DATABASE MANAGEMENT AND ADMINISTRATION

IAC
Information Access Center

Bangladesh–Korea
Information Access Center

Department of Computer Science and Engineering (CSE)
Bangladesh University of Engineering and Technology (BUET)

A brief Introduction

to

Oracle SQL/PL-SQL

***Database Management and Administration* Course**

Bangladesh Korea Information Access Center (BK-IAC), Dept. of CSE, BUET

Author

Sukarna Barua

Assistant Professor

Dept. of CSE

Bangladesh University of Engineering and Technology (BUET)

Dhaka-1000, Bangaldesh.

This tutorial is indented for –

- Beginners who want to learn the basics of Oracle SQL and PL/SQL.
- Learning starting knowledge that can help one to start building Oracle SQL and PL/SQL applications.

This tutorial is not intended for –

- Experts who already have a good knowledge of Oracle SQL and PL/SQL.
- Working as a complete reference of Oracle SQL and PL/SQL.

Contents

Chapter 1: Introduction

1. Introduction

What is SQL

Structured Query Language (SQL) is the language used to interaction with a database management system. The language defines the statements for retrieving data from the database, updating data in the database, inserting data into the database, and more similar things.

SQL is the way by which all programs and users access data in an Oracle database. SQL provides statements for a variety of tasks including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects
- Controlling access to the database, and its objects
- Guaranteeing database consistency and integrity

SQL Statements

All SQL statements supported by Oracle database can be grouped in several categories as illustrated in following table:

SQL statement	Function of the statement
SELECT	Retrieves data from database
INSERT UPDATE DELETE	Inserts data, updates data, and deletes data in database
CREATE ALTER DROP	Creates new tables in database, alters existing tables, deletes tables from database
COMMIT ROLLBACK	Saves data permanently in database, erases changes done form database

SQL Working Tools

There are two primary tools that can be used for SQL. They are:

- SQL * PLUS – This is a command line tool and is the most popular tool
- SQL Developer – This is a graphical tool used by Oracle users to interact with Oracle database.
- Navicat – This is a third party tool that is used to connect to several databases such as Oracle, MySQL, etc. This is one of the most popular GUI tools to interact with database.

The HR schema

The Human Resource (HR) schema is a part of Oracle sample database that can be installed with Oracle. This course uses the tables in HR schema for all practice and homework sessions. The schema contains the following tables:

- REGIONS – contain rows that represent a region such as America, Asia, and so on.
- COUNTRIES – contain rows for countries each of which is associated with a region
- LOCATIONS – contains the specific address of a specific office, warehouse, or a production site of a company in a particular country
- DEPARTMETNS – shows detail about a department in which an employee works. Each department may have a relationship representing the department manager in the EMPLOYEES table.
- EMPLOYEES – contain detail about each employee working for a department. Some employees may not be assigned to any department.
- JOBS – contain the job types that can be held by each employee.
- JOB_HISTORY – contain the job history of the employees. If an employee changes department within a job, or changes a job within a department, a new row is inserted in this table with the earlier job information of the employee.

The following table shows the column names for all tables in HR schema.

Table Name	**Column Names**
REGIONS	REGION_ID, REGION_NAME
COUNTRIES	COUNTRY_ID, COUNTRY_NAME, REGION_ID
LOCATIONS	LOCATION_ID, STREET_ADDRESS, POSTAL_CODE, CITY,

	STATE_PROVINCE, COUNTRY_ID
DEPARTMENTS	DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID
EMPLOYEES	EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, SALARY, COMMISSION_PCT, MANAGER_ID, DEPARTMENT_ID
JOB_HISTORY	EMPLOYEE_ID, START_DATE, END_DATE, JOB_ID, DEPARTMENT_ID
JOBS	JOB_ID, JOB_TITLE, MIN_SALARY, MAX_SALARY

CHAPTER 2: Retrieve Data from Database Tables

1. Selecting Data Using Basic SELECT Statement

Basic SELECT statement

The SELECT statement is used to retrieve data from database tables. The very basic general form of the SELECT statement is given below:

```
SELECT Column1, Column2, …
FROM table_name ;
```

For example, the following SELECT statement retrieves some data from EMPLOYEES table.

```
SELECT EMPLOYEE_ID, LAST_NAME, SALARY
FROM EMPLOYEES ;
```

The above SELECT statement –

- Will retrieve all rows from the EMPLOYEES table
- Will retrieve values of three columns only, EMPLOYEE_ID, LAST_NAME, SALARY

If you want select values of all columns you can either specify all column names or specify a '*' as follows:

```
SELECT * FROM EMPLOYEES ;
```

Expressions instead of column names

You can also use expressions instead of direct column names in the SELECT clause as shown below. The query will multiply values of SALARY column by 12, and then will display the results.

```
SELECT EMPLOYEE_ID, LAST_NAME, SALARY*12
FROM EMPLOYEES ;
```

You can use arithmetic expressions involving addition, subtraction, multiplication, and division as shown in following SELECT statements.

```
SELECT EMPLOYEE_ID, LAST_NAME, SALARY*12
FROM EMPLOYEES ;

SELECT EMPLOYEE_ID, LAST_NAME, SALARY+12
FROM EMPLOYEES ;

SELECT EMPLOYEE_ID, LAST_NAME, (SALARY-1000)*5
FROM EMPLOYEES ;

SELECT EMPLOYEE_ID, LAST_NAME, (SALARY + SALARY*0.15) / 1000
FROM EMPLOYEES ;
```

Use of column alias

You will notice that, when you execute the first SELECT query above with expressions instead of column names, the displayed column header in the result is SALARY*12. These headers sometime become misleading and unreadable. Using alias, i.e., naming the expression, will solve the problem. You can give meaning names to expressions in this way. The following example illustrates this and gives the expression a name ANNSAL. The query results will be displayed with ANNSAL as the column header.

```
SELECT EMPLOYEE_ID, LAST_NAME, SALARY*12 ANNSAL
FROM EMPLOYEES ;
```

The column alias should be double quoted if it contains spaces as shown below:

```
SELECT EMPLOYEE_ID, LAST_NAME, SALARY*12 "ANNUAL SALARY"
FROM EMPLOYEES ;
```

You should remember the following when writing SQL statements.

- All SQL statements have a terminator, i.e., semicolon (;) at the end
- SQL Keywords are not case-sensitive, e.g., SELECT, Select, and select are all same.
- SQL statements can be entered in one or more lines to increase readability

NULL values in columns

In Oracle, a NULL value -

- Means a value in the column which is unassigned.
- NULL value means empty value, i.e., no value was given
- NULL value is not same as zero value or some other specific value, it is simply unknown
- Arithmetic expressions involving a NULL value outputs NULL

Execute the following SELECT statement and search through the values of COMMISSION_PCT column in the output. You will notice that there are many NULL, i.e., empty values there.

```
SELECT LAST_NAME, COMMISSION_PCT
FROM EMPLOYEES ;
```

The following example illustrate that when a column value is NULL, then arithmetic expression also outputs NULL. You will notice that SALCOMM column in the output is null in those rows where either SALARY value is NULL or COMMISSION_PCT value is NULL.

```
SELECT LAST_NAME, (SALARY+SALARY*COMMISSION_PCT) SALCOMM
FROM EMPLOYEES ;
```

Use of text concatenation operator (||)

The concatenation operation, || is used to join multiple text values. For example, the following query outputs employee's first name and last name joined together and separated by a space. The output column header is also given a new name FULLNAME. Although, the parenthesis is not necessary, it improves readable of the statement.

```
SELECT (FIRST_NAME || LAST_NAME) FULLNAME, SALARY
FROM EMPLOYEES ;
```

You can also add additional texts with the column values as the following statement does.

```
SELECT ('NAME is: ' || FIRST_NAME || ' ' || LAST_NAME) FULLNAME, SALARY
FROM EMPLOYEES ;
```

Note the use of single quotes in the above statement. The single quote is used to mean text values. This is necessary and without the single quote, Oracle will think the text as a column name, which is definitely wrong in this case. So, you must use single quote around text values (not around column names). However, no quote is used for numeric values, e.g., 350, and 7777.

Use of DISTINCT keyword for removing duplicate column values

Consider the output of the following SELECT statements.

```
SELECT JOB_ID
FROM EMPLOYEES ;
```

You will notice that, the output contains similar, i.e., duplicate values since several rows (employees) have the same JOB_ID in the EMPLOYEES table. You can remove the duplicate outputs by using the DISTINCT keyword as shown below.

```
SELECT DISTINCT JOB_ID
FROM EMPLOYEES ;
```

In the above SELECT, output will contain unique values of JOB_ID therefore removing duplicate outputs. The DISTINCT Keyword works on all columns specified in the SELECT and outputs unique rows (removes duplicate rows).

```
SELECT DISTINCT DEPARTMENT_ID, JOB_ID
FROM EMPLOYEES ;
```

Use of DESCRIBE statement

You can use the DESCRIBE statement to view the structure, i.e., column names and data types of different columns of a table. The following example will output all column names, their data types, and all constraints of the EMPLOYEES table.

```
DESCRIBE EMPLOYEES ;
```

The basic data types used in Oracle are given in following table.

Data Type	**Description**
NUMBER	Used for storing numeric values
VARCHAR2(size)	Used for storing variable length text data, where parameter *size* specifies the maximum characters in the text
CHAR(size)	Used for storing fixed-length text data where *size* is the number of characters
DATE	Used for storing date values

Practice 2.1

a. Write an SQL query to retrieve all country names.

b. Write an SQL query to retrieve all job titles.

c. Write an SQL query to retrieve all MANAGER_IDs.

d. Write an SQL query to retrieve all city names. Remove duplicate outputs.

e. Write an SQL query to retrieve LOCATION_ID, ADDRESS from LOCATIONS table. The ADDRESS should print each location in the following format: STREET_ADDRESS, CITY, STATE_PROVINCE, POSTAL_CODE.

2. SELECT Statement with WHERE clause

Use of WHERE clause in SELECT Statement

The WHERE clause is used with one or more conditions to limit rows in the output. The general syntax of the SELECT statement with WHERE clause is given below.

```
SELECT …
FROM table_name
WHERE condition ;
```

For example, the following SELECT retrieves last names and salaries of those employees only whose DEPARTMENT_ID column value is 80.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80 ;
```

The following example shows conditions involving text and date values.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME = 'Whalen' ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE HIRE_DATE =  '01-JAN-1995' ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE COMMISSION_PCT IS NULL ;
```

Note that use of single quotes around text data and date data ('01-JAN-1995'). The date text given in the query is in default format. Until, you learn TO_DATE conversion function in later chapters, you will need to enter data values in this way in default format. Otherwise, Oracle may generate error messages.

Comparison operators in Oracle

The condition in the above WHERE clause uses the equal ('=') operator to compare DEPARTMENT_ID values. This is a comparison operator available in Oracle. The most commonly used comparison operators are:

Operator	**Description**
=	Equal to

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to
BETWEEN value1 AND value	Between value1 and value2 (inclusive)
IN (value1, value2, …)	Equal to value1 or value 2 or …
LIKE	Pattern matching for text
IS NULL	Is a NULL value

Range condition using BETWEEN operator

You can check whether a value is in the given range by using the BETWEEN operator. The following statements illustrate this.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY BETWEEN 5000 AND 10000 ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE HIRE_DATE BETWEEN '01-JAN-1990' AND '31-DEC-1995' ;
```

Use of IN operator to match the column value against a set of values

The IN operator can be used to check whether a column value is equal to a set of values. The following examples illustrate this.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID IN (50, 60, 70, 80, 90, 100) ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME IN ('Ernst', 'Austin', 'Pataballa', 'Lorentz') ;
```

Joining multiple conditions in WHERE clause

You can join two or more conditions using AND, OR, and NOT operators. You may need to use parenthesis to enforce operation execution order.

The following example shows the use of AND, OR and NOT operators.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80 AND SALARY > 5000 ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE JOB_ID = 'SALES_REP' OR SALARY >= 10000 ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE COMMISSION_PCT IS NOT NULL ;
```

Three or more conditions can be joined. In such cases, parenthesis should be used to clarify the execution order and combination of the conditions.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE DEPARTMENT_ID <> 80 AND
(SALARY > 5000 OR COMMISSION_PCT IS NOT NULL)  ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE (DEPARTMENT_ID = 80 AND SALARY > 5000) OR
COMMISSION_PCT IS NOT NULL)  ;
```

Note the difference in outputs of the above two statements. The use of parenthesis change the order of execution of the operators, therefore changes meaning and output of the SELECT statements.

Pattern matching in texts using LIKE operator

The LIKE operator is used for pattern matching in texts. Suppose, you want to retrieve those employee records whose last names contain the character, 's'. This type of condition is expressed using LIKE operators and two special symbols ('%' and '_'). For example, the following statement will retrieve all retrieve records of those employees whose last name column contain at least one 's'.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE '%s%' ;
```

Note the use of special symbol ('%') above. The '%' means zero or more characters. A '%' before 's' means zero or more characters can precede the 's', a '%' after 's' means zero or more characters can follow 's'. To understand the position of '%' and resultant effect, observe the outputs of the following queries.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE 'S%' ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE '%s' ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE 's' ;
```

If you execute the above three statements and observe the output, you will note that-

- The first statement retrieves those rows where LAST_NAME starts with a 'S'. Any number of any characters can follow after the 'S'. Note that text match is case-sensitive, hence 'S%' and 's%' are not same.
- The second statement retrieves those rows where LAST_NAME ends with 's'. Any number of any characters can precede the 's'. Only the last character must be 's'.
- The third statement retrieves those rows where LAST_NAME is exactly 's' (like the '=' operator)

Like the '%' special symbol, the '_' special symbol is used to match exactly one character. For example, execute the following statements and observe the outputs.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE 'a_' ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE '_ _ b' ;

SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE LAST_NAME LIKE '_%' ;
```

If you execute the above three statements and observe the output, you will note that-

- The first statement retrieves those rows where LAST_NAME contains exactly two characters where the first character must be 'a', e.g., 'ab', 'ak', and 'ac'.
- The second statement retrieves those rows where LAST_NAME contains exactly three characters in which the last one must be 'b'
- The third statement retrieves those rows where LAST_NAME contains at least one character, i.e., last name cannot be empty text

NULL values in comparison

In a comparison statement, if anything is NULL, than the comparison is regarded as FALSE, and corresponding row is not retrieved. Consider the following statement.

```
SELECT LAST_NAME, SALARY, COMMISSION_PCT
FROM EMPLOYEES
WHERE COMMISSION_PCT <  0.20 ;
```

The above statement retrieves records of those employees whose COMMISSION_PCT value is less than 0.20. The query should retrieve those rows that have NULL values in COMMISSION_PCT column,

because NULL should mean a 0 value in most cases. However, Oracle would not retrieve those rows. This is because; comparison with NULL is regarded as FALSE (not matched). If you want to retrieve records with NULL values also, then you have to use IS NULL comparison operator as shown below.

```
SELECT LAST_NAME, SALARY, COMMISSION_PCT
FROM EMPLOYEES
WHERE COMMISSION_PCT < 0.20 OR COMMISSION_PCT IS NULL ;
```

A common mistake is using equality (=) operator to retrieve records with NULL values as shown below. But, the query will retrieve no rows as the equality comparison fails due to NULL value.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE COMMISSION_PCT = NULL ;
```

Practice 2.2

a. Select names of all employees who have joined before January 01, 1998.
b. Select all locations in the following countries: Canada, Germany, United Kingdom.
c. Select first names of all employees who do not get any commission.
d. Select first names of employees whose last name starts with an 'a'.
e. Select first names of employees whose last name starts with an 's' and ends with an 'n'.
f. Select all department names whose MANAGER_ID is 100.
g. Select all names of employees whose job type is 'AD_PRES' and whose salary is at least 23000.
h. Select names of all employees whose last name do not contain the character 's'.
i. Select names and COMMISSION_PCT of all employees whose commission is at most 0.30.
j. Select names of all employees who have joined after January 01, 1998.
k. Select names of all employees who have joined in the year 1998.

3. Sorting Rows in the Output

ORDER BY Clause

You can use the ORDER BY clause with the SELECT statement to sort the rows in the results. The general syntax is given below:

```
SELECT …
FROM table_name
WHERE …
ORDER BY Column1 [ASC | DESC], Column2 [ASC | DESC], …
```

The ORDER BY clause is used after the WHERE clause to specify sorting order. The following statement retrieves records of employees and sorts the output in descending order of SALARY values.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
ORDER BY SALARY DESC ;

SELECT LAST_NAME, SALARY, HIRE_DATE
FROM EMPLOYEES
ORDER BY HIRE_DATE ASC ;
```

Outputs can be sorted based on multiple columns. So, if the first column is equal for multiple rows, then the rows are sorted according to the second column. If the second column is also same, then rows are sorted based on third column, and so on. The following statement illustrates such queries.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
ORDER BY SALARY DESC, LAST_NAME ASC ;
```

You can use column alias in ordering results. The following examples illustrate this.

```
SELECT JOB_TITLE, (MAX_SALARY - MAX_SALARY) DIFF_SALARY
FROM JOBS
ORDER BY DIFF_SALARY DESC ;
```

Practice 2.3

a. Select names, salary, and commissions of all employees of job type 'AD_PRES'. Sort the result in ascending order of commission and then descending order of salary.

b. Retrieve all country names in lexicographical ascending order.

4. More Practice Problems

In class.

Chapter 3: Use of Oracle Single Row Functions

1. Character Functions

Case conversion functions

The case conversion functions are used to convert case of text characters. The most commonly used Oracle functions for this purpose are given below:

Function	Description
LOWER (text)	Converts the text to all lowercase. Here, text can be column name or an expression.
UPPER (text)	Converts the text to all uppercase. Here, text can be column name or an expression.
INITCAP (text)	Converts the first character of the text to uppercase. Here, text can be column name or an expression.

