
AS table_alias
USING ( table|view|sub_query) AS alias
ON (join condition)
WHEN MATCHED THEN
UPDATE SET
col1 = col_val1,
col2 = col2_val
WHEN NOT MATCHED THEN
INSERT (column_list)
VALUES (column_values);
```

Merging Rows

Insert or update rows in the OPY_EMP table to match the EMPLOYEES table.

```
MERGE INTO copy_emp AS c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
```

```
UPDATE SET
c.first_name     = e.first_name,
c.last_name      = e.last_name,
...
c.department_id  = e.department_id
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

Merging Rows: Example

```
MERGE INTO copy_emp AS c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
c.first_name     = e.first_name,
c.last_name      = e.last_name,
c.email          = e.email,
c.phone_number   = e.phone_number,
c.hire_date      = e.hire_date,
c.job_id         = e.job_id,
c.salary         = e.salary,
c.commission_pct = e.commission_pct,
c.manager_id     = e.manager_id,
c.department_id  = e.department_id
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name, e.last_name,
e.email, e.phone_number, e.hire_date, e.job_id,
e.salary, e.commission_pct, e.manager_id,
e.department_id);
```

Merging Rows

```
SELECT *
FROM COPY_EMP;
no rows selected
MERGE INTO copy_emp c
USING employees e
ON (c.employee_id = e.employee_id)
WHEN MATCHED THEN
UPDATE SET
...
WHEN NOT MATCHED THEN
INSERT VALUES...;
SELECT *
FROM COPY_EMP;
```

20 rows selected.

Database Transactions

A database transaction consists of one of the following:
•DML statements which constitute one consistent change to the data
•One DDL statement
•One DCL statement

Database Transactions

•Begin when the first DML SQL statement is executed
•End with one of the following events:
–A COMMIT or ROLLBACK statement is issued
–A DDL or DCL statement executes (automatic commit)
–The user exits iSQL*Plus
–The system crashes

Advantages of COMMIT and ROLLBACK Statements With COMMIT and ROLLBACK

statements, you can:
•Ensure data consistency
•Preview data changes before making changes permanent
•Group logically related operations
Controlling Transactions
COMMIT Time
Transaction
DELETE
SAVEPOINT A
INSERT
UPDATE
SAVEPOINT B
INSERT
ROLLBACK
ROLLBACK
ROLLBACK
to SAVEPOINT A
to SAVEPOINT B
Note:SAVEPOINT is not ANSI standard SQL.

Rolling Back Changes to a Marker

• Create a marker in a current transaction by using

The SAVEPOINT statement.
•Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.
UPDATE... SAVEPOINT update_done;
Savepoint created.
INSERT...
ROLLBACK TO update_done;
Rollback complete.

Implicit Transaction Processing

• An automatic commit occurs under the following circumstances:
–DDL statement is issued
–DCL statement is issued
–Normal exit from iSQL*Plus, without explicitly Issuing COMMIT or ROLLBACK statements
•An automatic rollback occurs under an abnormal termination of iSQL*Plus or a system failure.
Implicit Transaction Processing Status Circumstances

State of the Data Before COMMIT Or ROLLBACK

•The previous state of the data can be recovered.
•The current user can review the results of the DML operations by using the SELECT statement.
•Other users cannot view the results of the DML statements by the current user.
• The affected rows are locked; other users cannot change the data within the affected rows.

State of the Data After COMMIT

•Data changes are made permanent in the database.
•The previous state of the data is permanently lost.
•All users can view the results.
•Locks on the affected rows are released; those rows are available for other users to manipulate.
•All savepoints are erased.

Committing Data

•Make the changes.

```
DELETE FROM employees
WHERE  employee_id = 99999;
1 row deleted.
INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
```

1 row inserted.
•Commit the changes.
COMMIT;
Commit complete.
Example
Remove departments 290 and 300 in the DEPARTMENTS table, and update a row in the COPY_EMP table. Make the data change permanent.
DELETE FROM departments
WHERE department_id IN (290, 300);
2 rows deleted.
UPDATE copy_emp
SET department_id = 80
WHERE employee_id = 206;
1 row updated.
COMMIT;
Commit Complete.

State of the Data After ROLLBACK

Discard all pending changes by using the
ROLLBACK
statement:
•Data changes are undone.
•Previous state of the data is restored.
•Locks on the affected rows are released.
DELETE FROM copy_emp;
22 rows deleted.
ROLLBACK;
Rollback complete.
Rolling Back Changes
DELETE FROM test;
25,000 rows deleted.
ROLLBACK;
Rollback complete.
DELETE FROM test
WHERE id = 100;
1 row deleted.
SELECT *
FROM test
WHERE id = 100;
No rows selected.
COMMIT;
Commit complete.

Statement-Level Rollback

•If a single DML statement fails during execution, only that statement is rolled back.
• The Oracle Server implements an implicit savepoint.
•All other changes are retained.
•The user should terminate transactions explicitly by executing a COMMIT or ROLLBACK statement.