The following table shows the outputs of applying case conversion functions on the text 'hello WORLD'.

Function	Output
LOWER('hello WORLD')	hello world
UPPER('hello WORLD')	HELLO WORLD
INITCAP('hello WORLD')	Hello world

The following statement shows the use case-conversion functions in SELECT statements.

```
SELECT (INITCAP(FIRST_NAME) ||  LOWER(LAST_NAME)) NAME, UPPER(JOB_ID) JOB
FROM EMPLOYEES ;
```

Note the use of three functions in the above statement. The column aliases are good ways to redefine column headers.

Case conversion functions are frequently used in WHERE clause to match specific texts. For example, if you want to retrieve records of those departments which contain the text SALE in its name, then you may write the following statement.

```
SELECT *
FROM DEPARTMENTS
WHERE UPPER(DEPARTMENT_NAME) LIKE '%SALE%' ;

SELECT *
FROM DEPARTMENTS
WHERE LOWER(DEPARTMENT_NAME) LIKE '%sale%' ;
```

Any of the above two statements will retrieve the required records successfully. However, if no functions were used, then the query may not retrieve all records due to case differences. For examples, if there were two departments named as 'Sales Office' and 'Whole sale', then comparing without functions may not be able to retrieve both records simultaneously.

Character manipulation functions

The most commonly used functions in this category are given below:

Function	**Description**
CONCAT (Column1, Column2)	Concatenates, i.e., joins the texts of two columns.
SUBSTR(Column, m [, n])	Extracts n characters from the Column, starting from m-th position. If n is omitted, the all characters from the starting position are extracted.
LENGTH (Column)	Outputs the length, i.e., number of characters of text.
INSTR(Column, 'text')	Returns the numeric position of the 'text' in Column if found. Otherwise, returns 0.
LPAD(Column, n, 'text')	Returns a left-padded text of n characters, padded with 'text'.
RPAD(Column, n, 'text')	Returns a right-padded text of n characters, padded with 'text'
TRIM(Column)	Trims, i.e., removes whitespace characters from left and right.
REPLACE(Column,'text1','text2')	Searches for 'text1' in Column, and if found, replaces with 'text2'.

The following table shows the output of applying these functions.

Function	**Output**
CONCAT('Hello', ' world')	Hello world
SUBSTR('Hello world', 7)	World
SUBSTR('Hello world', 1, 5)	Hello
INSTR('Hello world', 'world')	7
LPAD('12345', 10, '*')	*****12345
RPAD('12345', 10, '*')	12345*****
TRIM(' Hello world ')	Hello world
REPLACE('Hesso worsd', 's', 'l')	Hello world

The following statements show the use of the above functions in SELECT query.

```
SELECT CONCAT(FIRST_NAME, LAST_NAME) NAME, JOB_ID
FROM EMPLOYEES
WHERE INSTR( UPPER(JOB_ID), 'CLERK') > 0 ;

SELECT ( SUBSTR(FIRST_NAME, 1, 1) || '.' || SUBSTR(LAST_NAME, 1, 1) || '.' ) ABBR
FROM EMPLOYEES ;

SELECT INITCAP( TRIM(LAST_NAME) )  NAME, LPAD(SALARY, 10, 0)
FROM EMPLOYEES ;
```

The first query retrieves name and job id of all employees whose job id field contains the word 'CLERK'. The second query outputs the abbreviation of all employee names. The third query prints the last name and padded salary in 10 character width.

Practice 3.1

a. Print the first three characters and last three characters of all country names. Print in capital letters.

b. Print all employee full names (first name followed by a space then followed by last name). All names should be printed in width of 60 characters and left padded with '*' symbol for names less than 60 characters.

c. Print all job titles that contain the text 'manager'.

2. Number functions

ROUND, TRUNC, and MOD functions

There are three number functions most commonly used in Oracle. They are given below.

Function	Description
ROUND (Column, n)	Rounds the value in Column up-to n decimal places after the decimal point. If n is 0, then rounding occurs in the digit before the decimal point.
TRUNC (Column, n)	Truncates the value up-to n decimal places after the decimal point. If n is 0, the truncation occurs up-to the digit before the decimal point.
MOD(m, n)	Returns the value of the remainder for m divided by n

The following table shows the results of applying numeric functions.

Function	Output
ROUND(45.923, 2)	45.92
ROUND(45.926, 2)	45.93
ROUND(45.926, 0)	46
TRUNC(45.923, 2)	45.92
TRUNC(45.926, 2)	45.92
TRUNC(45.926, 0)	45
MOD(23, 5)	3

The following statements show the use of numeric functions.

```
SELECT LAST_NAME, ROUND(SALARY/1000, 2) "SALARY (IN THOUSANDS)"
FROM EMPLOYEES
WHERE INSTR(JOB_ID, 'CLERK') > 0 ;

SELECT LAST_NAME, ROUND( (SYSDATE - HIRE_DATE)/7, 4) WEEKS_EMPLOYED
FROM EMPLOYEES
```

```
WHERE DEPARTMENT_ID = 80 ;


SELECT LAST_NAME, TRUNC(SALARY/1000, 0) || ' thousands ' ||
TRUNC( MOD(SALARY,1000)/100, 0) || ' hundreds ' || MOD(SALARY,100) || ' taka
only'
FROM EMPLOYEES ;
```

Practice 3.2

a. Print employee last name and number of days employed. Print the second information rounded up to 2 decimal places.

b. Print employee last name and number of years employed. Print the second information truncated up to 3 decimal place.

3. Date functions

Use of SYSDATE function

The SYSDATE function returns the current database server date and time. You can use this function as follows in SELECT query.

```
SELECT (SYSDATE - HIRE_DATE)/7 "WEEKS EMPLOYED"
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80 ;
```

Note the use of double quotes around "WEEKS EMPLOYED". It is necessary as the aliasing text contains spaces. The SYSDATE is used here to find the number of days an employee has worked, and then this number is divided by 7 to find the number of weeks, the employee passed in the company.

Date arithmetic

You can use arithmetic operators to subtract dates, add some numeric value with dates, etc. In the previous query, you subtracted SYSDATE from HIRE_DATE, and the result is the number of days between these two dates. The following table explains the outcomes of different arithmetic operators that can be applied on DATE type values.

Operation	Outcome

DATE + number	DATE value. Adds the number of days with the given DATE value.
DATE – number	DATE value. Subtracts the number of days with from the given DATE value.
DATE – DATE	Outcome is a numeric value specifying the number of days between these two dates.
DATE + DATE	Invalid operation

Suppose the value of HIRE_DATE column for an employee is '05-FEB-1995'. Then, the following table shows the use date arithmetic operations.

Operation	**Outcome**
HIRE_DATE + 7	'12-FEB-1995' which will be a DATE type value
HIRE_DATE – 4	'01-FEB-1995' which will be a DATE type value
SYSDATE – HIRE_DATE	Number of days between '01-FEB-1995' and today.

Date manipulation functions

The most commonly used date manipulation functions are given below.

Function	**Outcome**
MONTHS_BETWEEN(date1, date2)	Number of months between date1 and date2.
ADD_MONTHS(date1, n)	Adds n months with the date. Result is another date after addition of the months.
ROUND(date, 'MONTH')	Rounds the date to the nearest month. This results date corresponding to either 1st day of the next month or 1st day of the same month.
ROUND(date, 'YEAR')	Rounds the date to the nearest year. This results date corresponding to either 1st day of the next yearor 1st day of the same year.
TRUNC(date, 'MONTH')	Truncates the date to the start of the month.
TRUNC(date, 'YEAR')	Truncates the date to the start of the year.

If we assume SYSDATE = '20-MAR-14' then the following table illustrates some outputs of the date functions.

Expression	Outcome
ADD_MONTHS(SYSDATE, 5)	20-AUG-14
ROUND(SYSDATE, 'MONTH')	01-APR-14
ROUND(date, 'YEAR')	01-MAR-14
TRUNC(date, 'MONTH')	01-JAN-14
TRUNC(date, 'YEAR')	01-JAN-14

The following statements show the user of date manipulation functions.

```
SELECT LAST_NAME, MONTHS_BETWEEN(SYSDATE, HIRE_DATE) MON_EMPLOYED
FROM EMPLOYEES ;
```

Practice 3.3

a. For all employees, find the number of years employed. Print first names and number of years employed for each employee.

b. Suppose you need to find the number of days each employee worked during the first month of his joining. Write an SQL query to find this information for all employees.

4. Functions for manipulating NULL values

NVL function

The NVL function works on a column and outputs a specific value whenever the column value is NULL. The general syntax of this function is given below:

```
NVL(expr1, expr2)
```

If expr1 evaluated to NULL, the NVL function outputs expr2, otherwise, output is expr1.

The following query outputs COMMISSION_PCT value for all employees. Whenever, COMMISSION_PCT is NULL, the output shows a 0 instead of NULL.

```
SELECT LAST_NAME, NVL(COMMISSION_PCT, 0)
FROM EMPLOYEES ;
```

The NVL function can also be used in where to check if a column contains NULL value. For example, the following statement selects all employee records, whose COMMISSION_PCT value is NULL. We assume that, usually COMMISSION_PCT value is not negative.

```
SELECT *
FROM EMPLOYEES
WHERE NVL(COMMISSION_PCT, -1) = -1 ;
```

A better use of the NVL function is shown below where total annual salary is calculated for all employees. Without the NVL function, the expression SALARY*12 + SALARY*12*COMMMISSION_PCT would result in NULL whenever COMMISSION_PCT is NULL.

```
SELECT LAST_NAME,
(SALARY*12 + SALARY*12*NVL(COMMISSION_PCT, 0) ) ANNSAL
FROM EMPLOYEES
WHERE NVL(COMMISSION_PCT, -1) = -1 ;
```

Practice 3.4

a. Print the commission_pct values of all employees whose commission is at least 20%. Use NVL function.

b. Print the total salary of an employee for 5 years and 6 months period. Print all employee last names along with this salary information. Use NVL function assuming that salary may contain NULL values.

5. Data Type Conversion Functions

Oracle Automatic (Implicit) Type Conversion

Oracle can convert automatically the following whenever it requires -

- Convert a NUMBER value to VARCHAR2 value if used in text operations such as concatenation.
- Convert a VARCHAR2 value to NUMBER value in an arithmetic expression provided VARCHAR2 text represents a valid number. Otherwise Oracle would generate an error.
- Convert a DATE value to VARCHAR2 value (in default date format). This conversion is applied in most operations like comparison, concatenation, etc.
- Convert a VARCHAR2 value to a date value provided that VARCHAR2 text is in a default date format. Otherwise, automatic conversion fails. The default date format in Oracle is 'DD-MON-YY' or 'DD-MON-YYYY'.

The following table illustrates this in more detail using examples.

Operation	**Description**
MOD('25', 3)	Works fine. Although, '25' is a text, Oracle automatically convert the text to a number before applying the MOD
ADD_MONTHS('31-JAN-2011', 5)	Works fine. Converts the text '01-JAN-2011' automatically to a DATE value, then adds 5 months. Result will be '30-JUN-2011'
ADD_MONTHS('JAN/31/2011', 5)	Fails! The text is not in default format, so Oracle cannot automatically convert the text to DATE before ADD_MONTHS works.
CONCAT('Today is ', SYSDATE)	Here, converts the DATE value to VARCHAR2 before concatenation.
'Hello' || 123	Works fine. Oracle converts the number 123 to a text of VARCHAR2 before concatenation.
512 + '123'	Works fine. Oracle converts the text '123' to a number.
512 – 'hello'	Fails! Oracle generates an error because it cannot convert the text to a number for arithmetic operation.
SYSDATE – '01-MON-1998'	Converts the text to a DATE value. Then applies DATE arithmetic.

So, whenever, default conversion does not work, we need to use manual conversion functions.

Explicit type conversion: TO_CHAR function

The TO_CHAR function is used to convert DATE value to VARCHAR2 value. The general syntax of the function is given below:

```
TO_CHAR(date, format)
```

The format string specifies how to convert the DATE value to VARCHAR2 value. The following table illustrates the format texts that are most commonly applied to DATE value. Assume that, the value of HIRE_DATE is '31-JAN-1995'.

Expression	**Output (a VARCHAR2 value)**
TO_CHAR(HIRE_DATE, 'DD/MM/YYYY')	'31/01/1995'
TO_CHAR(HIRE_DATE, 'MONTH DD, YYYY')	'JANUARY 31, 1995'
TO_CHAR(HIRE_DATE, 'MONTH DD, YEAR')	'JANUARY 31, NINETEEN HUNDRED AND NINETY FIVE
TO_CHAR(HIRE_DATE, 'Ddspth Month, YEAR')	'Thirty-First January, Nineteen hundred and ninety five'
TO_CHAR(123456)	'123456'

The last example in above table shows that, TO_CHAR function can applied to numeric values also.

Explicit type conversion: TO_NUMBER function

The TO_NUMBER function converts any text of VARCHAR2 to a numeric value. This is very much essential when computing arithmetical operations on two or more columns where all columns were defined of type VARCHAR2. In such cases, we must explicitly convert them to number before operation.

Consider the following query that retrieves all locations in ascending order of POSTAL_CODE.

```
SELECT STREET_ADDRESS, POSTAL_CODE
FROM LOCATIONS
ORDER BY POSTAL_CODE ASC ;
```

However, the above query may not give correct ordering because POSTAL_CODE column is of type VARCHAR2. So, Oracle will do the ordering lexicographically ('1000' will come before '999'). To do numerical order, you can use the following query instead.

```
SELECT STREET_ADDRESS, POSTAL_CODE
FROM LOCATIONS
ORDER BY TO_NUMBER(POSTAL_CODE) ASC ;
```

Explicit type conversion: TO_DATE function

The TO_DATE function converts VARCHAR2 type text strings to DATE type value. The general syntax of the function expression is given below.

```
TO_DATE(text, format)
```

The following table illustrates the use of TO_DATE function with examples.

Expression	**Output (a DATE type value)**
TO_DATE('DEC 31 1995', 'MON DD YYYY')	Works fine.
TO_DATE('31/12/1995', 'DD/MM/YYYY')	Works fine.
TO_DATE('1995-12-31', 'YYY-MM-DD')	Works fine.
TO_DATE('1995-12-31', 'YYY-DD-MM')	Fails! Format does not match with the given text.

You are always encouraged to use explicit date conversions in comparison operations. Otherwise, you query may bring unexpected results. For example, the following query finds all employee last names who was hired before 1st January 1997.

```
SELECT LAST_NAME, TO_CHAR(HIRE_DATE, 'DD-MON-YYYY') HD
FROM EMPLOYEES
WHERE HIRE_DATE < TO_DATE('01-JAN-1997', 'DD-MON-YYYY')
ORDER BY HIRE_DATE ASC ;
```

Practice 3.5

a. Print hire dates of all employees in the following formats:

(i) 13th February, 1998 (ii) 13 February, 1998.

6. More Practice Problems

In class.

Chapter 4: Aggregate Functions

1. Retrieve Aggregate Information: Simple GROUP BY queries

Group functions

In previous chapter, we have studied single-row functions. Those functions operates on one single row and outputs something based on the input columns of the row. The group functions operate on a group of rows and outputs a value such as total, average, etc. These functions report only summary information per group, rather than per row.

The group functions result in only one row per group of rows in the output. The most commonly used group functions are given below.

Function	Description
SUM (Column)	Finds the total, i.e., summation of values in Column for all rows in the group
MAX(Column)	Finds the maximum value in Column for all rows in the group
MIN(Column)	Finds the minimum value in Column for all rows in the group
AVG(Column)	Finds the average value in Column for all rows in the group
COUNT(Column)	Count the number of non-NULL values in Column for all rows in the group

The group functions given above discard NULL values during their computation.

Use of group function in SELECT statement

The group functions are used in the SELECT statement to retrieve information by groups. The groups are identified by GROUP BY clause. The general syntax is given below. These queries are called GROUP BY queries and are very important to database for generating various types of reports.

```
SELECT Column1, Column2, … , SUM(Column1), MAX(Column1), …
FROM table_name
GROUP BY Column1, Column2, …
```

In the above statement, Oracle will first create several groups based on <Column1, Column2,…>. Every unique combination of these columns will denote a group and all rows in the table that have these combinations of values will be in the group. Note that, only the columns in the GROUP BY clause can be selected along with group function expressions.

For example, suppose, you want to create a report showing the total salary paid by the company to each departments. This requires a GROUP BY query and group function as shown below.

```
SELECT DEPARTMENT_ID, SUM(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID ;
```

For another example, suppose, you want to know the maximum salary, minimum salary and average salary the company pays in different job types. The following GROUP BY query will retrieve the required information.

```
SELECT JOB_ID, MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEES
GROUP BY JOB_ID ;
```

Note that, you cannot select a column that is not present in GROUP BY clause. For example, the following query would return an error.

```
SELECT JOB_ID, JOB_TITLE, COUNT(*) TOTAL
FROM JOBS
GROUP BY JOB_ID ;
```

For the third example, suppose you want to know the number of employees working in each department. As you may recall, this requires creating a group based on DEPARTMENT_ID column, then count the number of rows in each group, and then output the information. So, you may think, the following query will output the information.

```
SELECT DEPARTMENT_ID, COUNT(LAST_NAME)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID ;
```

In the above statement, counting the LAST_NAME column is similar to counting any other columns. However, COUNT(LAST_NAME) will ignore the NULL values in LAST_NAME field, therefore not counting rows that contain NULL values in LAST_NAME field. The correct query should be the following.

```
SELECT DEPARTMENT_ID, COUNT(*)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID ;
```

In the above, COUNT(*) counts number of rows (NULL and non-NULL) and therefore will give the correct result.

Omitting GROUP BY clause in group function query

You can omit the GROUP BY clause in group function query. In such cases, the entire table will form a single group, and summary information will be calculated for whole table as one group only.