Read Consistency

•Read consistency guarantees a consistent view of the data at all times.
•Changes made by one user do not conflict with changes made by another user.
•Read consistency ensures that on the same data:
–Readers do not wait for writers
–Writers do not wait for readers

Implementation of Read Consistency

User A Data
UPDATE employees blocks
SET salary = 7000
WHERE last_name = 'Goyal';
Undo segments changed and
SELECT *
Read unchanged
FROM userA.employees;
Data consistent before image change: "old" data
User B

Locking

In an Oracle database, locks:
•Prevent destructive interaction between concurrent transactions
•Require no user action
•Use the lowest level of restrictiveness
•Are held for the duration of the transaction
•Are of two types: explicit locking and implicit locking

What Are Locks?

Locks are mechanisms that prevent destructive interaction between tra nsactions accessing the same resource, either a user object (such as tables or rows) or a system object not visible to users (such as shared data structures and data dictionary rows).

How the Oracle Database Locks Data

Locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested. Implicit locking occurs for all SQL statements except SELECT.
The users can also lock data manually, which is called explicit locking.

Implicit Locking

•Two lock modes:
–Exclusive: Locks out other users
–Share: Allows other users to access the server
•High level of data concurrency:
–DML: Table share, row exclusive
–Queries: No locks required
–DDL: Protects object definitions
•Locks held until commit or rollback

DML Locking

When performing data manipulation language (DML) operations, the Oracle Server provides data concurrency through DML locking. DML locks occur at two levels:

• A share lock is automatically obtained at the table level during DML operations. With share lock mode, several transactions can acquire share locks on the same resource.

• An exclusive lock is acquired automatically for each row modified by a DML statement. Exclusive locks prevent the row from being changed by other transactions until the transaction is committed or rolled back. This lock ensures that no other user can modify the same row at the same time and overwrite changes not yet committed by another user.

Note: DDL locks occur when you modify a database object such as a table.

Practice 6

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab8_1.sql script to build the MY_EMPLOYEE table to be used for the lab.
2. Describe the structure of the MY_EMPLOYEE table to identify the column names.
3. Add the first row of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

ID LAST_NAME FIRST_NAME USERID SALARY

1 Patel Ralph rpatel 895
2 Dancs Betty bdancs 860
3 Biri Ben bbiri 1100
4 Newman Chad cnewman 750
5 Ropeburn Audrey aropebur 1550

4. Populate the MY_EMPLOYEE table with the second row of sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.
5. Confirm your addition to the table.

6. Write an INSERT statement in a text file named loademp.sql to load rows into the MY_EMPLOYEE table. Concatenate the first letter of the first name and the f irst seven characters of the last name to produce the user ID.
7. Populate the table with the next two rows of sample data by runn ing the INSERT statement in the script that you created.
8. Confirm your additions to the table.
9. Make the data additions permanent.

Update and delete data in the MY_EMPLOYEE table.

10. Change the last name of employee 3 to Drexler.
11. Change the salary to 1000 for all employees with a salary less than 900.
12. Verify your changes to the table.
13. Delete Betty Dancs from the MY_EMPLOYEE table.
14. Confirm your changes to the table.
15. Commit all pending changes. Control data transaction to the MY_EMPLOYEE table.
16. Populate the table with the last row of sample data by modifying the statements in the script that you created in step 6. Run the statements in the script.
17. Confirm your addition to the table.
18. Mark an intermediate point in the processing of the transaction.
19. Empty the entire table.
20. Confirm that the table is empty.
21. Discard the most recent DELETE operation without discarding the earlier INSERT operation.
22. Confirm that the new row is still intact.
23. Make the data addition permanent.

Creating and Managing Tables

Objectives

After completing this lesson, you should be able to do the following:

- **Describe the main database objects**
- **Create tables**
- **Describe the data types that can be used when specifying column definition**
- **Alter table definitions**
- **Drop, rename, and truncate tables**

The CREATE TABLE Statement

- **You must have:**
  - **CREATE TABLE privilege**
  - **A storage area**

CREATE TABLE[schema.]table(column datatype[DEFAULTexpr][, ...]);

- **You specify:**
  - **Table name**
  - **Column name, column data type, and column size**

Referencing Another User's Tables

- **Tables belonging to other users are not in the user's schema.**
- **You should use the owner's name as a prefix to those tables.**

```
SELECT *
FROM user_b.employees;
```

Creating Tables

•Create the table.

```
CREATE TABLE dept
(deptno  NUMBER(2),
dname  VARCHAR2(14),
loc  VARCHAR2(13));
```

Table created.

•Confirm creation of the table.

```
DESCRIBE dept
```

Tables in the Oracle Database

•User tables:

–Are a collection of tables created and maintained by the user

–Contain user information

•Data dictionary:

–Is a collection of tables created and maintained by the Oracle Server

–Contain database information

Querying the Data Dictionary

•See the names of tables owned by the user.

```
SELECT table_name FROM user_tables;
```

•View distinct object types owned by the user.

```
SELECT DISTINCT object_type
FROM user_objects;
```

•View tables, views, synonyms, and sequences owned by the user.

```
SELECT * FROM user_catalog;
```

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.

```
CREATE TABLE  dept80
```

AS
SELECT employee_id, last_name,
salary*12 ANNSAL,
hire_date
FROM employees
WHERE department_id = 80;

Table created.