The following statements find the maximum, minimum, and average salary for the entire table, i.e., all employees of the company. The second statement finds the number of rows in EMPLOYEE table.

```
SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEES ;

SELECT COUNT(*)
FROM EMPLOYEES ;
```

Use of WHERE clause in GROUP BY query

In a GROUP BY statement, the WHERE clause can be used for filtering rows before grouping is applied. The general syntax of such statements is given below.

```
SELECT Column1, Column2, … , SUM(Column1), MAX(Column1), …
FROM table_name
WHERE condition
GROUP BY Column1, Column2, …
```

The following statement finds the maximum and minimum salary for each job types for only the employees working in the department no. 80.

```
SELECT JOB_ID, MAX(SALARY), MIN(SALARY)
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80
GROUP BY JOB_ID ;
```

Use of ORDER BY clause in GROUP BY query

In a GROUP BY statement, ORDER BY clause can be used to sort the final group results based on grouping column. The following example sorts the final results based on JOB_ID value.

```
SELECT JOB_ID, MAX(SALARY), MIN(SALARY)
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80
GROUP BY JOB_ID
ORDER BY JOB_ID ASC ;
```

Practice 4.1

a. For all managers, find the number of employees he/she manages. Print the MANAGER_ID and total number of such employees.

b. For all departments, find the number of employees who get more than 30k salary. Print the DEPARTMENT_ID and total number of such employees.

c. Find the minimum, maximum, and average salary of all departments except DEPARTMENT_ID 80. Print DEPARTMENT_ID, minimum, maximum, and average salary. Sort the results in descending order of average salary first, then maximum salary, then minimum salary. Use column alias to rename column names in output for better display.

2. DISTINCT and HAVING in GROUP BY queries

Use of DISTINCT keyword in Group functions

You can use DISTINCT keyword inside the group functions as given below. In this case, group functions consider only unique rows, and discards duplicate values. So, duplicate value is counted only once for calculation.

```
SELECT JOB_ID, MAX(DISTINCT SALARY), MIN(DISTINCT SALARY),
SUM(DISTINCT SALARY), COUNT(DISTINCT DEPARTMENT_ID)
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80
GROUP BY JOB_ID
ORDER BY JOB_ID ASC ;
```

Note that, the use of DISTINCT does not change the result of MAX and MIN group function But, it does change the outcomes of SUM, and COUNT functions.

Use of HAVING Clause to discard groups

You can use the HAVING clause to remove some group information from the final results. Suppose, you want to retrieve the maximum and minimum salaries of each department except the department no. 80. The following GROUP BY statement will retrieve the required information. The information for group no. 80 are removed using a HAVING clause. HAVING works like WHERE except that HAVING works on groups.

```
SELECT DEPARTMENT_ID, MAX(SALARY), MIN(SALARY)
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID
HAVING DEPARTMENT_ID <> 80 ;
```

Note that, the HAVING condition is applied after group information is calculated unlike WHERE condition is applied before grouping.

The following statements show some more examples of HAVING clause.

```
SELECT JOB_ID, MAX(SALARY), MIN(SALARY)
FROM EMPLOYEES
GROUP BY JOB_ID
HAVING MAX(SALARY) > 5000 ;

SELECT JOB_ID, AVG(SALARY)
FROM EMPLOYEES
GROUP BY JOB_ID
HAVING AVG(SALARY) <= 5000
ORDER BY AVG(SALARY) DESC ;
```

The first statement above outputs maximum and minimum salary for each job types. However, only those results are printed where group maximum is greater than 5000. The second query retrieves average salary of each job type. This time, only those group results are printed for which average salary is less than or equal to 5000. Results are sorted in descending order of average salary.

Practice 4.2

a. Find for each department, the average salary of the department. Print only those DEPARTMENT_ID and average salary whose average salary is at most 50k.

3. Advanced GROUP BY queries

Expression in GROUP BY clause

You can use expressions in group by clause. Expressions help you to retrieve more complex information from the table.

The following query finds the total employees hired in each year.

```
SELECT TO_CHAR(HIRE_DATE, 'YYYY') YEAR, COUNT(*) TOTAL
FROM EMPLOYEES
GROUP BY TO_CHAR(HIRE_DATE, 'YYYY')
ORDER BY YEAR ASC ;
```

The following query finds the total number of employees whose name starts with the same character. The query reports the first character and total employees.

```
SELECT SUBSTR(FIRST_NAME, 1, 1) FC, COUNT(*) TOTAL
FROM EMPLOYEES
GROUP BY SUBSTR(FIRST_NAME, 1, 1)
ORDER BY FC ASC ;
```

More than one column in GROUP BY clause

Sometimes you will need to use multiple columns in group by. For example, the following query finds the number of employees for each department and for each job type. One column would not be sufficient to find this group information.

```
SELECT DEPARTMENT_ID, JOB_ID, COUNT(*) TOTAL
FROM EMPLOYEES
GROUP BY DEPARTMENT_ID, JOB_ID ;
```

An important reason for adding more than one column in a GROUP BY clause is to select more columns. Consider the following query that finds the number of employees for each job type.

```
SELECT JOB_ID, COUNT(*) TOTAL
FROM JOBS
GROUP BY JOB_ID ;
```

Now, if we want to print job titles along with job id in the above query, we has to put the job title column in the GROUP BY clause, because Oracle does not allow to select columns in the SELECT clause that are not present in the GROUP BY clause.

```
SELECT JOB_ID, JOB_TITLE, COUNT(*) TOTAL
FROM JOBS
GROUP BY JOB_ID, JOB_TITLE ;
```

Practice 4.3

a. Find number of employees in each salary group. Salary groups are considered as follows. Group 1: 0k to <5K, 5k to <10k, 10k to <15k, and so on.

b. Find the number of employees that were hired in each year in each job type. Print year, job id, and total employees hired.

4. More Practice Problems

In class.

Chapter 5: Query Multiple Tables – Joins

1. Oracle Joins to Retrieve Data from Multiple Tables

Data from multiple tables

All queries used in previous chapters fetched data from one table only. Sometimes, we need to retrieve data from multiple tables. For example, suppose you want to create a report containing last names of employee, salary, and name of the department employee works in. The first two fields are available in EMPLOYEES table (LAST_NAME and SALARY column). But, the third field is available in DEPARTMENTS table. To know in which department an employee works, we need to join rows from EMPLOYEES table with rows from DEPARTMENT table based on the DEPARTMENT_ID field. This type of joins is done by join queries.

By join operation, rows of two or more tables are joined together to form larger rows. The rows of different tables are joined together based on some columns that may be present in both tables.

Joining two tables by USING clause

Rows from two tables can be joined together by USING clause. The general syntax of this query is given below.

```
SELECT …
FROM table1 JOIN table2 USING (Column1, Column2, …)
WHERE …
GROUP BY …
ORDER BY …
```

The following example statement joins EMPLOYEES and DEPARTENT tables based on the values of DEPARTMENT_ID column. A row from EMPLOYEES table is joined with a row from DEPARTMENTS table where both rows have the same DEPARTMENT_ID value.

```
SELECT E.LAST_NAME, D.DEPARTMENT_NAME
FROM EMPLOYEES E JOIN DEPARTMENTS D USING (DEPARTMENT_ID) ;
```

Note the use of table aliases in the above statement. The table aliases are used in SELECT clause to choose columns from the tables. However, you cannot alias the column, i.e., DEPARTMENT_ID which is used in the USING clause. Otherwise, Oracle will generate error message. The following query clarifies this by not aliasing the DEPARTMENT_ID column in the SELECT clause.

```
SELECT E.LAST_NAME, DEPARTMENT_ID, D.DEPARTMENT_NAME
FROM EMPLOYEES E JOIN DEPARTMENTS D USING (DEPARTMENT_ID) ;
```

Joining two tables by ON clause

Two tables can be joined using the ON clause which is more general than USING. The ON clause specifies an explicit condition on which rows will be joined together. In case of USING, rows are joined based on same values (an equality condition) in join column. But, in case of ON clause, the condition can be specified explicitly allowing equality and non-equality conditions for join.

The following statement shows the same join as before but using ON clause.

```
SELECT E.LAST_NAME,  E.DEPARTMENT_ID,  D.DEPARTMENT_NAME
FROM EMPLOYEES E JOIN DEPARTMENTS D
ON (E.DEPARTMENT_ID = D.DEPARTMENT_ID) ;
```

The more general nature of the ON clause will be clear if you observer the following statements.

Self-join: joining a table with itself using ON clause

A table can be joined with itself using ON clause which is called self-join. This is very useful in many cases. Suppose you want to know the last name of manager for each employee. Each row in the EMPLOYEES table has a MANAGER_ID column which is actually an EMPLOYEE_ID of the same table since the manager is also an employee. To retrieve the last name of the manager for each employee, we need to join two copies of EMPLOYEES table based on MANAGER_ID field.

The following statement shows how the join happens.

```
SELECT E1.LAST_NAME EMPLOYEE, E2.LAST_NAME MANAGER
FROM EMPLOYEES E1 JOIN EMPLOYEES E2
ON (E1.MANAGER_ID = E2.EMPLOYEE_ID) ;
```

Joins using non-quality condition

Tables can be joined by non-equality condition by ON clause. The following statement shows such a join operation. Assume that there is table named JOB_GRADES that has LOWEST_SAL, HIGHEST_SAL, and GRADE_LEVEL columns. We want to retrieve an employee's grade based on his salary. In this case, we need to join EMPLOYEES table with JOB_GRADES table. However, a row from the EMPLOYEES table should join with that row from the JOB_GRADES table whose LOWEST_SAL and HIGHEST_SAL contains the SALARY value of the employee.

```
SELECT E.LAST_NAME, E.SALARY, J.GRADE_LEVEL
FROM EMPLOYEES E JOIN JOB_GRADES J
ON (E.SALARY BETWEEN J.LOWEST_SAL AND J.HIGHEST_SAL)  ;
```

Self-join allows many interesting queries. Suppose you want to create a report showing last name of an employee and a number which is the number of employees getting higher salary than the employee. This type of results can be calculated using joins and groupings. The following statement computes this.

```
SELECT E1.LAST_NAME, COUNT(*) HIGHSAL
FROM EMPLOYEES E1 JOIN EMPLOYEES E2
ON (E1.SALARY < E2.SALARY)
GROUP BY E1.EMPLOYEE_ID, E1.LAST_NAME
ORDER BY E1.LAST_NAME ASC ;
```

Observe that in the above query, the E1.LAST_NAME column was not required to be put in the GROUP BY clause unless it was selected in SELECT clause.

Joining more than two tables

Three or more tables can also be joined using ON clause. The following statement shows such a query that joins three tables.

```
SELECT E.LAST_NAME, D.DEPARTMENT_NAME, L.CITY
FROM EMPLOYEES E
JOIN DEPARTMENTS D
ON (E.DEPARTMENT_ID = D.DEPARTMENT_ID)
JOIN LOCATIONS L
ON (D.LOCATION_ID = L.LOCATION_ID) ;
```

Oracle left outer join and right outer join

The general JOIN operation keeps rows that are found matched in both tables based on join condition. In some cases, we need to keep the rows of first table (or second table or both) in the output that does not meet join criteria. The normal JOIN operation would not keep such rows. In such cases, we need to use LEFT OUTER JOIN and RIGHT OUTER JOIN syntax.

The LEFT OUTER JOIN keeps rows of the left table that did not have any match with rows of right table based on join criteria.

The RIGHT OUTER JOIN keeps rows of the right table that did not have any match with rows of the left table based on join criteria.

The application of such syntax is illustrated in the following query. The query finds the number of employees managed by each employee. If an employee does not mange anyone, then the output displays 0.

```
SELECT E1.LAST_NAME, COUNT(E2.EMPLOYEE_ID) TOTAL
FROM EMPLOYEES E1 LEFT OUTER JOIN EMPLOYEES E2
ON (E1.EMPLOYEE_ID = E2.MANAGER_ID)
GROUP BY E1.EMPLOYEE_ID, E1.LAST_NAME
ORDER BY TOTAL ASC ;
```

If you replace COUNT(E2.EMPLOYEE_ID) by COUNT(*) in the above query, then the query would not output correct results! Find the reason yourself! Morever, E1.LAST_NAME does not any effect on grouping. It was included in the grouping clause so that it can be selected in the output.

Practice 5.1

a. For each employee print last name, salary, and job title.

b. For each department, print department name and country name it is situated in.

c. For each country, finds total number of departments situated in the country.

d. For each employee, finds the number of job switches of the employee.

e. For each department and job types, find the total number of employees working. Print department names, job titles, and total employees working.

f. For each employee, finds the total number of employees those were hired before him/her. Print employee last name and total employees.

g. For each employee, finds the total number of employees those were hired before him/her and those were hired after him/her. Print employee last name, total employees hired before him, and total employees hired after him.

h. Find the employees having salaries greater than at least three other employees

i. For each employee, find his rank, i.e., position with respect to salary. The highest salaried employee should get rank 1 and lowest salaried employee should get the last rank. Employees with same salary should get same rank value. Print employee last names and his/he rank.

j. Finds the names of employees and their salaries for the top three highest salaried employees. The number of employees in your output should be more than three if there are employees with same salary.

Chapter 6: Query Multiple Tables – Sub-query

1. Retrieving Records using Sub-query

What is a sub-query

A sub-query is a SELECT statement which is used inside another SELECT statement. The sub-query can be place in FROM clause, WHERE clause, HAVING clause, etc. The use of sub-query in the WHERE clause makes SELECT statements easier and powerful for retrieving information.

There are special comparison operators when sub-queries are used in WHERE clause. They are ANY, ALL. The general syntax of sub-query in a WHERE clause is given below.

```
SELECT Column1, Column2, …
FROM table_name
WHERE sub-query-with-condition
```

The sub-query executes before the main query and results of sub-query are used in main query.

Use of sub-query in WHERE clause

Suppose you want to know the last names and salaries of all employees whose salary is greater than Abel's salary. The following sub-query solution will retrieve the information efficiently.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY >
(
SELECT SALARY
FROM EMPLOYEES
WHERE LAST_NAME = 'Abel'
) ;
```

Sub-query can result in one row or multiple rows. In the above statement, sub-query is first executed, and SALARY value of Abel is retrieved. Then, this value is used in the main query.

The following is another example where the main query retrieves information of those employees whose JOB_ID is same as the employee numbered 141.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE JOB_ID =
(
SELECT JOB_ID
FROM EMPLOYEES
WHERE EMPLOYEE_ID = 141
) ;
```

Use of multiple sub-queries in single statement

The following statement shows the use of multiple sub-queries in single statement. The statement retrieves records of those employees whose JOB_ID is same as employee numbered 114 and whose SALARY is greater than Abel's SALARY.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE JOB_ID =
(
SELECT JOB_ID
FROM EMPLOYEES
WHERE EMPLOYEE_ID = 141
)
AND SALARY >
(
SELECT SALARY
FROM EMPLOYEES
WHERE EMPLOYEE_ID = 141
)  ;
```

Use of group functions in sub-query

Suppose you want to retrieve the name of the employee who gets highest salary among all employees. The following statement retrieves the information using sub-query.

```
SELECT LAST_NAME, SALARY
FROM EMPLOYEES
WHERE  SALARY =
(
SELECT MAX(SALARY)
FROM EMPLOYEES
)  ;
```

Sub-query returns more than one row

In case sub-query returns more than one row, usual comparison operators must be used with ANY or ALL keyword. Their meanings are

- ANY – If used with comparison operation, the outcome will be true if operator evaluates to true for any of the sub-query values
- ALL – if used with comparison operator, the outcome will be true only if operator evaluates to true for all values returned by the sub-query

To understand the use of ANY and ALL, examine the following statement. The query retrieves those employee records (working in other than 'IT_PROG' department) whose SALARY is less than at least one employee of 'IT_PROG'.

```
SELECT LAST_NAME, JOB_ID, SALARY
FROM EMPLOYEES
WHERE  JOB_ID <> 'IT_PROG'
AND SALARY < ANY
(
SELECT SALARY
FROM EMPLOYEES
WHERE JOB_ID = 'IT_PROG'
)  ;
```

In the query below, ALL is used instead of ANY. This query retrieves those employee records whose SALARY is less than all employees of 'IT_PROG'.

```
SELECT LAST_NAME, JOB_ID, SALARY
FROM EMPLOYEES
WHERE  JOB_ID <> 'IT_PROG'
```

```
AND SALARY < ALL
(
SELECT SALARY
FROM EMPLOYEES
WHERE JOB_ID = 'IT_PROG'
)  ;
```

Practice 6.1

a. Find the last names of all employees that work in the SALES department.

b. Find the last names and salaries of those employees who get higher salary than at least one employee of SALES department.

c. Find the last names and salaries of those employees whose salary is higher than all employees of SALES department.

d. Find the last names and salaries of those employees whose salary is within $\pm$ 5k of the average salary of SALES department.

Chapter 7: Set operations

1. Performing Set Operations

Set operators

The general syntax of set operations is given below.

```
SELECT Column1, Column2, … , ColumnN
FROM table_name
…
SET-OPERATOR
(
SELECT Column1, Column2, … , ColumnN
FROM table_name
…
)
```

The following must be met for the sub-query to be successful-

- Number of columns must be same
- The data type of each column in the second query must match the data type of its corresponding column in the first query.

The following set-operators are available in Oracle.

Set operator	**Description**
UNION	Combines results of two queries eliminating duplicate rows
UNION ALL	Combines results of two queries keeping duplicate values
INTERSECT	Finds the intersection of results of two queries
MINUS	Rows in the first query that are not present in the second query

UNION and UNION ALL operators

The following statement shows the use of UNION operator. The query does not keep duplicate values.