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

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);
The ALTER TABLE

Adding a Column

•Use the ADD clause to add columns.

ALTER TABLE dept80
ADD (job_id VARCHAR2(9));
Table altered.

Modifying a Column

• You can change a column's data type, size, and default value.

ALTER TABLE dept80
MODIFY (last_name VARCHAR2(30));
Table altered.

• A change to the default value affects only subsequent insertions to the table.

Dropping a Column

Use the DROP COLUMN

clause to drop columns you no longer need from the table.

```
ALTER TABLE dept80
DROP  COLUMN   job_id;
Table altered.
```

The SET UNUSED Option

•You use the SET UNUSED option to mark one or more columns as unused.

•You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.

```
ALTER TABLE table SET   UNUSED ( column );
OR
ALTER     TABLE  table COLUMN column;
SET UNUSED ALTER TABLE table
DROP  UNUSED COLUMNS;
```

```
ALTER TABLE  dept80
SET UNUSED (last_name);
Table altered.
ALTER TABLE  dept80
DROP  UNUSED COLUMNS;
Table altered.
```

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 dept80;
Table dropped.
```

Changing the Name of an Object

•To change the name of a table, view, sequence, or synonym, execute the RENAME statement.

```
RENAME dept TO detail_dept;
Table renamed.
```

•You must be the owner of the object.

Truncating a Table

• The TRUNCATE TABLE statement:

– Removes all rows from a table

–Releases the storage space used by that table

```
TRUNCATE TABLE detail_dept;
Table truncated.
```

•You cannot roll back row removal when using

TRUNCATE.
•Alternatively, you can remove rows by using the DELETE statement.

Adding Comments to a Table

• You can add comments to a table or column by using The COMMENT statement.

```
COMMENT ON TABLE employees
IS 'Employee Information';
```

Comment created.

•Comments can be viewed through the data dictionary views:
–ALL_COL_COMMENTS
–USER_COL_COMMENTS
–ALL_TAB_COMMENTS
–USER_TAB_COMMENTS

Practice 7

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab9_1.sql , then execute the statement in the script to create the table.
Confirm that the table is created.

ID	NAME
Column Name	
Key Type	
Nulls/Unique	
FK Table	
FK Column	
NUMBER	VARCHAR2
Data type	
7	25
Length	

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.
3. Create the EMP table based on the following table instance chart. Place the syntax in a script called lab9_3.sql , and then execute the statement in the script to create the table. Confirm that the table is created.

ID	LAST_NAME	FIRST_NAME	DEPT_ID
Column Name			
Key Type			
Nulls/Unique			
FK Table			
FK Column			
NUMBER	VARCHAR2	VARCHAR2	NUMBER
Data type Length			
7	25	25	7

4. Modify the EMP table to allow for longer employee last names. Confirm your modification.
5. Confirm that both the DEPT and EMP tables are stored in the data

dictionary. (**Hint:**USER_TABLES)
6. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE_ID , FIRST_NAME , LAST_NAME , SALARY , and DEPARTMENT_ID columns. Name the columns in your new table ID , FIRST_NAME , LAST_NAME , SALARY , and DEPT_ID , respectively.
7. Drop the EMP table.
8. Rename the EMPLOYEES2 table as EMP.
9. Add a comment to the DEPT and EMP table definitions describing the ables. Confirm your additions in the data dictionary.
10. Drop the FIRST_NAME column from the EMP table. Confirm your modification by checking the description of the table.
11. In the EMP table, mark the DEPT_ID column in the EMP table as UNUSED . Confirm your modification by checking the description of the table.
12. Drop all the UNUSED columns from the EMP table. Confirm your modification by checking the description of the table.

8- Constraints

Objectives

After completing this lesson, you should be able to do the following:

• Describe constraints

•Create and maintain constraints

What Are Constraints?

•Constraints enforce rules at the table level.

•Constraints prevent the deletion of a table if there are dependencies.

•The following constraint types are valid:

–NOT NULL

–UNIQUE

–PRIMARY KEY

–FOREIGN KEY

–CHECK

Defining Constraints

```
CREATE TABLE [schema.]table(column datatype
[DEFAULTexpr][column_constraint],...[table_constraint
][,...]);
CREATE TABLE employees(
employee_id  NUMBER(6),
first_name   VARCHAR2(20),
```

...
job_id VARCHAR2(10) NOT NULL,
CONSTRAINT emp_emp_id_pk
PRIMARY KEY (EMPLOYEE_ID));

Defining Constraints

•Column constraint level:
Column [CONSTRAINT constraint_name]
constraint_type ,
•Table constraint level:
column,...[CONSTRAINT constraint_name]
constraint_type(column, ...),

The NOT NULL Constraint

Ensures that null values are not permitted for the
The NOT NULL Constraint
The NOT NULL constraint ensures that the column contains no null values. Columns withou t the NOT NULL constraint can contain null values by default.

The NOT NULL Constraint

Is defined at the column level
CREATE TABLE employees(
employee_id NUMBER(6),
System
last_name VARCHAR2(25) NOT NULL,
named
salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE

The UNIQUE Constraint

UNIQUE constraint
EMPLOYEES
INSERT INTO
Allowed
Not allowed:
already exists

The UNIQUE Constraint

Is defined at either the table level or the column level
CREATE TABLE employees(
employee_id NUMBER(6),
last_name VARCHAR2(25) NOT NULL,
email VARCHAR2(25),
salary NUMBER(8,2),
commission_pct NUMBER(2,2),
hire_date DATE NOT NULL,

...
CONSTRAINT emp_email_uk UNIQUE(email));

The PRIMARY KEY Constraint

DEPARTMENTS PRIMARY KEY
Not allowed (null value)
INSERT INTO
Not allowed
(50 already exists)
Is defined at either the table level or the column level

```
CREATE TABLE   departments(
department_id       NUMBER(4),
department_name     VARCHAR2(30)
CONSTRAINT dept_name_nn NOT NULL,
manager_id          NUMBER(6),
location_id         NUMBER(4),
CONSTRAINT dept_id_pk PRIMARY KEY(department_id));
```

The FOREIGN KEY Constraint

DEPARTMENTS PRIMARY KEY
EMPLOYEES FOREIGN KEY
Not allowed
INSERT INTO
(9 does not exist)
Allowed

The FOREIGN KEY Constraint

Is defined at either the table level or the column level

```
CREATE TABLE employees(
employee_id     NUMBER(6),
last_name       VARCHAR2(25) NOT NULL,
email           VARCHAR2(25),
salary          NUMBER(8,2),
commission_pct   NUMBER(2,2),
hire_date       DATE NOT NULL,
...
department_id    NUMBER(4),
CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
REFERENCES departments(department_id),
CONSTRAINT emp_email_uk UNIQUE(email));
```

FOREIGN KEY Constraint Keywords

•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

The CHECK Constraint

•Defines a condition that each row must satisfy
•The following expressions are not allowed:
–References to CURRVAL,NEXTVAL,LEVEL, and ROWNUM pseudocolumns
–Calls to SYSDATE ,UID,USER, and USERENV functions
–Queries that refer to other values in other rows

```
..., salary NUMBER(2)
CONSTRAINT emp_salary_min
CHECK (salary > 0),...
```

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);
```

Adding a Constraint

Add a FOREIGN KEY constraint to the EMPLOYEES table to indicate that a manager must already exist as a valid employee in the EMPLOYEES table.

```
ALTER TABLE     employees
ADD CONSTRAINT  emp_manager_fk
FOREIGN KEY(manager_id)
REFERENCES employees(employee_id);
```

Table altered.

Dropping a Constraint

• Remove the manager constraint from the EMPLOYEES table.

```
ALTER TABLE     employees
DROP CONSTRAINT  emp_manager_fk;
```

Table altered.

• Remove the PRIMARY KEY constraint on the DEPARTMENTS table and drop the associated FOREIGN KEY constraint on the EMPLOYEES.DEPARTMENT_ID column.

```
ALTER TABLE departments DROP PRIMARY KEY CASCADE;
```

Table altered.

Disabling Constraints

• Execute the DISABLE clause of the ALTER TABLE statement to deactivate an integrity constraint.

• Apply the CASCADE option to disable dependent integrity constraints.

```
ALTER TABLE employees
DISABLE CONSTRAINT emp_emp_id_pk CASCADE;
```

Table altered.

Enabling Constraints

•Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

```
ALTER TABLE employees
ENABLE CONSTRAINT emp_emp_id_pk;
```

Table altered.

•A UNIQUE or PRIMARY KEY index is automatically created if you enable a UNIQUE key or PRIMARY KEY constraint.

Cascading Constraints

•The CASCADE CONSTRAINTS clause is used along with the DROP COLUMN clause.

• The CASCADE CONSTRAINTS clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.

• The CASCADE CONSTRAINTS clause also drops all multicolumn constraints defined on the dropped columns.

Cascading Constraints

Example

```
ALTER TABLE test1
DROP (pk) CASCADE CONSTRAINTS;
```

Table altered.

```
ALTER TABLE test1
DROP (pk, fk, col1) CASCADE CONSTRAINTS;
```

Table altered.

Viewing Constraints

Query the USER_CONSTRAINTS
table to view all constraint definitions and names.

```
SELECT constraint_name, constraint_type,
search_condition
FROM user_constraints
WHERE table_name = 'EMPLOYEES';
```

Viewing Constraints

Viewing the Columns Associated with Constraints

View the columns associated with the constraint names in the USER_CONS_COLUMNS view.

```
SELECT constraint_name, column_name
FROM user_cons_columns
WHERE table_name = 'EMPLOYEES';
```

Practice 8

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk.