```
SELECT EMPLOYEE_ID, JOB_ID
FROM EMPLOYEES
UNION
(
SELECT EMPLOYEE_ID, JOB_ID
FROM JOB_HISTORY
) ;
```

The UNION ALL operator combines results of two queries and keeps duplicate values.

```
SELECT EMPLOYEE_ID, JOB_ID
FROM EMPLOYEES
UNION ALL
(
SELECT EMPLOYEE_ID, JOB_ID
FROM JOB_HISTORY
) ;
```

INTERSECT operator

The intersect operator performs the set intersection. Rows that are common in both queries are retrieved in the output. The following statement finds the employees whose current job title is same as one of their previous job titles.

```
SELECT EMPLOYEE_ID, JOB_ID
FROM EMPLOYEES
INTERSECT
(
SELECT EMPLOYEE_ID, JOB_ID
FROM JOB_HISTORY
) ;
```

MINUS operator

The MINUS operator performs set minus operation. Rows of first query that are not in the second query will be retrieved for output. The following query finds the employees who have not changed their jobs even once.

```
SELECT EMPLOYEE_ID, JOB_ID
FROM EMPLOYEES
MINUS
(
SELECT EMPLOYEE_ID, JOB_ID
FROM JOB_HISTORY
) ;
```

Use of conversion functions in set operations

In case of set operators, it is mandatory that data types of the columns in both queries must match. In that case, we may need to use type conversion functions to explicitly match data types. For example, assume, there is an EMPLOYEES2 table in which JOB_ID was defined as a numeric value. Now, if you want to find the employee records (LAST_NAME, JOB_ID, SALARY of both tables (EMPLOYEES and EMPLOYEEES2), you may issue the following statement.

```
SELECT LAST_NAME, JOB_ID, SALARY
FROM EMPLOYEES
UNION ALL
(
SELECT LAST_NAME, JOB_ID, SALARY
FROM EMPLOYEES2
) ;
```

But, the above statement would generate an error message as the JOB_ID fields of the two tables have different data types. So, in order to match the data types explicitly, you will require converting the data type of the second query to VARCHAR2 as follows.

```
SELECT LAST_NAME, JOB_ID, SALARY
FROM EMPLOYEES
UNION ALL
```

```
(
SELECT LAST_NAME, TO_CHAR(JOB_ID), SALARY
FROM EMPLOYEES2
) ;
```

Practice 7.1

a. Find EMPLOYEE_ID of those employees who are not managers. Use minus operator to perform this.

b. Find last names of those employees who are not managers. Use minus operator to perform this.

c. Find the LOCATION_ID of those locations having no departments.

2. More Practice Problems

In class.

Chapter 8: Data Manipulation Language (DML)

DML Statements

DML statements are used to:

- Adding new rows to a table – INSERT statements
- Changing data in a table – UPDATE statements
- Removing rows from a table – DELETE statements
- Transactions controls in database – COMMIT, ROLLBACK statements

1. Inserting Data into Table

INSERT statement

The INSERT statement is used to insert new row in a table. INSERT statement allows you to insert one or more rows to the table. The general syntax of the INSERT statement is given below:

```
INSERT INTO table_name (column1, column2, column3, …)
VALUES (value1, value2, value3, ...) ;
```

For example, the following statement inserts a new row in the DEPARTMENTS table.

```
INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID)
VALUES (179, 'Public Relations', 100, 2600) ;
```

Inserting rows without column names unspecified

You can insert values in a table without column names unspecified in the insert statement. However, in this case, you have to ensure the following:

- Values are given for all columns in the table
- Values are given in the default order of the columns as defined in the table creation statements.

For example, the following statement would successfully insert a row in the DEPARTMENTS table.

```
INSERT INTO DEPARTMENTS VALUES (189, 'PRINTING_STATIONARY', 100, 2600) ;
```

However, the following insert statement would not work and will generate an Oracle error message. The reason is that, the DEPARTMENTS table has four columns, but in the statement only values are specified for two columns only.

```
INSERT INTO DEPARTMENTS VALUES (170, 'Public Relations') ;
```

The following insert statement will also fail. Even though, all values are specified, they are not in correct order as defined in the original table definition.

```
INSERT INTO DEPARTMENTS VALUES (170, 100, 'Public Relations', 1700) ;
```

Inserting rows with NULL values for some columns

You can insert rows in a table with NULL values in some columns. In this case, you have to specify the column names of the table explicitly as shown in the following query.

```
INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME)
VALUES (279, 'Purchasing') ;
```

In the above INSERT statement, rows will successfully be inserted. Since, values for MANAGER_ID and LOCATION_ID columns were not specified; they will be filled with NULL. Note that, this will work only if these columns satisfy the following conditions:

- None of these columns is part of a primary key of the table
- None of these columns has NOT NULL constraints

Inserting Date type values

If the column is of DATE type, then you may use the TO_DATE conversion function to convert a text to DATE type value. The following examples show such a conversion to insert a new employee in the EMPLOYEES table.

```
INSERT INTO EMPLOYEES VALUES (279, 'Den', 'Raphealy', 'Drapheal', '515.127.4515',
TO_DATE ( 'FEB 13, 1999', 'MON DD, YYYY'), 'SA_REP', 11000, 0.2, 100, 60) ;
```

However, if you specify the text in default date format, i.e, 'DD-MON-YYYY', then TO_DATE function is not required as shown in the following statement. Oracle will automatically convert the text to DATE type value. **However, it is a good practice to always use the TO_DATE function to explicitly convert to DATE values.**

```
INSERT INTO EMPLOYEES VALUES (279, 'Den', 'Raphealy', 'Drapheal', '515.127.4515',
'13-FEB-1999', 'SA_REP', 11000, 0.2, 100, 60) ;
```

Inserting rows from another table

You can insert several rows from a table directly into another table. The following statement copies all rows from EMPLOYEES table into another table named EMPLOYEES2. Note that, VALUES keyword is not required in the statement.

```
INSERT INTO Employees2
Select * from employees ;
```

You can also use sub-query and where condition to copy some specific rows as shown in the following statement.

```
INSERT INTO SALES_EMPLOYEES (ID, NAME, SALARY, COMMISSION_PCT)
SELECT EMPLOYEE_ID, LAST_NAME, SALARY, COMMISSION_PCT
FROM EMPLOYEES
WHERE JOB_ID LIKE '%REP%' ;
```

Some Common Mistakes of INSERT statements

The oracle server automatically enforces all data types, data ranges, and data integrity constraints. Common errors that may occur during INSERT statements are following:

- Value missing for a column which has NOT NULL constraint
- Duplicate value violating UNIQUE, or PRIMARY KEY constraint
- Any value violating a CHECK constraint
- Data type mismatch or value too large to fit in the column
- Referential integrity violation for FOREIGN KEY constraint.

2. Changing Data in a Table

UPDATE Statement

The UPDATE statement is used to change data in a table. The general syntax of the UPDATE statement is given below. The statement can be used to change data in one or more columns. The WHERE clause can be used to change data for some particular rows and is optional.

```
UPDATE table_name
SET Column1 = value1, Column2 = value2, …
WHERE condition ;
```

The following example changes DEPARTMENT_ID of the employee numbered 113.

```
UPDATE EMPLOYEES
SET DEPARTMENT_ID = 50
WHERE EMPLOYEE_ID = 113 ;
```

To change the DEPARTMENT_ID of all employees, i.e., all rows in the table, omit the WHERE clause as shown below:

```
UPDATE EMPLOYEES
SET DEPARTMENT_ID = 110 ;
```

You can update multiple column values as shown below.

```
UPDATE EMPLOYEES
SET JOB_ID = 'IT_PROG', COMMISSION_PCT = NULL
WHERE EMPLOYEE_ID = 114 ;
```

Use of sub-query in UPDATE statement

You can use a sub-query in the UPDATE statement to fetch date from other tables. For example, the following statement updates job and salary of employee numbered 113 to match those of employee number 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 = 113 ;
```

Practice 8.2

a. Update COMMISSION_PCT value to 0 for those employees who have NULL in that column.

b. Update salary of all employees to the maximum salary of the department in which he/she works.

c. Update COMMISSION_PCT to N times for each employee where N is the number of employees he/she manages. When $N = 0$, keep the old value of COMMISSION_PCT column.

d. Update the hiring dates of all employees to the first day of the same year. Do not change this for those employees who joined on or after year 2000.

3. Deleting Rows from a Table

DELETE Statement

The DELETE statement is used to remove rows from a table. The general syntax of the DELETE statement is given below. The WHERE condition is required to delete only some specified values. If the WHERE condition is omitted, then all rows are deleted from the table.

```
DELTE FROM table_name
WHERE condition ;
```

The following statement removes the Finance department row from the DEPARTMENTS table. *Actually, it will not remove any row due to violation of constraints!*

```
DELETE FROM DEPARTMENTS
WHERE DEPARTMENT_NAME = 'Finance' ;
```

The following statement removes two rows from the DEPARTMENT table. *Actually, it will not remove any row due to violation of constraints!*

```
DELETE FROM DEPARTMENTS
WHERE DEPARTMENT_ID IN (30, 40) ;
```

Note that, the following statement will remove all rows from the DEPARTMENTS table as there is no WHERE condition. *Actually, it will not remove any row due to violation of constraints!*

```
DELETE FROM DEPARTMENTS ;
```

Use of sub-query in the DELETE statement

You can use sub-query to delete rows from a table based on information from another table. The following statement removes all employees from the EMPLOYEES table who are working in the Finance department. *Actually, it will not remove any row due to violation of constraints!*

```
DELETE FROM EMPLOYEES
WHERE DEPARTMENT_ID =
(
SELECT DEPARTMENT_ID FROM DEPARTMENTS
WHERE UPPER(DEPARTMENT_NAME) LIKE '%FINANCE%'
)
```

Note that use of UPPER function in the above query. The UPPER function ensures that, the statement will work even in department names are stored in lower case or upper case letters.

Practice 8.3

a. Delete those employees who earn less than 5k.

b. Delete those locations having no departments.

c. Delete those employees from the EMPLOYEES table who joined before the year 1997.

4. Database Transaction Controls Using COMMIT, ROLLBACK

Database Transactions

A transaction means one or more SQL statements which together make a unit of work. All SQL statements should successfully execute or fail together. In Oracle database, a transaction is automatically started with the first DML statement use executes. The already-started transactions will end whenever one of the following events occurs:

- A COMMIT or ROLBACK statement is issued
- A DDL or DCL statement is issued (automatic COMMIT)
- The user exits SQL*DEVELOPER or SQL*PLUS (automatic COMMIT)! Do not forget this in your entire life!
- The system crashes (automatic ROLLBACK) or SQL*PLUS stopped unexpectedly (automatic ROLLBACK).

After one transaction ends, another transaction will start with the execution of next DML statement.

COMMIT statement

The COMMIT statement saves results of DML operations of the current transaction permanently in database. It also ends the ongoing transaction.

The following statements show the use of COMMIT to store data permanently to database. One row is deleted from the EMPLOYEES table and a row is added to the DEPARTMENTS table. Finally, COMMIT saves this changed permanently into database.

```
DELTE FROM EMPLOYEES
WHERE EMPLOYEE_ID = 99999 ;

INSERT INTO DEPARTMENTS
VALUES (290, 'Corporate Tax', NULL, 1700) ;

COMMIT ;
```

ROLLBACK Statement

The ROLLBACK statement undoes the results of all DML operations executed in the current transaction. It also ends the current transaction. The state of the tables will be restored in the previous values before the current transaction started.

The following statements when executed will not store the new row in the DEPARTMENTS table and will not remove the row from the EMPLOYEES table. The ROLLBACK statement will undo all the changes done by the DELETE and INSERT statements.

```
DELTE FROM EMPLOYEES
WHERE EMPLOYEE_ID = 99999 ;

INSERT INTO DEPARTMENTS
VALUES (290, 'Corporate Tax', NULL, 1700) ;

ROLLBACK ;
```

5. Practice Problems

In class.

Chapter 9: Data Definition Language (DDL) Statements

DDL Statements

Most commonly used DDL statements are for:

- Creating tables in database – CREATE TABLE
- Specifying constraints in tables – NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK
- Deleting tables from database – DROP TABLE

1. Creating Tables

CEATE TABLE statement

The CREATE TABLE statement is used to create new tables in database. The general syntax of the statement is given below:

```
CREATE TABLE [schema_name.]table_name
(
Column1 Datatype1,
Column2 Datatype2,
…
) ;
```

For example, the following example creates a new table in the database:

```
CREATE TABLE PERSON
(
NID VARCHAR2(15),
NAME VARCHAR2(50),
BDATE DATE
) ;
```

The above statement will create a table named PERSON in database. The table PERSON will have three columns which are:

- NID – Data type is VARCHAR2, maximum length 15 characters.

- NAME – Data type if VARCHAR2, maximum length 50 characters
- BDATE – Date type value

Naming rules

Table names and column names have to satisfy the following

- Must begin with a letter
- Must be 1-30 characters long
- Must contain only A-Z, a-z, 0-9, _, $, and #
- Must not duplicate the name of another object of the same user
- Must not be an Oracle keyword

Use the following commands to delete a table entirely from database/

```
DROP TABLE PERSON ;
```

DEFAULT option in CREATE TABLE

You can specify a DEFAULT option in column definition to specify default value for the column as given below:

```
CREATE TABLE PERSON
(
NID VARCHAR2(15),
NAME VARCHAR2(50),
BDATE DATE DEFAULT SYSDATE
) ;
```

The PERSON table created above will have a default value, i.e., SYSDATE for the BDATE column. So, if user does not specify any value for this parameter, then the default value will be used for the column. For example, the following INSERT statement does not specify value for the BDATE column. The value of SYSDATE will be used by default for the BDATE column.

```
INSERT INTO PERSON(NID, NAME) VALUES ('10010010010001', 'Sakib') ;
```

Data Types of Columns

When you are creating tables, you have to specify data type for the columns. The following commonly used data types are available in Oracle:

Data Types	Description
VARCHAR2(size)	Used to store variable length text data. A maximum size must be specified. The minimum size is 1 and maximum size is 4000.
CHAR(size)	Used to store fixed length text data. Minimum size is 1 and maximum size is 2000.
NUMBER(p, s)	Used to store numeric values. You can optionally specify precision (p) and scale (s) parameter values. Precision is the total number of decimal digits and scale is the number to the right of the decimal point.
DATE	Used to store date values.
CLOB	Used to store variable length text data (Up-to 4GB)
BLOB	Used to store binary data (Up-to 4GB)

Creating tables using a sub-query

You can create a table using a sub-query. In this case, table is created automatically with column definitions matched with the results of sub-query. The rows that are returned by the sub-query are also get inserted in the newly created table. The general syntax of this statement is given below:

```
CREATE TABLE table_name [(Column1, Column2, …)]
AS sub-query
```

For example, the following statement creates a table DEPT80 with a sub-query.

```
CREATE TABLE DEPT80
AS
SELECT EMPLOYEE_ID, LAST_NAME, SALARY*12 ANNSAL
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80 ;
```

The DEPT80 table created above –

- Will contain three columns named as EMPLOYEE_ID, LAST_NAME, and ANNSAL.
- The data types of the three columns are inherited from the EMPLOYEES table. So, the data types of the three columns reflect the same as in the EMPLOYEES table.
- The newly created table will inherit all constraints that are defined on the EMPLOYEE_ID, LAST_NAME columns in the EMPLOYEE table.
- The newly created table will contain all data rows of employees whose DEPARTMENT_ID column value is 80.
- Note that, the column alias ANNSAL for the expression SALARY*12 is necessary here. Otherwise, Oracle will generate an error message.

2. Specifying Constraints in Tables

Constraints

Constraints are used in table definitions to specify rules so that invalid data is not entered in database. Usually, constraints do the following:

- Enforce rules on the data. Whenever, a row is inserted, updated, or deleted, these rules are checked against the data. Constraints must be satisfied for the operation to be successful.
- Constraints prevent the deletion of important rows from a table, which may have dependencies in other tables

The commonly used Oracle constraints are following:

Constraint	**Description**
NOT NULL	Specify that the column can not contain NULL values
UNIQUE	The column values in different rows must be unique, however, NULL values are allowed for multiple rows.
PRIMARY KEY	This constraint is used for a column that uniquely identifies each row of a table. Column values must be unique and cannot be NULL. So, this is equivalent to NOT NULL and UNIQUE constraints together.
FOREGIN KEY	Used to create referential integrity checks. Values in a column of

	one table must have similar values in a column of another table.
CHECK	This specifies a condition which must be true for all rows.

Defining constraints

Constraints can be defined in column-level and table-level. The general syntax of column-level constraint definition is as follows:

```
CREATE TABLE [schema_name.]table_name
(
…,
Column datatype [CONSTRAINT constraint_name] constraint_type,
…
) ;
```

The general syntax of table-level constraint definition is given below:

```
CREATE TABLE [schema_name.]table_name
(
Column1 Datatype1,
Column2 Datatype2,
…,
[CONSTRAINT constraint_name] constraint_type (Column1, Column2, …),
…
) ;
```

The following statements show column-level constraints:

```
CREATE TABLE PERSON
(
NID VARCHAR2(15) CONSTRAINT PERSON_PK PRIMARY KEY,
NAME VARCHAR2(50) NOT NULL,
BDATE DATE DEFAULT SYSDATE
) ;
```

The above statement –

- Creates PERSON table with two column-level constraints
- NID column is the PRIMARY KEY of the table. So, NID will uniquely specify each row of the table. Moreover, NID cannot be NULL for any row inserted in the table.
- The PRIMARY KEY constraint of NID column has been given a name, i.e., PERSON_PK
- NAME column cannot have NULL values. This constraint is not given any name.