Hint: The constraint is enabled as soon as the ALTER TABLE command executes successfully.

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my_deptid_pk .

Hint:

The constraint is enabled as soon as the ALTER TABLE command executes successfully.

3. Add a column DEPT_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.

4. Confirm that the constraints were added by querying the USER_CONSTRAINTS view. Note the types and names of the constraints. Save your statement text in a file called lab10_4.sql.

5. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP and DEPT tables. Notice that the new tables and a new index were created.

If you have time, complete the following exercise:

6. Modify the EMP table. Add a COMMISSION column of NUMBER

data type, precision 2, scale 2. Add a constraint to the commission column that ensures that a commission value is greater than zero.

9- Creating Views

Objectives

After completing this lesson, you should be able to do the following:

•Describe a view

•Create, alter the definition of, and drop a view

•Retrieve data through a view

•Insert, update, and delete data through a view

Creating a View

•Create a view,EMPVU80, that contains details of employees in department 80.

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

View created.

•Describe the structure of the view by using the iSQL*Plus DESCRIBE command.

```
DESCRIBE empvu80
```

Creating a View

•Create a view by using column aliases in the subquery.

```
CREATE VIEW  salvu50
AS SELECT  employee_id ID_NUMBER, last_name NAME,
salary*12 ANN_SALARY
FROM    employees
WHERE   department_id = 50;
```

View created.

•Select the columns from this view by the given alias names.

Retrieving Data from a View

```
SELECT *
FROM salvu50;
```

Querying a View

Oracle Server iSQL*Plus USER_VIEWS

```
SELECT   *
EMPVU80
FROM      empvu80;
SELECT employee_id,
last_name, salary
FROM   employees
WHERE  department_id=80;
```

Modifying a View

•Modify the EMPVU80 view by using

CREATE OR REPLACE VIEW

clause. Add an alias for each column name.

```
CREATE OR REPLACE VIEW empvu80
(id_number, name, sal, department_id)
AS SELECT  employee_id, first_name || ' ' || last_name,
salary, department_id
FROM    employees
WHERE   department_id = 80;
```

View created.

•Column aliases in the CREATE VIEW clause are listed in the same order as the columns in the subquery.

Creating a Complex View

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

```
CREATE VIEW dept_sum_vu
(name, minsal, maxsal, avgsal)
AS SELECT d.department_name, MIN(e.salary),
MAX(e.salary),AVG(e.salary)
FROM      employees e, departments d
WHERE     e.department_id = d.department_id
GROUP BY  d.department_name;
```

View created.

Rules for Performing DML Operations on a View

•You can perform DML operations on simple views.

•You cannot remove a row if the view contains the following:

–Group functions

–A GROUP BY clause
–The DISTINCT keyword
– The pseudocolumn ROWNUM keyword

Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
(employee_number, employee_name, job_title)
AS SELECT employee_id, last_name, job_id
FROM     employees
WHERE    department_id = 10
WITH READ ONLY;
View created.
```

Removing a View

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

```
DROP VIEW view;
DROP VIEW empvu80;
View dropped.
```

Practice 09

1. Create a view called EMPLOYEES_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table.
Change the heading for the employee name to EMPLOYEE.
2. Display the contents of the EMPLOYEES_VU view.
3. Select the view name and text from the USER_VIEWS data dictionary view.

Note: Another view already exists. The EMP_DETAILS_VIEW was created as part of your schema.

Note: To see more contents of a LONG column, use the *i*SQL*Plus command SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

4. Using your EMPLOYEES_VU view, enter a query to display all employee names and department numbers.

5. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.
6. Display the structure and contents of the DEPT50 view.
7. Attempt to reassign Matos to department 80.

If you have time, complete the following exercise:

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the EMPLOYEES,DEPARTMENTS, and JOB_GRADES tables. Label the columns Employee,Department,Salary, andGrade, respectively.

PL/SQL Section

1-Declaring Variables

2-Writing Executable Statements

3-Interacting with the Oracle Server

4-Writing Control Structures

1-Declaring Variables

PL/SQL Block Structure

`DECLARE` (Optional)
Variables, cursors, user-defined exceptions
`BEGIN` (Mandatory)
– SQL statements
– PL/SQL statements
`EXCEPTION` (Optional)
Actions to perform when errors occur
`END;` (Mandatory)

Executing Statements and PL/SQL Blocks

```
DECLARE
v_variable VARCHAR2(5);
BEGIN
SELECT column_name
INTO v_variable
FROM table_name;
EXCEPTION
```

```
WHEN exception_name THEN
...
END;
```

Declaring PL/SQL Variables

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

```
DECLARE
v_hiredate DATE;
v_deptno NUMBER(2) NOT NULL := 10;
v_location VARCHAR2(13) := 'Atlanta';
c_comm CONSTANT NUMBER := 1400;
```

Base Scalar Data Types

- **CHAR [(*maximum_length*)]**
- **VARCHAR2 (*maximum_length*)**
- **LONG**
- **LONG RAW**
- **NUMBER [(*precision, scale*)]**
- **BINARY_INTEGER**
- **PLS_INTEGER**
- **BOOLEAN**

```
DECLARE
v_job VARCHAR2(9);
v_count BINARY_INTEGER := 0;
v_total_sal NUMBER(9,2) := 0;
v_orderdate DATE := SYSDATE + 7;
c_tax_rate CONSTANT NUMBER(3,2) := 8.25;
v_valid BOOLEAN NOT NULL := TRUE;
```

Declaring Variables with the %TYPE Attribute

```
identifier Table.column_name%TYPE;
```

```
...
v_name employees.last_name%TYPE;
v_balance NUMBER(7,2);
v_min_balance v_balance%TYPE := 10;
...
```

Bind Variables

```
VARIABLE g_salary NUMBER
BEGIN
SELECT salary
INTO :g_salary
FROM employees
WHERE employee_id = 178;
END;
/
PRINT g_salary
```

DBMS_OUTPUT.PUT_LIN

```
SET SERVEROUTPUT ON
DEFINE p_annual_sal = 60000

DECLARE
v_sal NUMBER(9,2) := &p_annual_sal;
BEGIN
v_sal := v_sal/12;
DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' ||
TO_CHAR(v_sal));
END;
/
```

Practice 1

1. Evaluate each of the following decla rations. Deter mine which of them are *not* legal and explain why.

a. DECLARE v_id NUMBER(4);
b. DECLARE v_x, v_y, v_z VARCHAR2(10);
c. DECLARE v_birthdate DATE NOT NULL;
d. DECLARE v_in_stock BOOLEAN := 1;

2. In each of the following assignments, indicate whether the statement is valid and what the valid data type of the result will be.

a.v_days_to_go := v_due_date - SYSDATE;
b.v_sender := USER || ': ' || TO_CHAR(v_dept_no);
c.v_sum := $100,000 + $250,000;
d.v_flag := TRUE;
e.v_n1 := v_n2 > (2 * v_n3);
f.v_value := NULL;
If you have time, c omplete the following exercise:
4. Create a block that declares two variables. Assign the value of these PL/SQL variables to*i*SQL*Plus host variables and print the results of the PL/SQL variables to the screen. Executeyour PL/SQL block. Save your PL/SQL block in a file named p1q4.sql , by clicking theSave Script button. Remember to save the script with a .sql extension.
V_CHAR Character (variable length) V_NUM Number
Assign values to thes e variables as foll ows:
Variable Value
-------- ------------------------------------
V_CHAR The literal '42 is the answer'
V_NUM The first two characters from V_CHAR

2-Writing Executable Statements

Data Type Conversion

Conversion functions:

– **TO_CHAR**

– **TO_DATE**

- **TO_NUMBER**

```
DECLARE
v_date DATE := TO_DATE('12-JAN-2001', 'DD-MON-YYYY');
BEGIN
   –  . . .
```

Practice 2

PL/SQL Block

```
DECLARE
v_weight NUMBER(3) := 600;
v_message VARCHAR2(255) := 'Product 10012';
BEGIN
DECLARE
v_weight NUMBER(3) := 1;
v_message VARCHAR2(255) := 'Product 11001';
v_new_locn VARCHAR2(50) := 'Europe';
BEGIN
v_weight := v_weight + 1;
v_new_locn := 'Western ' || v_new_locn;
1
END;
v_weight := v_weight + 1;
v_message := v_message || ' is in stock';
v_new_locn := 'Western ' || v_new_locn;
2
END;
```

/
1. Evaluate the PL/SQL block above and determine the data type and value of each of the following variables a ccording to the rules of scoping.
a. The value of V_WEIGHT at position 1 is:
b. The value of V_NEW_LOCN at position 1 is:
c. The value of V_WEIGHT at position 2 is:
d. The value of V_MESSAGE at position 2 is:
e. The value of V_NEW_LOCN at position 2 is:

Scope Example