The following statement creates the same constraints, but this time, constraints are defined in table-level with no difference.

```
DROP TABLE PERSON;

CREATE TABLE PERSON
(
NID VARCHAR2(15),
NAME VARCHAR2(50) NOT NULL UNIQUE,
BDATE DATE DEFAULT SYSDATE,
CONSTRAINT PERSON_PK PRIMARY KEY (NID)
) ;
```

In some cases, two columns together may uniquely identify each row of a table. So, PRIMARY KEY is composed of two column values rather than a single column. This type of constraints cannot be defined using column-level constraints. They must be defined using table-level constraints as shown below:

```
DROP TABLE PERSON;

CREATE TABLE PERSON
(
COUNTRYID CHAR(3),
PERSONID VARCHAR2(15),
NAME VARCHAR2(50) NOT NULL UNIQUE,
BDATE DATE DEFAULT SYSDATE,
CONSTRAINT PERSON_PK PRIMARY KEY (COUNTRYID, PERSONID)
) ;
```

In the above PERSON table, the PRIMARY KEY is composed of two columns, COUNTRYID, and PERSONID.

FOREIGN KEY constraint

A FOREIGN KEY constraint establishes a relationship between columns in one table with another column (which is a PRIMARKY KEY) of another table. Consider the PERSON table created below:

```
DROP TABLE PERSON;

CREATE TABLE PERSON
(
NID VARCHAR2(15) CONSTRAINT PERSON_PK PRIMARY KEY,
NAME VARCHAR2(50) NOT NULL,
BDATE DATE DEFAULT SYSDATE
) ;
```

Now, consider the following ADDRESS table, which stores the address of each person. This person must be a valid row of the PERSON table.

```
CREATE TABLE PERSON_ADDRESS
(
PID VARCHAR2(15) CONSTRAINT PERSON_ADDRESS_PK PRIMARY KEY,
ADDR_LINE1 VARCHAR2(50) NOT NULL,
ADDR_LINE2 VARCHAR2(50),
CITY VARCHAR2(50) NOT NULL,
DISTRICT VARCHAR2(50) NOT NULL,
CONSTRAINT PERSON_ADDRESS_FK FOREIGN KEY(PID) REFERENCES PERSON(NID)
)  ;
```

The PID column in the PERSON_ADDRESS table is defined as the PRIMARY KEY of the table. Moreover, this column is also specified as a FOREIGN KEY constraint which is linked (by REFERENCES Keyword) to NID column of PERSON table. This link will ensure that if a value is inserted in the PID column of PERSON_ADDRESS table, the same value must also be present in the NID column of a row in the PERSON table. Otherwise, the insertion will fail. The PERSON table will be called parent table and PERSON_ADDRESS table will be called a child table.

The FOREIGN KEY constraints create a problem during deletion of rows from the parent table. If a row is to be deleted from the parent table, but the child table have some rows which are dependent (because of

the FOREIGN KEY constraint) on that row, then Oracle will not allow deletion of the row. All dependent rows in the child table must be deleted manually before the deletion of a row in the parent table.

However, there are two solutions available in Oracle. The FOREIGN KEY constraints can have following optional keywords at the end of the constraint definition –

- ON DELETE CASCADE – When a row is deleted from the parent table, the dependent rows are also deleted from the child table automatically by Oracle.
- ON DELETE SET NULL – When a row is deleted from the parent table, the dependent rows are set to NULL values in the child table automatically by Oracle.

CHECK constraints

The CHECK constraints define one or more conditions that must be satisfied by the column values of a row. These constraints can be defined as column-level or table-level. The general syntax of the CHECK constraint is given below:

```
CHECK ( condition )
```

For example, the following table definition contains a CHECK constraint in column-level which ensure that, the SALARY value cannot be negative.

```
DROP TABLE EMPLOYEE2;

CREATE TABLE EMPLOYEE2
(
EID VARCHAR2(15) CONSTRAINT EMPLOYEE2_PK PRIMARY KEY,
SALARY NUMBER CONSTRAINT EMPLOYEE_SAL_MIN CHECK (SALARY > 0)
) ;
```

The above CHECK constraint –

- Is defined in column-level
- Constraint has been given a name, i.e., EMPLOYEE_SAL_MIN
- Condition of the constraint is: SALARY > 0

3. Deleting Tables from Database

DROP TABLE statement

The DROP table statement is used for removing tables from database. The general syntax of the statement is given below:

```
DROP TABLE table_name [PURGE] ;
```

For example, the following statement removes the EMPLOYEES2 table from the database.

```
DROP TABLE EMPLOYEES2 ;
```

Note that, the table structure is deleted and all rows are deleted. But, the space used by the table is not released. To release the storage used by the table, you need to specify PURGE option at the end of the DROP TABLE statements as shown below:

```
DROP TABLE EMPLOYEES2 PURGE ;
```

4. Add or Remove Table Columns

You can add new column to an existing table using the ALTER TABLE SQL command. For example, suppose we want to add a new column BDATE to our EMPLOYEES table whose type will be of date. The following command will add the column.

```
ALTER TABLE EMPLOYEES ADD BDATE DATE ;
```

To delete a column from a table, use the ALTER TABLE DROP command like below.

```
ALTER TABLE EMPLOYEES DROP COLUMN BDATE ;
```

5. More Practice Problems

In class.

Chapter 10: Creating Views, Sequences, and Indexes

Database objects

The database contains different kinds of objects, among which most common are:

- Tables – stores data, created with CREATE TABLE statement.
- Views – stores sub set of data from tables, created with CREATE VIEW statement.
- Sequence – Generate numeric values, created with CREATE SEQUENCE statement
- Index – Improves performance of database, created with CREATE INDEX statement

1. Creating Views

What is a View

A view –

- Is defined by a sub-query, i.e., a SELECT statement
- Is a logical table based on one or more tables or views
- Contains no data of its own, and contain no data at the physical level.
- When user executes query on the view, the view sub-query gets executed and view data are fetched dynamically

CREATE VIEW statement

A view is created by the CREATE VIEW statement. The general syntax of the statement is given below:

```
CREATE [OR REPLACE] VIEW view_name
AS sub-query ;
```

For example, the following statement creates a view named VIEW80.

```
CREATE OR REPLACE VIEW DEPTV80
AS
SELECT EMPLOYEE_ID, LAST_NAME, SALARY*12 ANNSAL
FROM EMPLOYEES
WHERE DEPARTMENT_ID = 80 ;
```

The above created view DEPTV80 –

- Contain three columns EMPLOYEE_ID, LAST_NAME, ANNSAL
- The view does not contain any physical records at the time of creation.
- The sub-query is not executed. It will be executed later when view data are required by another query on the view.
- OR REPLACE clause is optional and if it is specified, then the view is created even though a view with the same name already exists. The old view is deleted.
- The column alias ANNSAL is necessary here, otherwise Oracle will generate an error message.

You can issue SQL SELECT statements on the created view to retrieve data as follows:

```
SELECT * FROM DEPTV80 ;

SELECT EMPLOYEE_ID, ANNSAL FROM DEPTV80 ;
```

The following shows another example of CREATE VIEW statement.

```
CREATE OR REPLACE VIEW DEPT_SUMMARY
AS
SELECT D.DEPARTMENT_NAME DEPT, MIN(E.SALARY) MINSAL, MAX(E.SALARY) MAXSAL,
AVG(E.SALARY) AVGSAL
FROM EMPLOYEES E  JOIN DEPARTMENTS D
ON (E.DEPARTMENT_ID = D.DEPARTMENT_ID)
GROUP BY D.DEPARTMENT_ID, D.DEPARTMENT_NAME ;
```

The above created view –

- Will have four columns, DEPT, MINSAL, MAXSAL, and AVGSAL.
- The column aliases in the sub-query are necessary; otherwise Oracle will generate error messages.
- Since, the view sub-query contains JOIN and GROUP BY clause, DEPT_SUMMARY will be a complex view. In complex view, no DML operations are allowed unlike the simple views where DML operations are allowed.

Advantages of view over tables

There are some advantages of creating views over tables which are –

- Views give an option to restrict data access to a table.
- Views can make complex query easier.
- Views give different view of the same table to different users of the database.
- Views do not contain any record of its own which saves a lot of space of the database than copying a table multiple times.

Removing views from database

You can remove views from the database by DROP VIEW statement. The following example removes the DEPTV80 view from the database. Note that, no data is deleted from database since view does not contain any physical data.

```
DROP VIEW DEPTV80 ;
```

2. Creating Sequences

What is a Sequence

A sequence is a database objects that create numeric (integer) values. The sequence is usually used to create values for a table column, e.g., PID column in the PERSON table defined in previous chapters.

CREATE SEQUENCE statement

To create a sequence CREATE SEQUENCE statement is used. The general syntax of this statement is given below:

```
CREATE SEQUENCE sequence_name
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n]
[MINVALUE n]
[CYCLE | NOCYCLE]
```

Remember the PERSON table created in previous chapter with the following statement.

```
CREATE TABLE PERSON
(
NID VARCHAR2(15) CONSTRAINT PERSON_PK PRIMARY KEY,
NAME VARCHAR2(50) NOT NULL,
BDATE DATE DEFAULT SYSDATE
) ;
```

Now, the following statement creates a sequence named PERSON_NID_SEQ to be used for inserting rows in PERSON table.

```
CREATE SEQUENCE PERSON_NID_SEQ
INCREMENT BY 1
START WITH 100000000000001
MAXVALUE 999999999999999
NOCYCLE ;
```

The above created sequence –

- Will generate numeric values starting with 100000000000001
- The next value is found by adding 1 with previous value
- The maximum value will be 999999999999999
- The NOCYLE option ensures that after the maximum value is generated, the sequence will stop generating value. If you want the sequence to generate values again from the starting value, use the CYCLE option instead of NOCYCLE option.

Using the sequence

The created sequence PERSON_NID_SEQ will be used for inserting data in the PERSON table. The sequence will provide sequential numbers for the NID column of the PERSON table. The NEXTVAL is used to retrieve next sequential number from the sequence as shown by the following INSERT statements:

```
INSERT INTO PERSON
VALUES (PERSON_NID_SEQ.NEXTVAL, 'Ahsan', TO_DATE('12-DEC-1980','DD-MON-YYYY') ) ;
```

You can view the current value of the sequence by the following query:

```
SELECT PERSON_NID_SEQ.CURRVAL
FROM DUAL ;
```

Note that use of DUAL table here.

3. Creating Indexes

What is an Index

An index –

- Is created on a column of a database table
- Provides fast access to the table data using the column on which index is defined
- Is used by the Oracle server to retrieve query results efficiently.
- Reduces disk input outputs for retrieving query results.
- Are automatically managed by the Oracle server after user creates the index.

CREATE INDEX statement

An index is created automatically on those columns of the table which have PRIMARY KEY or UNIQUE constraints. To create index on other columns, CREATE INDEX statement is used. The general syntax of this statement is given below:

```
CREATE INDEX index_name
ON table_name (Column1, Column2, … ) ;
```

For example, the following statement creates an index on the NAME column of the PERSON table defined above. This will enable faster access of data based on NAME column. That means, if the NAME column is used in WHERE clause of a query, then Oracle can retrieve query results more quickly than if no index was present.

```
CREATE INDEX PERSON_NAME_IDX
ON PERSON (NAME) ;
```

You can drop an index using DROP INDEX statement as shown below:

```
DROP INDEX PERSON_NAME_IDX ;
```

When to create indexes?

In general, you should create an index on a column only when one or more of the following conditions are satisfied –

- The column contains wide range of values
- The column contains large number of NULL values
- The column is frequently used in WHERE clause condition
- The table is large and most queries are expected to retrieve only less than 2% to 4% rows.

4. More Practice Problems

In class.

Chapter 11: Introduction to PL/SQL

1. Anonymous PL/SQL blocks

General structure of Anonymous blocks

The general structure of PL/SQL block is given below.

```
DECLARE
      <declarative section>
      --variables are declared here
BEGIN
      <executable section>
      --processing statements are written here
EXCEPTION
      <exception code>
      --exception handling codes are written here if any
END ;
```

The different sections of PL/SQL block are summarized below:

- Declarative (after DECLARE) – Contains declaration of variables, constants, etc. This is an optional section.
- Executable (after BEGIN) – Contains SELECT statements to retrieve data from tables and contain PL/SQL statements to manipulate those data. This is a mandatory section.
- Exception (after EXCEPTION) – This is an optional section. Contains exception handling codes.

The following statement shows an anonymous block that outputs 'Hello world'.

```
BEGIN
      DBMS_OUTPUT.PUT_LINE('Hello World') ;
END ;
/
```

Note the following:

- Every statement in a block ends with a semi-colon (;)
- DBMS_OUT.PUT_LINE ( ) is a function that is used to print messages from a PL/SQL block.

Execute an anonymous block

To execute the anonymous block written above, you need to copy paste the entire code into SQL*PLUS. Do not forget to copy the forward slash (/) also. At first, you may not see any output. To view PL/SQL outputs in the SQL*PLUS, you need to issue following command before executing the script above.

```
SET SERVEROUTPUT ON
```

Use of variables in PL/SQL

For programming, you will need to declare variables to store temporary values. The following script declares a variable named ENAME. The SQL query in the executive section retrieves full name of the employee numbered 100. Then, output the result.

```
DECLARE
      ENAME VARCHAR2(100) ;
BEGIN
      SELECT (FIRST_NAME || LAST_NAME) INTO ENAME
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = 100 ;
      DBMS_OUTPUT.PUT_LINE('The name is : ' || ENAME) ;
END ;
/
```

Note that use INTO keyword to store value from SELECT statement into PL/SQL variable (ENAME).

Use of SQL functions in PL/SQL

You can use SQL functions in PL/SQL statements. The following statement outputs the number of months employee 100 worked in the company.

```
DECLARE
      JDATE DATE ;
      MONTHS NUMBER ;
      RMONTHS NUMBER ;
BEGIN
      SELECT HIRE_DATE INTO JDATE
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = 100 ;
```

```
      MONTHS  :=  MONTHS_BETWEEN(SYSDATE, JDATE) ;
      RMONTHS  :=  ROUND(MONTHS, 0) ;
      DBMS_OUTPUT.PUT_LINE('The employee worked ' || RMONTHS || ' months.') ;
END ;
/
```

Note that, the assignment operator in PL/SQL is colon-equal (:=). The equal (=) operator is the equality comparison operator and cannot be used as an assignment operator. *Do not give space between the colon (:) and equal (=) of the colon-equal (:=) operator, otherwise it will generate an error.*

View errors of a PL/SQL block

In case, your PL/SQL block contains a syntactic error, Oracle will generate and show an error-code rather than showing the errors. To view the errors along with their line numbers, you need to run the following command:

```
SHOW ERRORS ;
```

Oracle will not compile the erroneous blocks. You must correct the errors and re-run the corrected code.

Use of IF-ELSE conditional statement

The IF-ELSE statement processes conditional statements. The general structure of IF-ELSE statement is given below.

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

The following example shows the use of IF-ELSE in PL/SQL. The script finds whether employee numbered 100 worked for more than 10 years in the company.

```
DECLARE
      JDATE DATE ;
```

```
      YEARS NUMBER ;
BEGIN
      SELECT HIRE_DATE INTO JDATE
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = 100 ;
      YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
      IF YEARS >= 10 THEN
            DBMS_OUTPUT.PUT_LINE('The employee worked 10 years or more') ;
      ELSE
            DBMS_OUTPUT.PUT_LINE('The employee worked less than 10 years') ;
      END IF ;
END ;
/
```

The following example shows another use of IF-ELSE statement to find the grade level of employee numbered 100.

```
DECLARE
      ESALARY NUMBER ;
BEGIN
      SELECT SALARY INTO ESALARY
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = 100 ;
      IF ESALARY < 1000 THEN
            DBMS_OUTPUT.PUT_LINE('Job grade is D') ;
      ELSIF ESALARY >= 1000 AND ESALARY < 2000 THEN
            DBMS_OUTPUT.PUT_LINE('Job grade is C') ;
      ELSIF ESALARY >= 2000 AND ESALARY < 3000 THEN
            DBMS_OUTPUT.PUT_LINE('Job grade is B') ;
      ELSIF ESALARY >= 3000 AND ESALARY < 5000 THEN
            DBMS_OUTPUT.PUT_LINE('Job grade is A') ;
      ELSE
            DBMS_OUTPUT.PUT_LINE('Job grade is A+') ;
      END IF ;
END ;
/
```

Do not forget to use THEN after each ELSIF clause. This is a common mistake everyone does.

Comments in PL/SQL blocks

You can put single-line comments and multi-line comments in a PL/SQL block. The syntax of both comments is given below.

```
--This is a single-line comment

/*
This is a multi-line
Comment.
*/
```

2. Exception Handling in PL/SQL block

Handling exceptions

An Oracle statement can throw exceptions. For example, in our previous two blocks, the first SELECT statement can throw exception if not data is found in database for the given employee id.

When an Oracle statement throws an exception, the PL/SQL block immediately *exits* discarding later statements in the block. To examine this behavior, just run the above two blocks by modifying the employee id from 100 to 10000. Since, there is no employee numbered 10000 in the EMPLOYEES table, the SELECT statement would throw an exception and PL/SQL block would exit.