```
DECLARE
v_customer VARCHAR2(50) := 'Womansport';
v_credit_rating VARCHAR2(50) := 'EXCELLENT';
BEGIN
DECLARE
v_customer NUMBER(7) := 201;
v_name VARCHAR2(25) := 'Unisports';
BEGIN
v_customer v_name v_credit_rating
END;
v_customer v_name v_credit_rating
END;
/
```

2. Suppose you embed a subblock within a block, as shown above. You declare two variables,
V_CUSTOMER and V_CREDIT_RATING , in the main block. You also declare two variables,
V_CUSTOMER and V_NAME , in the subblock. Determine the values and data types for each of
the following cases.
a. The value of V_CUSTOMER in the subblock is:
b. The value of V_NAME in the subblock is:
c. The value of V_CREDIT_RATING in the subbloc k is:
d. The value of V_CUSTOMER in the main block is:
e. The value of V_NAME in the main block is:
f. The value of V_CREDIT_RATING in the main block is:

3. Create and execute a PL/SQL block that accepts two numbers through
*i*SQL*Plus substitution variables.
a. Use the DEFINE commandtoprovidethetwovalues.
DEFINE p_num1 =**2**
DEFINE p_num2 =**4**
b. Pass the two values defined in step a above, to the PL/SQL block through
*i*SQL*Plus substitution variables. The first number should be divided by the second

number and ha ve the
second number added to the result. The result should be stored in a PL/SQL variable and printed on the screen.
Note:
SET VERIFY OFF in t he P L/S QL block.
4. Build a PL/SQL block that computes the total compensation for one year.
a. The annual salary and the annual bonus percentage values are defined using the DEFINE comma nd.
b. Pass the values defined in the above step to the PL/SQL block through *i*SQL*Plus substitutionvaria bles. The bonus must be converted from a whole number to a decimal (for example, 15 to.15). If the salary is null , set it to zero before computing the total compensation. Execute the
P L/ SQL block.
Reminder: Use the NVL function to ha ndle null values.
Note: Total compensation is the sum of the annual salary and the annual bonus.
To test the NVL function, set the DEFINE variable equal to NULL.
DEFINE p_salary =**50000**
DEFINE p_bonus =**10**

3-Interacting with the Oracle Server

SELECT Statements in PL/SQL

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

```
DECLARE
v_deptno NUMBER(4);
v_location_id NUMBER(4);
BEGIN
SELECT department_id, location_id
INTO v_deptno, v_location_id
FROM departments
WHERE department_name = 'Sales';
...
```

```
END;
   /
```

Retrieving Data in PL/SQL

```
DECLARE
v_hire_date employees.hire_date%TYPE;
v_salary employees.salary%TYPE;
BEGIN
SELECT hire_date, salary
INTO v_hire_date, v_salary
FROM employees
WHERE employee_id = 100;
...
END;
   /
```

```
   Return the sum of the salaries for all employees in

   the specified department.

SET SERVEROUTPUT ON
DECLARE
v_sum_sal NUMBER(10,2);
v_deptno NUMBER NOT NULL := 60;
BEGIN
SELECT SUM(salary) -- group function
INTO v_sum_sal
FROM employees
WHERE department_id = v_deptno;
DBMS_OUTPUT.PUT_LINE ('The sum salary is ' ||
TO_CHAR(v_sum_sal));
END;
   /
```

Manipulating Data Using PL/SQL

Inserting Data

```
BEGIN
INSERT INTO employees
(employee_id, first_name, last_name,
email,
```

```
hire_date, job_id, salary)
VALUES
(employees_seq.NEXTVAL, 'Ruth',
'Cores', 'RCORES',
sysdate, 'AD_ASST', 4000);
END;
  /
```

Updating Data

```
DECLARE
v_sal_increase employees.salary%TYPE :=
800;
BEGIN
UPDATE employees
SET salary = salary + v_sal_increase
WHERE job_id = 'ST_CLERK';
END;
  /
```

D e l e t i n g D a t a

```
DECLARE
v_deptno employees.department_id%TYPE
:= 10;
BEGIN
DELETE FROM employees
WHERE department_id = v_deptno;
END;
  /
```

Merging Rows

Insert or update rows in the COPY_EMP table to match

the EMPLOYEES table.

```
DECLARE
v_empno employees.employee_id%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
USING employees e
ON (e.employee_id = v_empno)
WHEN MATCHED THEN
UPDATE SET
c.first_name = e.first_name,
c.last_name = e.last_name,
c.email = e.email,
. . .
WHEN NOT MATCHED THEN
INSERT VALUES(e.employee_id, e.first_name,
e.last_name,
. . .,e.department_id);
  END;
```

SQL Cursor Attributes

Delete rows that have the specified employee ID from

the EMPLOYEES table. Print the number of rows

deleted.

Example:

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
v_employee_id
employees.employee_id%TYPE := 176;
BEGIN
DELETE FROM employees
WHERE employee_id = v_employee_id;
:rows_deleted := (SQL%ROWCOUNT ||
' row deleted.');
END;
/
  PRINT rows_deleted
```

Practice 3

1. Create a PL/SQL block that selects the maximum department number in the DEPARTMENTS able and stores it in an *i*SQL*Plus variable. Print the results to the screen. Save your PL/SQL block in a file na med 3q1.sql.by clicking the Save Script button. S ave t he script wit h a .sql extension.
2. Modify the PL/SQL block you created in exercise 1 to insert a new department into the DEPARTMENTS table. Save the PL/S QL block in a fil e na med p3q2.sql
by clicking the Save Script but ton. Save the script with a .sqlextension.
a. Use the DEFINE command to provide the department name. Name the new department Education.
b. Pass the value defined for the department name to the PL/SQL block through a *i*SQL*Plus substitution variable. Rather than print ing the department number retrieved from exercise 1, a dd 10 to it and use it as the department number for the new department.
c. Leave the location number as null for now.
d. Execute the PL/SQL block.
e. Display t he new depart ment that you creat ed.
3. Create a PL/SQL block that updates the location ID for the new department that you added in the previous pract ice. Save your PL/SQL block in a file named p3q3.sql
by clic king the Save Script butt on. Save the script with a .sql ext ension.