Sometimes, we need to show meaningful messages to the user when such an exception occurs. In this case, we will need somehow to handle the exception ourselves and generate those messages. In Oracle PL/SQL, we can handle an exception by writing an EXCEPTION section before the END statement of the block. The following example shows how to do this.

```
DECLARE
      JDATE DATE ;
      YEARS NUMBER ;
BEGIN
      --first retrieve hire_date and store the value into JDATE variable
      SELECT HIRE_DATE INTO JDATE
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = 10000 ;
      --calculate years from the hire_date field
      YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
```

```
      IF YEARS >= 10 THEN
            DBMS_OUTPUT.PUT_LINE('The employee worked 10 years or more') ;
      ELSE
            DBMS_OUTPUT.PUT_LINE('The employee worked less than 10 years') ;
      END IF ;
EXCEPTION
       --handle exceptions one by one here
       WHEN NO_DATA_FOUND THEN
             DBMS_OUTPUT.PUT_LINE('Employee is not present in database.') ;
       WHEN OTHERS THEN
             DBMS_OUTPUT.PUT_LINE('I dont know what happened!') ;
END ;
/
```

The general syntax of handling exception codes is following:

```
DECLARE
       ...
BEGIN
       ...
EXCEPTION
       WHEN <EXCEPTION_NAME1> THEN
             Statement1 ;
       WHEN <EXCEPTION_NAME2> THEN
             Statement2 ;
       ...
       WHEN OTHERS THEN
             Default Statement ;
END ;
/
```

Note the following:

- <EXCEPTION_NAME1>, <EXCEPTION_NAME2> can be Oracle pre-defined exception names or user defined names.
- The default exception name OTHERS is optional.
- In our example above, we used two pre-defined exception names: NO_DATA_FOUND and OTHERS. There are many pre-defined exception names in Oracle. You must have the knowledge

of these exception names as Oracle throws such exceptions for several types of errors while processing a statement.

Oracle pre-defined exception names

The following table shows some Oracle pre-defined exception names.

Name of Exception	When Oracle Throws It?
NO_DATA_FOUND	When a SELECT INTO statement cannot retrieve a row.
TOO_MANY_ROWS	When a SELECT INTO statement retrieves more than one row.
DUP_VAL_ON_INDEX	When duplicate values are attempted to be stored in a table column with unique index.
NVALID_NUMBER	When the conversion of a character string into a number fails because the string does not represent a valid number.
VALUE_ERROR	When an arithmetic, conversion, truncation, or size-constraint error occurs.
ZERO_DIVIDE	When an attempt is made to divide a number by zero.

Practice 11.2

a. Extend the above block by handing the following exception: TOO_MANY_ROWS. Print an appropriate message when such an exception occurs.

b. Write an example PL/SQL block that inserts a new arbitrary row to the COUNTRIES table. The block should handle the exception DUP_VAL_ON_INDEX and OTHERS. Run the block for different COUNTRY_ID and observe the cases when above exception occurs.

3. Loops in PL/SQL block

There are several types of loop statements in PL/SQL such as FOR loops, WHITE loops, and unconditional loops. These are described below.

Use of loops in PL/SQL

The general structure of writing loops using FOR is given below.

```
FOR variable in lowest val..highest val
LOOP
      Statement1 ;
```

```
      Statement2 ;
      …
END LOOP ;
```

The following for loop prints the numbers from 1 to 100.

```
DECLARE
BEGIN
       FOR i in 1..100
       LOOP
              DBMS_OUTPUT.PUT_LINE(i);
       END LOOP ;
End ;
/
```

The general structure of WHILE loop is given below.

```
WHILE Condition
LOOP
      Statement1 ;
      Statement2 ;
      …
END LOOP ;
```

The following example prints the numbers from 1 to 100 using WHILE loop.

```
DECLARE
       i number;
BEGIN
       i := 1;
       WHILE i<=100
       LOOP
              DBMS_OUTPUT.PUT_LINE(i);
              i := i + 1;
       END LOOP ;
End ;
/
```

There is a type of loop called unconditional-loop in Oracle PL/SQL. The following example shows how to write the unconditional loop to print the numbers from 1 to 100.

```
DECLARE
       i number;
BEGIN
       --this is an unconditional loop, must have EXIT WHEN inside loop
       i := 1;
       LOOP
              DBMS_OUTPUT.PUT_LINE(i);
              i := i + 1;
              EXIT WHEN  (i > 100) ;
       END LOOP ;
End ;
/
```

Note the statement EXIT WHEN to exit the unconditional loop. This is a must in an unconditional loop otherwise the loop will run for-over. Also note that EXIT WHEN statement can also be used inside a WHILE loop or FOR loop to exit at any time.

Use of Cursor FOR loops in PL/SQL

The cursor FOR loop is special FOR loop in Oracle PL/SQL. The loop variable of this type of loop iterates over the rows of a SQL query. So, it is widely used when loop is to be done over the rows of a SELECT statement.

The general structure of writing a loop (a FOR loop) in PL/SQL is given below.

```
FOR variable in (SELECT statement)
LOOP
      Statement1 ;
      Statement2 ;
      …
END LOOP ;
```

The following example augments our previous code of finding employees working for more than 10 years. This time, the script counts the number of employees who worked 10 years or more in the company. Then, it displays the count.

```
DECLARE
```

```
      YEARS NUMBER ;
      COUNTER NUMBER ;
BEGIN
      COUNTER := 0 ;
      --the following for loop will iterate over all rows of the SELECT results
      FOR R IN (SELECT HIRE_DATE FROM EMPLOYEES )
      LOOP
            --variable R is used to retrieve columns
            YEARS := (MONTHS_BETWEEN(SYSDATE, R.HIRE_DATE) / 12) ;
            IF YEARS >= 10 THEN
                  COUNTER := COUNTER + 1 ;
            END IF ;
      END  LOOP ;
      DBMS_OUTPUT.PUT_LINE('Number of employees worked 10 years or more: ' ||
COUNTER) ;
END ;
/
```

In the above script, a special variable R is used which is called a *cursor-variable*. Details of cursor are out of scope of this book. In summary –

- A cursor variable fetches rows of a SELECT statement one be one
- The FOR LOOP ends automatically after the last row have been fetched by the cursor variable R
- The columns of a row are accessed by *variable_name.column_name* format (R.HIRE_DATE)
- Only those columns are available that are retrieved in the SELECT statement (R.LAST_NAME is invalid as LAST_NAME is not retrieved in SELECT)

Updating table from PL/SQL block

The following PL/SQL block increases salary of each employee X by 15% who have worked in the company for 10 years or more.

```
DECLARE
       YEARS NUMBER ;
       COUNTER NUMBER ;
       OLD_SAL NUMBER;
       NEW_SAL NUMBER;
BEGIN
```

```
       COUNTER := 0 ;
       FOR R IN (SELECT EMPLOYEE_ID, SALARY, HIRE_DATE FROM EMPLOYEES )
       LOOP
              OLD_SAL := R.SALARY ;
              YEARS := (MONTHS_BETWEEN(SYSDATE, R.HIRE_DATE) / 12) ;
              IF YEARS >= 10 THEN
                     UPDATE EMPLOYEES SET SALARY = SALARY * 1.15
                     WHERE EMPLOYEE_ID = R.EMPLOYEE_ID ;
              END IF ;
              SELECT SALARY INTO NEW_SAL FROM EMPLOYEES
              WHERE EMPLOYEE_ID = R.EMPLOYEE_ID ;
              DBMS_OUTPUT.PUT_LINE('Employee id:' || R.EMPLOYEE_ID || ' Salary: '
|| OLD_SAL || ' -> ' || NEW_SAL) ;
       END  LOOP ;
       COMMIT;
END ;
/
```

Practice 11.1

a. Write a PL/SQL block that will print 'Happy Anniversary X' for each employee X whose hiring date is today. Use cursor FOR loop for the task.

4. PL/SQL Procedures

PL/SQL procedures and functions (to be discussed in later section) are like PL/SQL blocks except the following:

- Unlike PL/SQL (anonymous) blocks, a PL/SQL procedure or function has a name and zero or more parameters. The parameters are used to give inputs to the procedure or function. The function can even return values.
- A PL/SQL (anonymous) block is to be saved in a separate file by the user, if he wishes to run it later. However, a PL/SQL procedure or function can be saved in the database, so it need not be saved in a separate file by the user.
- When a procedure or function code is run in the SQL command line, the procedure or function codes are not executed. The codes are just stored in the database. To execute the procedure, we

will need to call it from the command line using EXEC PROC_NAME statement. To execute the function, we need to call it in a select statement or from inside a PL/SQL block.

Wring a procedure

The general syntax of writing a PL/SQL procedure is given below.

```
CREATE OR REPLACE PROCEDURE <PROC_NAME> ( <PARAM1> [IN/OUT] <DATATYPE1>, ...) IS
      Declaration1 ;
      Declaration2 ;
      ... ;
BEGIN
      Statement1 ;
      Statement2 ;
      ... ;
END ;
/
```

Note the following regarding procedure parameters:

- Parameters are optional so a procedure can have zero parameters.
- Parameters can be of two types: IN and OUT. IN parameters receive input and OUT parameters generate output (like returning values from procedure).

Let us re-write our PL/SQL block to find whether employee numbered 100 has worked10 years or more. Let us call this procedure IS_SENIOR_EMPLOYEE. The following code shows the procedure.

```
CREATE OR REPLACE PROCEDURE IS_SENIOR_EMPLOYEE IS
      JDATE DATE ;
      YEARS NUMBER ;
BEGIN
      SELECT HIRE_DATE INTO JDATE
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = 100 ;
      YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
      IF YEARS >= 10 THEN
            DBMS_OUTPUT.PUT_LINE('The employee worked 10 years or more') ;
      ELSE
            DBMS_OUTPUT.PUT_LINE('The employee worked less than 10 years') ;
```

```
      END IF ;
END ;
/
```

If you run the above code, then the procedure is saved in the database. Later you can execute the procedure form the SQL*PLUS command line as follows.

```
EXEC  IS_SENIOR_EMPLOYEE ;
```

You can also execute the procedure inside a PL/SQL block, inside another procedure or function as shown below.

```
DECLARE
BEGIN
       IS_SENIOR_EMPLOYEE ;
END ;
```

Use of procedure parameters

Procedure parameters can be utilized to send inputs to the procedure. Let us re-write our previous procedure IS_SENIOR_EMPLOYEE so that the employee id is not kept fixed in the code rather we can send the employee id as a parameter to the procedure during run-time. The modified version of the procedure is given below.

```
CREATE OR REPLACE PROCEDURE IS_SENIOR_EMPLOYEE(EID IN VARCHAR2) IS
      JDATE DATE ;
      YEARS NUMBER ;
BEGIN
      SELECT HIRE_DATE INTO JDATE
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = EID ;
      YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
      IF YEARS >= 10 THEN
            DBMS_OUTPUT.PUT_LINE('The employee worked 10 years or more') ;
      ELSE
            DBMS_OUTPUT.PUT_LINE('The employee worked less than 10 years') ;
      END IF ;
```

```
END ;
/
```

The greatest advantage of this modified parameterized version is that we can now run the same procedure for several employees as shown below.

```
DECLARE
BEGIN
      IS_SENIOR_EMPLOYEE(100) ;
      IS_SENIOR_EMPLOYEE(105) ;
END ;
```

Handling exception in procedure

We can handle exceptions in a PL/SQL procedure like we did in PL/SQL (anonymous) block. In all practical tasks, handling exception is a must.

Although, we did not handle any exception in the above blocks, a good programmer should handle all required exceptions and show meaning messages to the user. The following code illustrates this.

```
CREATE OR REPLACE PROCEDURE IS_SENIOR_EMPLOYEE(EID IN VARCHAR2) IS
      JDATE DATE ;
      YEARS NUMBER ;
BEGIN
      SELECT HIRE_DATE INTO JDATE
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = EID ;
      YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
      IF YEARS >= 10 THEN
            DBMS_OUTPUT.PUT_LINE('The employee worked 10 years or more') ;
      ELSE
            DBMS_OUTPUT.PUT_LINE('The employee worked less than 10 years') ;
      END IF ;
EXCEPTION
      WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('No employee found.') ;
      WHEN TOO_MANY_ROWS THEN
            DBMS_OUTPUT.PUT_LINE('More than one employee found.') ;
```

```
      WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Some unknown error occurred.') ;
END ;
/
```

After handling exceptions as above, run the following block to understand the effect.

```
DECLARE
BEGIN
      IS_SENIOR_EMPLOYEE(10000) ;
      IS_SENIOR_EMPLOYEE(105) ;
END ;
```

Generate output from a procedure

Let us re-write above procedure so that it stores the resulting message in an output variable rather than printing the message itself. The following code shows how to do this.

```
CREATE OR REPLACE PROCEDURE IS_SENIOR_EMPLOYEE(EID IN VARCHAR2, MSG OUT VARCHAR2)
IS
      JDATE DATE ;
      YEARS NUMBER ;
BEGIN
      SELECT HIRE_DATE INTO JDATE
      FROM EMPLOYEES
      WHERE EMPLOYEE_ID = EID ;
      YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
      IF YEARS >= 10 THEN
            MSG := 'The employee worked 10 years or more' ;
      ELSE
            MSG := 'The employee worked less than 10 years' ;
      END IF ;
EXCEPTION
      WHEN NO_DATA_FOUND THEN
            MSG := 'No employee found.' ;
      WHEN TOO_MANY_ROWS THEN
            MSG := 'More than one employee found.' ;
      WHEN OTHERS THEN
            MSG := 'Some unknown error occurred.' ;
```

```
END ;
/
```

The following code now tests the re-written procedure. It uses a new variable MESSAGE to retrieve the output of the procedure. Then it prints the MESSAGE.

```
DECLARE
      MESSAGE VARCHAR2(100) ;
BEGIN
      IS_SENIOR_EMPLOYEE(10000, MESSAGE) ;
      DBMS_OUTPUT.PUT_LINE(MESSAGE) ;
      IS_SENIOR_EMPLOYEE(105, MESSAGE) ;
      DBMS_OUTPUT.PUT_LINE(MESSAGE) ;
END ;
```

5. PL/SQL Functions

A PL/SQL function is like a PL/SQL procedure except that it must return a value. The general syntax of a PL/SQL function is slightly different than a PL/SQL procedure. It only has an RETURN <DATATYPE> clause at the end of the CREATE OR REPLACE statement.

```
CREATE OR REPLACE FUNCTION <FUNC_NAME> ( <PARAM1> [IN/OUT] <DATATYPE1>, ...)
RETURN <DATATYPE> IS
      Declaration1 ;
      Declaration2 ;
      ... ;
BEGIN
      Statement1 ;
      Statement2 ;
      ... ;
END ;
/
```

The effect of the RETURN statement is that a value is return after successful execution of the function. That value will be used in the calling section.

Let us re-write the same procedure used before as a function. In this task, the MSG variable used in the procedure to output the message will be done by return statement. The following code shows the function how to do this.

```
CREATE OR REPLACE FUNCTION GET_SENIOR_EMPLOYEE(EID IN VARCHAR2)
RETURN VARCHAR2 IS
      JDATE DATE ;
      YEARS NUMBER ;
       MSG VARCHAR2(100) ;
BEGIN
       SELECT HIRE_DATE INTO JDATE
       FROM EMPLOYEES
       WHERE EMPLOYEE_ID = EID ;
       YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
       IF YEARS >= 10 THEN
            MSG := 'The employee worked 10 years or more' ;
       ELSE
            MSG := 'The employee worked less than 10 years' ;
       END IF ;
       RETURN MSG ; --return the message
EXCEPTION
       --you must return value from this section also
       WHEN NO_DATA_FOUND THEN
              RETURN 'No employee found.' ;
       WHEN TOO_MANY_ROWS THEN
              RETURN 'More than one employee found.' ;
       WHEN OTHERS THEN
              RETURN 'Some unknown error occurred.' ;
END ;
/
```

Note the following:

- A function must return a value of desired value from the BEGIN section
- A function must also return a value of desired type from the EXCEPTION section.

The following code now tests our function.

```
DECLARE
       MESSAGE VARCHAR2(100) ;
```

```
BEGIN
	MESSAGE := GET_SENIOR_EMPLOYEE(10000) ;
	DBMS_OUTPUT.PUT_LINE(MESSAGE) ;
	MESSAGE := GET_SENIOR_EMPLOYEE(105) ;
	DBMS_OUTPUT.PUT_LINE(MESSAGE) ;
END ;
```

Why function when procedure can output value?

We have seen that a procedure can output values. So why should one write a function? The most important application of writing a function is that it can be used in a SELECT query unlike a procedure which can't be used. The beautiful effect of a function can then be best understood like other SQL functions we have seen and used before in various queries.

Let us write the following SQL query and see the output.

```
SELECT LAST_NAME, GET_SENIOR_EMPLOYEE(EMPLOYEE_ID)
FROM EMPLOYEES ;
```

Nested PL/SQL blocks

PL/SQL blocks can be nested. This means we can write a PL/SQL block inside a PL/SQL block. We will go into too much details of nesting. We will just illustrate the nesting by re-writing the function we created above.

Let us re-write the above function as follows. In this case, before retrieving the HIRE_DATE of the employee, we will first check whether such an employee occurs in database. After it is found that such an employee occurs, we will retrieve its HIRE_DATE and continue later processing. The modified function is shown below.

```
CREATE OR REPLACE FUNCTION GET_SENIOR_EMPLOYEE(EID IN VARCHAR2)
RETURN VARCHAR2 IS
	ECOUNT NUMBER;
	JDATE DATE ;
	YEARS NUMBER ;
	MSG VARCHAR2(100) ;
BEGIN
	--Inner PL/SQL block
```

```
      BEGIN
            SELECT COUNT(*) INTO ECOUNT
            FROM EMPLOYEES
            WHERE EMPLOYEE_ID = EID ;
      END ;
      IF ECOUNT = 0 THEN
            MSG := 'No employee found.' ;
      ELSIF ECOUNT > 1 THEN
            MSG := 'More than one employee found.' ;
      ELSE
            SELECT HIRE_DATE INTO JDATE
            FROM EMPLOYEES
            WHERE EMPLOYEE_ID = EID ;
            YEARS :=  (MONTHS_BETWEEN(SYSDATE, JDATE) / 12) ;
            IF YEARS >= 10 THEN
                  MSG := 'The employee worked 10 years or more' ;
            ELSE
                  MSG := 'The employee worked less than 10 years' ;
            END IF ;
      END IF ;
      RETURN MSG ;
END ;
/
```

Note the following:

- Inner PL/SQL blocks are like a general block that can have EXCEPTION section as well if required.
- In the modified function, exception handling is not required. Because this time number of employees is first checked before retrieving employee's hiring date. If no such data is found, then the first SELECT COUNT(*) INTO ECOUNT query retrieves 0 into ECOUNT variable. So, no exception occurs and therefore exception handling is also not required.