a. Use an *i*SQL*Plus variable for the department ID number that you added in the previous practice.
b. Use the DEFINE command to provide the location ID. Name the new location ID 1700.
DEFINE p_deptno = 280
DEFINE p_loc = 1700
c. Pass the value to the PL/SQL block through a
*i*SQL*Plus substitution variable. Test the PL/SQL block.
d. Display the department that you updated.
Practice 3 (continued)
4. Create a PL/SQL block that deletes the department that you created in exercise 2. Save the PL/SQLblockinafilenamedp3q4.sql.
by clicking the Save Script button. Save the script with a .sql extension.
a. Use the DEFINE commandtoprovidethedepartmentID.
DEFINE p_deptno=**280**
b. Pass the value to the PL/SQL block through a
*i*SQL*Plus substitution variable. Print to t he screen the number of rows affect ed.
c. Test the PL/SQL block.
d. Confirm that t he department has been delet ed.

4-Writing Control Structures

IF Statements

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

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

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

IF-THEN-ELSE Statements

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

IF-THEN-ELSIF Statements

```
. . .
IF v_start > 100 THEN
v_start := 0.2 * v_start;
ELSIF v_start >= 50 THEN
v_start := 0.5 * v_start;
ELSE
v_start := 0.1 * v_start;
END IF;
. . .
```

CASE Expressions

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

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

```
DBMS_OUTPUT.PUT_LINE ('Grade: '||
v_grade || '
Appraisal ' || v_appraisal);
END;
/
```

Iterative Control: LOOP Statements

There are three loop types:

– Basic loop

– FOR loop

- **WHILE loop**

Basic Loops

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

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

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

WHILE Loops

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

```
DECLARE
v_country_id locations.country_id%TYPE :=
'CA';
v_location_id locations.location_id%TYPE;
```

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

FOR Loops

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

```
DECLARE
v_country_id locations.country_id%TYPE :=
'CA';
v_location_id locations.location_id%TYPE;
v_city locations.city%TYPE := 'Montreal';
```

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

Practice 4

1. Execute the command in the file lab04_1.sql to create the MESSAGES table. Write a PL/SQL block to insert numbers into the MESSAGES table.

a. Insert the numbers 1 to 10, excluding 6 and 8.

b. Commit before the end of the block.

c. Select from the MESSAGES table to verify tha t your P L/SQL block worked.

2. Create a PL/SQL block that computes the commission amount for a given employee based on the employee's salary.

a. Use the DEFINE command to provide the employee ID. Pass the value to the PL/SQL block through a *i*SQL*Plus substitution variable.

DEFINE p_empno = 100

b. If the employee's sa lary is less than $5,000, display the bonus amount for the employee as 10% of the salary.
c. If the employee's salary is between $5,000 and $10,000, displa y the bonus amount for the employee as 15% of the salary.
d. If the employee's sa lary exceeds $10,000, display the bonus amount for the employee as
20% of the salary.
e. If the employee's salary is NULL, display the bonus amount for the employee as 0.
f. Test the PL/SQL block for each case using the following test cases, and check each bonus amount.
Note:Include SET VERIFY OFF in your solution.
Employee Number Salary Resulting Bonus
100 24000 4800
149 10500 2100
178 7000 1050

Table altered.

EMPLOYEE_ID	SALARY	STARS
104	6000	******
174	11000	**********
176	8600	*********

Practice 4 (continued)
If you have time, c omplete the following exercises:
3. Create an EMP table that is a replica of the
EMPLOYEES table. You can do this by executing the
script lab04_3.sql .Addanewcolumn, STARS ,of VARCHAR2
data type and length of 50 to the EMP ta ble for storing asterisk (*).
4. Create a PL/SQL block that rewards an employee by appending an asterisk in the STARS column for ever y $1000 of the employee's salary. Save your PL/SQL block in a file called
p4q4.sql by clicking on the Save Script button. Remember to save the script with a.sql extension.
a. Use the DEFINE command to provide the employee ID. Pass the value to the PL/SQL block through a *i*SQL*Plus substitution variable.
DEFINE p_empno=104
b. Init ialize a v_asterisk variable t hat contains a NULL.
c. Append an asterisk to the string for every $1000 of the salary amount. For example, if the employee has a salary amount of $8000, the string of asterisks should contain eight asterisks. If the employee has a salary amount of $12500, the string of asterisks should contain 13 ast erisks.
d. Update the STARS column for the employee wit h the string of asterisks.
e. Commit.
f. Test the block for the following values:
DEFINE p_empno=174
DEFINE p_empno=176

g. Display the rows from the EMP table to verify whether your PL/SQL block has executed
successfully.

Note:

SET VERIFY OFF
in the PL/SQL block