Practice

- In Oracle, there is a function TO_NUMBER that converts a VARCHAR2 value to a numeric value. If the input to this function is not a valid number, then this function throws an exception. This is a problem in a SQL query because the whole query would not produce any result if one row generates an exception. So, your job is to write a PL/SQL function ISNUMBER that receives

an input VARCHAR2 value and checks whether the input can be converted to a valid number. If the input can be converted to a valid number than ISNUMBER should return 'YES', otherwise ISNUMBER should return 'NO'.

6. PI/SQL Triggers

A trigger is a PL/SQL stored block. It is like a function or a procedure. However, it is different than a function or a procedure. To run a function or a procedure, it is explicitly called from a code. However, a trigger is automatically run by the Oracle. It is not called from any code directly.

A trigger is automatically run by Oracle -

- After/Before a database insertion operation
- After/Before a database deletion operation
- After/Before a database update operation
- After DDL operations, etc.

Trigger is very useful when certain tasks are needed to be done after or before a DML operation. For example, suppose we need to ensure that after every deletion of an employee records from the EMPLOYEES table, the records of the deleted employee should go to a backup table. In such cases, a trigger can be written which will perform the required task easily. Since, trigger will be automatically called after the deletion operation; we need not to manually handle the writing backup records.

The general syntax of a trigger is like a procedure or a function except the header line.

```
CREATE OR REPLACE TRIGGER <TRIGGER_NAME>
(BEFORE | AFTER) (INSERT | UPDATE | DELETE)
[OF <COLUMN_NAME>]
ON <TABLE_NAME>
[FOR EACH ROW]
[WHEN <CONDITION>]
DECLARE
      Declaration1 ;
      Declaration2 ;
      ... ;
BEGIN
      Statement1 ;
      Statement2 ;
      ... ;
```

```
EXCEPTION
       Exception handing codes ;
END ;
/
```

Let's first create the following table STUDENTS.

```
CREATE TABLE STUDENTS(
STUDENT_NAME VARCHAR2(250),
CGPA NUMBER
) ;
```

Let's understand the trigger by writing our first trigger HELLO_WORLD trigger. This trigger will print "Hello World" whenever an insertion statement is run on the table STUDENTS.

The following shows the trigger codes.

```
CREATE OR REPLACE TRIGGER HELLO_WORLD
AFTER INSERT
ON STUDENTS
DECLARE
BEGIN
       DBMS_OUTPUT.PUT_LINE('Hello World');
END ;
/
```

Now, insert some rows (as shown below) and see what output comes!

```
INSERT INTO STUDENTS VALUES ('Fahim Hasan', 3.71);
INSERT INTO STUDENTS VALUES ('Ahmed Nahiyan', 3.80);
```

The trigger will run twice for the two insert statements and it will ouput "Hello World".

Although, HELLO_WORLD trigger does not show the actual power of triggers, it demonstrates at least the basic properties of a trigger which are following:

- HELLO_WORLD trigger will run automatically by Oracle when an insert statement is run on the table STUDENTS.
- It will run after the execution of insert statement.

Let's try several modification of the above trigger so that it runs in various settings. The following codes show five variations of the HELLO_WORLD trigger.

```
--This trigger will run before insert on STUDENTS table

CREATE OR REPLACE TRIGGER HELLO_WORLD2
BEFORE INSERT
ON STUDENTS
DECLARE
BEGIN
	DBMS_OUTPUT.PUT_LINE('Hello World2');
END ;
/

--This trigger will run after an insert or a delete statement on STUDENTS table

CREATE OR REPLACE TRIGGER HELLO_WORLD3
AFTER INSERT OR DELETE
ON STUDENTS
DECLARE
BEGIN
	DBMS_OUTPUT.PUT_LINE('Hello World3');
END ;
/

--The following trigger will run after an update statement on STUDENTS table.
--Added with that, this trigger will only run when update is performed on the
--CGPA column.

CREATE OR REPLACE TRIGGER HELLO_WORLD4
AFTER UPDATE
OF CGPA
ON STUDENTS
DECLARE
BEGIN
	DBMS_OUTPUT.PUT_LINE('Hello World4');
END ;
/

--The following trigger will run after an update operation on the STUDENTS table.
--Added with that, the trigger will run once for each row. The previous rows will
```

```
--run once for the whole statement, where this trigger will run N times if N rows
--are affected by the SQL statement.

CREATE OR REPLACE TRIGGER HELLO_WORLD5
AFTER UPDATE
OF CGPA
ON STUDENTS
FOR EACH ROW
DECLARE
BEGIN
       DBMS_OUTPUT.PUT_LINE('Hello World5');
END ;
/
```

Read the above five trigger codes and understand them. After you properly understand all five triggers, let's now determine which trigger will be called by Oracle for the following five SQL statements.

```
INSERT INTO STUDENTS VALUES ('Shakib Ahmed', 3.63);
--This will run HELLO_WORLD, HELLO_WORLD2, HELLO_WORLD3

DELETE FROM STUDENTS WHERE CGPA < 3.65 ;
--This will run HELLO_WOLRD3

UPDATE STUDENTS SET CGPA = CGPA + 0.01 WHERE STUDENT_NAME LIKE '%Shakib%';
--This will run HELLO_WORLD4, but will not run HELLO_WORLD5!!! Why? Because
--HELLO_WORLD5 is declared with FOR EACH ROW clause. This means trigger should be
--run for each row affected. Since, the above statement does not update any row
--(as the previous DELETE operation already deleted that row from the table)
--it will not run HELLO_WORLD5!

UPDATE STUDENTS SET STUDENT_NAME = 'Fahim Ahmed'
WHERE STUDENT_NAME = 'Fahim Hasan' ;
--This will not run any trigger. Although HELLO_TRIGGER4 is declared to be run
--after an update operation, the trigger will not run because the update is done
--on the column STUDENT_NAME rather than CGPA

UPDATE STUDENTS SET CGPA = CGPA + 0.01 ;
--This will run both HELLO_WORLD4 and HELLO_WORLD5 trigger. However, HELLO_WORLD5
--will run twice! Why? Because two rows will be affected by the SQL statement!
```

After analyzing the above five triggers, you should now understand the following:

- A trigger can run be before or after a DML operation.
- The DML operation can be insert, update, delete or a combination of the three.
- The trigger can be created such that it is run only once after an SQL statement or it is run once per row affected by an SQL statement.

Classification of triggers

Trigger can be classified in following ways:

- *BEFORE trigger vs. AFTER trigger* – A BEFORE trigger is executed before a statement and an AFTER trigger is executed after the statement.
- *ROW LEVEL trigger vs. STATEMENT LEVEL trigger* – A ROW LEVEL trigger is created when FOR EACH ROW clause is specified in the trigger definition. In this case, the trigger will be called after or before each row that is affected by the operation. A STATEMENT LEVEL trigger is executed only once for each statement. So, if you issue a DML statement that affect N rows, then a STATEMENT LEVEL trigger will be executed once while a ROW LEVEL trigger will be executed N times. A STATEMENT LEVEL trigger must get executed once while a ROW LEVEL trigger may not get executed at all if the DML operation does not affect any row!!!

Practice

- Write a trigger HELLO_WORLD6 that will run after a deletion operation on the STUDENTS table. The trigger should be a ROW LEVEL trigger.

Problem Example 1

Write a PL/SQL trigger LOG_CGPA_UPDATE. This trigger will log all updates done on the CGPA column of STUDENTS table. The trigger will save current user's name, current system date and time in a log table named LOG_TABLE_CGPA_UPDATE.

Solution

The trigger required will be a STATEMENT LEVEL trigger. Because, it is required to run only per SQL statement which will serve our purpose.

Let's first create the LOG_TABLE_CGPA_UPDATE table as follows.

```
CREATE TABLE LOG_TABLE_CGPA_UPDATE
(
USERNAME VARCHAR2(25),
DATETIME DATE
) ;
```

The trigger code is shown below.

```
CREATE OR REPLACE TRIGGER LOG_CGPA_UPDATE
AFTER UPDATE
OF CGPA
ON STUDENTS
DECLARE
       USERNAME VARCHAR2(25);
BEGIN
       USERNAME := USER; --USER is a function that returns current username
       INSERT INTO LOG_TABLE_CGPA_UPDATE VALUES (USERNAME, SYSDATE);
END ;
/
```

After the trigger is created, issue the following update commands and then finally view the rows inserted into LOG_TABLE_CGPA_UPDATE table.

```
--First update
UPDATE STUDENTS SET CGPA = CGPA + 0.01 ;

--Another update
UPDATE STUDENTS SET CGPA = CGPA - 0.01 ;

--View the rows inserted by the trigger
SELECT * FROM LOG_TABLE_CGPA_UPDATE
```

Problem Example 2

Write a PL/SQL trigger BACKUP_DELETED_STUDENTS. This trigger will save all records that are deleted from the STUDENTS table into a backup table named STUDENTS_DELETED. The trigger will save student's record along with current user's name and current system date and time.

Solution

The trigger required will be a ROW LEVEL trigger. Because, it is required to run per row affected. Each row that will be deleted will need to be saved in the backup table.

Let's first create the backup table STUDENTS_DELETED.

```
CREATE TABLE STUDENTS_DELETED(
STUDENT_NAME VARCHAR2(25),
CGPA NUMBER,
USERNAME VARCHAR2(25),
DATETIME DATE
) ;
```

The following code shows the trigger definition required to perform the specified task.

```
CREATE OR REPLACE TRIGGER BACKUP_DELETED_STUDENTS
BEFORE DELETE
ON STUDENTS
FOR EACH ROW
DECLARE
       V_NAME VARCHAR2(25);
       V_USERNAME VARCHAR2(25);
       V_CGPA NUMBER;
       V_DATETIME DATE;
BEGIN
       V_NAME := :OLD.STUDENT_NAME ;
       V_CGPA := :OLD.CGPA ;
       V_USERNAME := USER ;
       V_DATETIME := SYSDATE ;
       INSERT INTO STUDENTS_DELETED VALUES (V_NAME,V_CGPA,V_USERNAME,V_DATETIME);
END ;
/
```

Note the use of :OLD special reference. This reference is used to access the column values of currently affected row by the DELETE operation. Like the :OLD reference, there is a :NEW reference that can be used to retrieve the column values of the new row that will result after completion of the operation.

After the trigger is compiled and stored successfully, issue the following command and view the rows inserted by the trigger.

```
--Delete the two rows
DELETE FROM STUDENTS WHERE CGPA < 3.85 ;

--View the rows inserted by the trigger
SELECT * FROM STUDENTS_DELETED ;
```

References :OLD vs. :NEW for a ROW LEVEL trigger

Note the following regarding :OLD and :NEW references-

- They are valid only for ROW LEVEL triggers! You are not allowed to use these in a STATEMET LEVEL trigger.
- For an update statement, :OLD is used to retrieve old values of columns while :NEW is used to retrieve new values of columns.
- For an insert statement, :OLD retrieves NULL values for all columns while :NEW can be used to retrieve column values of the row
- For a delete statement :NEW retrieves NULL values while :OLD can be used to retrieve column values of the row
- :NEW reference can be used to change column values of a row before it is going to be inserted in the table. *This will require a BEFORE trigger*.

To understand the :OLD and :NEW reference more deeply, write the following trigger and issue the SQL commands listed after the trigger.

```
CREATE OR REPLACE TRIGGER OLD_NEW_TEST
BEFORE INSERT OR UPDATE OR DELETE
ON STUDENTS
FOR EACH ROW
DECLARE
BEGIN
```

```
      DBMS_OUTPUT.PUT_LINE(':OLD.CGPA = ' || :OLD.CGPA) ;
      DBMS_OUTPUT.PUT_LINE(':NEW.CGPA = ' || :NEW.CGPA) ;
END ;
/

--Issue the following SQL statements and view the dbms outputs
INSERT INTO STUDENTS VALUES ('SOUMIK SARKAR', 3.85);

UPDATE STUDENTS SET CGPA = CGPA + 0.02 ;

DELETE FROM STUDENTS WHERE CGPA < 3.90;
```

Practice

- Write a trigger that will save a student records in a table named LOW_CGPA_STUDENTS which contain only one column to store student's names. The trigger will work before an update operation or an insert operation. Whenever the update operation results in a CGPA value less than 2.0, the trigger will be fired and the trigger will save the students name in the LOW_CGPA_STUDENTS table. Similarly, when an insert operation inserts a new row with CGPA less than 2.0, the corresponding row must be saved in the LOW_CGPA_STUDENTS table.

Problem Example 3

Write a PL/SQL trigger CORRECT_STUDENT_NAME. This trigger will be used to correct the text case of the student names when it is going to be inserted. So, this will be a BEFORE INSERT trigger. The trigger will change the case of the student's name to INITCAP format if it was not given by the user in the INSERT statement. The trigger will thus ensure that all names stored in the STUDENTS table will be in a consistent format.

```
CREATE OR REPLACE TRIGGER CORRECT_STUDENT_NAME
BEFORE INSERT
ON STUDENTS
FOR EACH ROW
DECLARE
BEGIN
      :NEW.STUDENT_NAME := INITCAP(:NEW.STUDENT_NAME) ;
```

```
END ;
/

--Issue the following SQL statements and then view the rows of STUDENTS table
INSERT INTO STUDENTS VALUES ('SHAkil ahMED', 3.80);

INSERT INTO STUDENTS VALUES ('masum billah', 3.60);
```

Note that the above trigger must be a BEFORE INSERT trigger. If you write an AFTER INSERT trigger, then it will not work. *Because in an AFTER INSERT trigger, you are not allowed to change values of the :NEW row.*

Practice

- Write down a PL/SQL trigger on STUDENTS table. The trigger will ensure that whenever a new row is inserted in the STUDENTS table, the name of the student contains only alphabetic characters. Name your trigger INVALID_NAME. If the name is valid, then insertion should be allowed. However, if the name is invalid, then insertion should be denied. To deny insertion, you can throw an exception from the trigger that would halt the insertion operation.

Drop a trigger from database

```
DROP TRIGGER <TRIGGER_NAME> ;
```

So, to drop the trigger OLD_NEW_TEST from the database, issue the following command.

```
DROP TRIGGER OLD_NEW_TEST ;
```

Basic Oracle Database Administration

Database Management and Administration **Course**

Bangladesh Korea Information Access Center (BK-IAC), Dept. of CSE, BUET

Author

Sukarna Barua

Assistant Professor

Dept. of CSE

Bangladesh University of Engineering and Technology (BUET)

Dhaka-1000, Bangaldesh.

This tutorial is indented for –

- Beginners who want to learn the basics of Oracle Database 11g Administration.
- Learning starting knowledge that can help one to start administering an Oracle database.

This tutorial is not intended for –

- Experts who already have a good knowledge of Oracle Database administration.
- Working as a complete reference of Oracle Database administration.

Contents

Chapter 1: Concepts of Oracle Database Architecture, Oracle Instance, Oracle Administration

1 Oracle Database Architecture

A database is a –

- Collection of related, relevant data that is maintained in a centralized manner with minimum redundancy and maximum consistency.
- A database gives the user flexibility of storing data, updating data, deleting data, etc.
- Internally, the database is stored in physical files typically by the operating system files.

A database server is –

- Reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data.
- Delivers high performance.
- Prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle database is maintained by an Oracle Server installed in a computer.

Oracle Server

The Oracle Server comprises of the following two things:

- *Oracle Instance* in memory and
- The *Oracle Database* in disc.

Oracle Instance

The Oracle instance provides the means by which a user can access an Oracle database. The Oracle Instance is made up of two things (Figure 1) –

- *Memory structures*: These memory structures are created in the real memory of the computer. The instance manages the access to the database stored in physical files.

- *Background processes*: These are created by the Oracle Server when the database is started. The background processes of the instance perform various input-output (I/O) functions of behalf of the database.

Oracle Database

Oracle database consists of the data stored in physical files. These files are called *data files*. Beside the data files oracle database also use many other important files for its operation. They include c*ontrol files, parameter files, redo log files*, etc. All these files are stored in hard disc and maintained by either oracle server process or operating system.

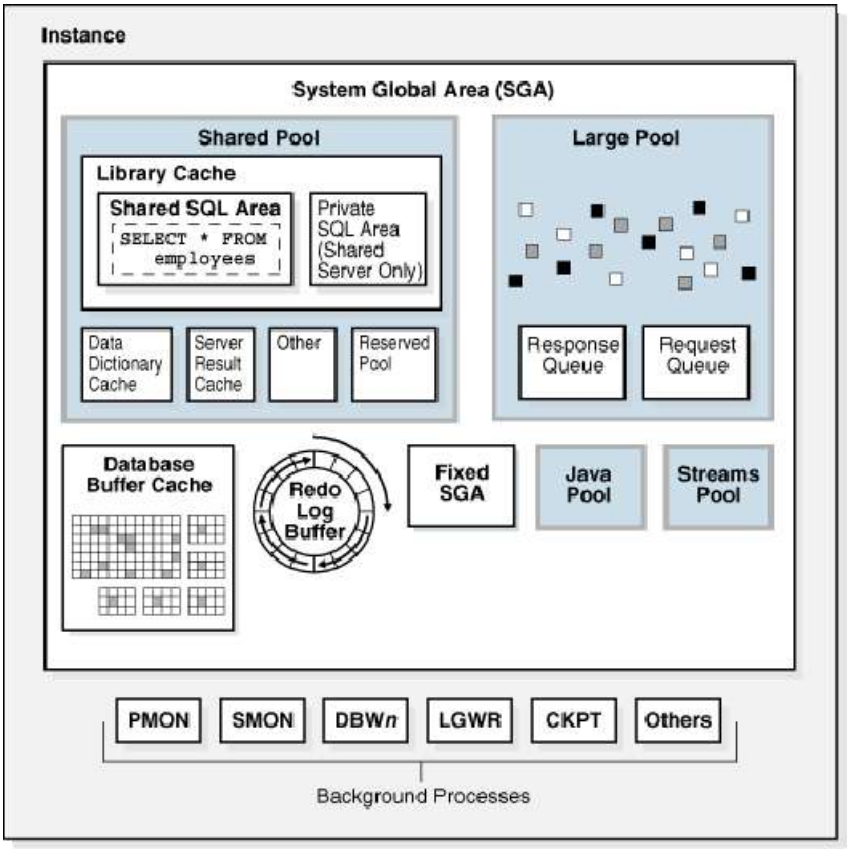

Figure 1: Oracle Database Instance showing memory structure and background processes.

Oracle Server Process

When a client connects to an oracle database, one server process is created for the user to handle the SQL or PL/SQL requests of the user. This server process is created in the server and is live till the user is logged in the database.

Functions of different SGA components	
Shared Pool	The shared pool is the area in the SGA that holds data and control information that can be shared between different users of the database. It consists of two parts, namely the Library Cache and the data dictionary cache.
Library Cache	The library cache contains the most recently executed SQL statements, their parsed code and execution plans. The function of the library cache is to allow sharing of parsed code and execution plans during SQL statement processing. Sharing this information speeds up query processing of similar statements in the database. Data Dictionary Cache - The data dictionary cache holds data dictionary information in memory.
Database Buffer Cache	The Database Buffer Cache holds data blocks that have been read into memory during the execution of SELECT and Data Manipulation Language statements (DML) statements. Data is modified in the database buffer cache. A modified block is referred to as a dirty block. A dirty block must be written back to the datafiles.
Redo Log Buffer	The redo log buffer contains a record of all the changes made in the database. The changes written to the redo log buffer are called redo entries.
Large Pool	The large pool is an optional memory structure that may be configured to enhance I/O functions and satisfy memory requirements of an Oracle Shared Server configuration, Recovery Manager and for Parallel Query Processing.
Java Pool	This optional pool may be configured when installing and using Java code. The pool is used for servicing the parsing requirements of Java-based applications.

Table 1: Functions of different SGA components of Oracle Instance

1.1 Oracle Instance Memory Architecture

The basic memory structures associated with Oracle Database are described below.

1.1.1 System Global Area (SGA)

When an oracle instance is started, Oracle Database allocates an area in memory called System Global Area (SGA) (Figure 1). The SGA serves various purposes, including the following:

- Maintaining internal data structures that are accessed by many processes and threads concurrently
- Caching data blocks read from disk
- Buffering redo data before writing it to the online redo log files
- Storing SQL execution plans

The SGA is made up of a number of individual memory structures which are summarized in Table 1. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

1.1.2 Program Global Area (PGA)

A PGA is a memory region that contains data and control information for a server process. It is non-shared memory created by Oracle Database when a server process is started. Access to the PGA is exclusive to the server process. There is one PGA for each server process. Background processes also allocate their own PGAs. The total PGA memory allocated for all background and server processes attached to an Oracle Database instance is referred to as the total instance PGA memory.

1.1.3 SGA Components

The most important SGA components are the following:

- Database Buffer Cache
- Redo Log Buffer
- Shared Pool
- Large Pool
- Java Pool
- Streams Pool

Table 1 illustrates the functions of these different components of SGA.

Physical Files	
Data Files	These files store the data dictionary and the user data.
Control Files	A control file contains information about the database such as its name, date of creation, names of the other physical files of the database, synchronization, log sequence, backup and recovery, archiving information etc.
Redo Log Files	The redo log files contain a record of all the changes that are made in the database.
Parameter Files	Parameter files contain parameter names and values. These parameters are used to configure the Oracle instance and data structures, specify values for various processes, identify the name of the database and instance etc.
Password File	The password file is a file that is used to authenticate privileged users of the database. It contains the usernames and passwords of individuals who have been granted the SYSDBA or SYSOPER role.
Archive Files	These are optional files that are created when the database is running in ARCHIVELOG mode. Archive files are offline copies of the redo log files.
Trace file	Each server and background process can write to an associated trace file. When an internal error is detected by a process, it dumps information about the error to its trace file.
Alert log file	The alert log is a chronological log of messages and errors, and includes the following items as internal errors, block corruption error, deadlock errors, administrative operations (CREATE, ALTER, DROP statements), values of initialization parameters, etc.

Table 2: Oracle Database Physical Files

1.2 The Physical Files of the Oracle Database

The physical files of oracle database are –

- Control Files
- Datafiles and
- Redo Log Files.
- Password File, Parameter Files, Archived Files, Alert and Trace files.

1.2.1 The Control Files

Control Files are binary files that contain information about the database and are needed to mount the database. It contains information such as the

- Database Identifier
- Database name
- Creation date
- The names and locations of the datafiles and redo log files,
- Tablespace information
- Log history, log sequence information, checkpoint data and backup information i

The names and locations of the conttrolfiles are stored in the parameter files of the database. Control files typically have the .ctl extension, for example the name of a controlfile can be *control01.ctl*. When a database instance is started, control files are read to retrieve information of the database.

1.2.2 The DataFiles

The datafiles of a database store data. Data files are related to a logical component of a database called a tablespace. Tablespaces are logical structures of the database that store data in form of segments. Tablespaces can be permanent, temporary or of undo type. Tablespaces that are temporary store data that is temporary in nature like data that is generated during sort operations. Undo tablespaces store data that is created to store the previous value of data that is manipulated. An Oracle database has two datafiles that are mandatory, the file(s) belonging to the SYSTEM tablespace and the file(s) belonging to the SYSAUX tablespace.

1.2.3 The Redo Log Files

The online redo logs files store the details of all the changes that are performed against the database. The changes could be data manipulation, data definition, or structural changes to the database. Every database must have a minimum of 2 redo logs. Redo log files belong to groups, called redo log groups. The group can contain members and every physical redo log file belongs to a group. It would be proper to say that every database must have a minimum of two redo groups with one member. The redo log files typically have the .log extension, for example the name of a redo log could be *redo01.log*.

1.2.4 The Parameter File(s)

The Oracle Database has two types of parameter files, the Initialization Parameter File (pfile) and the Server Parameter Files (spfile).

- The pfile is a text file that can be opened, read and edited using a simple text editor.
- The spfile is a binary file that is updated by the Oracle software when you make changes to initialization parameters.

The parameter file(s) are used for database startup and used to configure the Oracle Instance and the Oracle database. It contains parameters and values. If both the pfile and spfile are available, the spfile is used to starting up the database. Typically the parameter files have an .ora extension, for example *initorcl.ora* (pfile) and *spfileorcl.ora* (spfile).

1.2.5 The Password File

The password file contains authentication information for an Oracle database. It stores the names and passwords of users who have possess the SYSDBA or SYSOPER roles. These two are privileged roles and give a user the ability to perform administrative tasks in the database, such as startup, shutdown etc. The password file is a binary file and optional.

1.3 Oracle Database Processes

All connected Oracle Database users must run two modules of code to access an Oracle Database instance.

- *Application or Oracle tool*: A database user runs a database application (such as a precompiler program) or an Oracle tool (such as SQL*Plus), which issues SQL statements to an Oracle database.

- *Oracle database server code*: Each user has some Oracle database code executing on his or her behalf, which interprets and processes the application's SQL statements.

1.3.2 Oracle Database Server Processes

Oracle Server processes) created on behalf of each user's application can perform one or more of the following:

- Parse and run SQL statements issued through the application
- Read necessary data blocks from datafiles on disk into the shared database buffers of the SGA, if the blocks are not already present in the SGA
- Write and modifies data blocks in the database buffers if required
- Return results to users in such a way that the application can process the information

Oracle database can be configured to have -

- Only one server process for all users (*shared server architecture*)
- One server process per user (*dedicated server architecture*).

Each of the two architectures has its own advantages and disadvantages and will be described later.

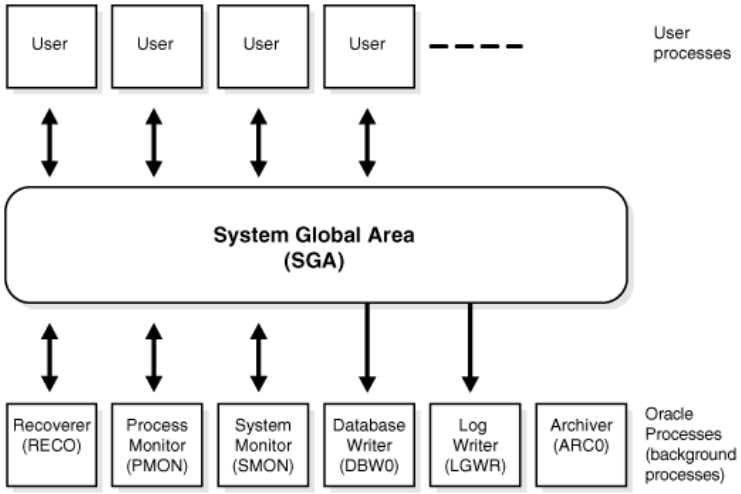

Figure 3: A simplified view of the interaction between oracle processes.

1.3.3 Oracle Database Background Processes

To maximize performance and accommodate many users, a multi-process Oracle Database system uses some additional Oracle Database processes called background processes. The functions of different background processes are summarized in Table 3.

Background Processes	
Database Writer (DBWn)	Is responsible for transferring all modified blocks from the database buffer cache in memory to the data-files on disk.
Log Writer (LGWR)	Is responsible for transferring the contents of the redo log buffer from memory to the online redo log files on disk.
System Monitor(SMON)	Performs instance recovery in the event of an instance failure in addition to other functions.
Process Monitor(PMON)	Is responsible for cleaning up failed user processes that is created when a user process is terminated abnormally.
Checkpoint (CKPT	The checkpoint process is responsible for ensuring database synchronization. It instructs the database writer to write a group of changes to the datafiles. After the write has completed it updates the headers of the data files and control files.
Archiver (ARCH)	This optional process is responsible for automatically archiving the contents of the online redo log files to archive log files. This is done when the database is running in ARCHIVELOG mode.

Table 3: Functions of Oracle Database Background Processes

1.4 Oracle Client/Server Architecture

An Oracle database system uses the familiar *client/server architecture*. In this architecture, the database system is divided into two parts: a front-end or a client, and a back-end or a server. The client software is run in client machine and the server software is run in server machine. Whenever client wants to do some database activity in the server, it starts the client software in its client machine and connects to the server software in the server machine via the network (Figure 5).

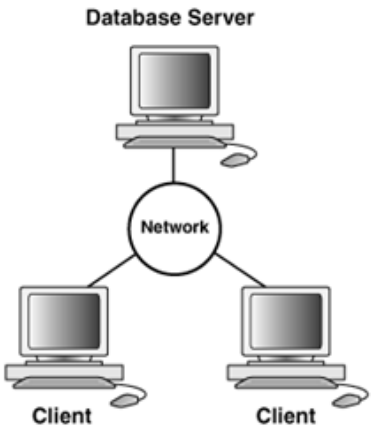

Figure 5: Client and Server. Client software residing in client machine connects with database server software running in the server via network.

The Client

The client is a database application that initiates a request for an operation to be performed on the database server. It requests, processes, and presents data managed by the server. The client machine can be optimized for its job. For example, it might not need large disk capacity, or it might benefit from graphic capabilities. Many clients can simultaneously connect to one server.

The Server

The server runs Oracle software and handles the functions required for concurrent, shared data access. The server receives and processes the SQL and PL/SQL statements that originate from client applications. The computer that manages the server can be optimized for its duties. For example, it can have large disk capacity and fast processors.

1.5 Database Administration

A database administrator (short form DBA) is a person responsible for the installation, configuration, upgrade, administration, monitoring and maintenance of databases in an organization. The role includes the development and design of database strategies, system monitoring and improving database performance and capacity, and planning for future expansion requirements. They may also plan, co-ordinate and implement security measures to safeguard the database.

1.5.1 Tasks of Database Administrator

Each database requires at least one database administrator (DBA). A database administrator's responsibilities can include the following tasks:

- Installing and upgrading the Oracle Database server and application tools
- Allocating system storage and planning future storage requirements for the database system
- Creating primary database storage structures (tablespaces) after application developers have designed an application
- Creating primary objects (tables, views, indexes) once application developers have designed an application
- Modifying the database structure, as necessary, from information given by application developers
- Enrolling users and maintaining system security
- Controlling and monitoring user access to the database
- Monitoring and optimizing the performance of the database
- Planning for backup and recovery of database information
- Maintaining archived data on tape
- Backing up and restoring the database

1.6 How to Administer Oracle Database

The database administrators oversee the database activity and perform administrative tasks. They communicate with the database by connecting using database tools. The primary means of communicating with Oracle Database is by submitting SQL statements. There are three ways to submit these SQL statements and commands to Oracle Database:

- Directly, using the command-line interface of SQL*Plus software
- Indirectly, using the graphical user interface of Oracle Enterprise Manager
- Directly, using SQL Developer. Developers use SQL Developer to create and test database schemas and applications, although you can also use it for database administration tasks.

1.6.1 Oracle Database Administration Software Tools

Table 4 briefly illustrates the Oracle Database Administration Software Tools that are automatically installed with oracle server software.

DBA Software	Software Description
SQLPlus	Used by the DBA and system users to access data in an Oracle database.
Oracle Universal Installer	This software is the standard software used to install, modify (upgrade), and remove Oracle software components for all Oracle products.
Oracle Database Configuration Assistant	This is a GUI tool that can be used to create, delete, or modify databases; however, it does not provide a lot of control on the database creation process.
Oracle Enterprise Manager	A GUI tool for administering one or more databases.
Database Upgrade Assistant (DBUA)	Can be started in command line mode (command is dbua) for LINUX, or by selecting the DBUA from the Oracle Configuration and Migrations Tools menu option – this upgrades Oracle databases to version 10g.

Table 4: Database administration software tools installed in a computer.

1.6.2 SQL*Plus

SQL*Plus is the primary command-line interface to your Oracle database. Before you can submit SQL statements and commands, you must connect to the database residing in server computer. With SQL*Plus, you can connect locally or remotely.

- *Connecting locally* means connecting to an Oracle database running on the same computer on which you are running SQL*Plus.
- *Connecting remotely* means connecting over a network to an Oracle database that is running on a remote computer. Such a database is referred to as a remote database. The SQL*Plus executable on the local computer is provided by a full Oracle Database installation, or an Oracle Client installation, or an Instant Client installation.

When you connect with SQL*Plus, you are connecting to the Oracle instance. Each instance has an instance ID, also known as a system ID (SID). Because there can be more than one Oracle instance on a host computer, each with its own set of data files, you must identify the instance to which you want to connect. For a local connection, you identify the instance by setting operating system environment

variables. For a remote connection, you identify the instance by specifying a network address and a database service name/instance identifier.

For both local and remote connections, you must set environment variables to help the operating system find the SQL*Plus executable and to provide the executable with a path to its support files and scripts. To connect to an Oracle instance with SQL*Plus, therefore, you must complete the following steps:

Step 1: Open a Command Window

Take the necessary action on your platform to open a window into which you can enter operating system commands.

Step 2: Set Operating System Environment Variables

Depending on your platform, you may have to set environment variables before starting SQL*Plus, or at least verify that they are set properly.

For example, on most platforms, ORACLE_SID and ORACLE_HOME must be set. In addition, it is advisable to set the PATH environment variable to include the ORACLE_HOME/bin directory. On the Windows platform, Oracle Universal Installer (OUI) automatically assigns values to ORACLE_HOME and ORACLE_SID in the Windows registry. If you did not create a database upon installation, OUI does not set ORACLE_SID in the registry; after you create your database at a later time, you must set the ORACLE_SID environment variable from a command window as follows:

CMD> set ORACLE_SID=orcl

Step 3: Start SQL*Plus

Enter the following command (case sensitive on UNIX and Linux):

CMD> sqlplus /nolog

Step 4: Submit the SQL*Plus CONNECT Statement

You submit the SQL*Plus CONNECT statement to initially connect to the Oracle instance or at any time to reconnect as a different user. The syntax of the CONNECT statement is as follows:

CONN[ECT] [logon] [AS {SYSOPER | SYSDBA}]

The syntax of logon is: {username | / }[@connect_identifier]

When you provide username, SQL*Plus prompts for a password. The password is not echoed as you type it. The following table describes the syntax components of the CONNECT statement.

/	Used for operating system authentication, where the database user is authenticated by having logged in to the host operating system with a certain host user account.
AS {SYSOPER | SYSDBA}	Indicates that the database user is connecting with either the SYSOPER or SYSDBA system privilege. **If you are going to administer the oracle database, you need to login using this feature.**
username	A valid database user name. The database authenticates the connection request by matching username against the data dictionary and prompting for a user password.
connect_identifier (1)	An Oracle Net connect identifier, for a remote connection. The exact syntax depends on the Oracle Net configuration. If omitted, **SQL*Plus attempts connection to a local instance**. A common connect identifier is a net service name. This is an alias for an Oracle Net connect descriptor (network address and database service name). The alias is typically resolved in the tnsnames.ora file on the local computer, but can be resolved in other ways.
connect_identifier (2)	As an alternative, a connect identifier can use **easy connect** syntax. Easy connect provides out-of-the-box TCP/IP connectivity for remote databases without having to configure Oracle Net Services on the client (local) computer. Easy connect syntax for the connect identifier is as follows: host[:port][/service_name] where: *host* is the host name or IP address of the computer hosting the

	remote database. *port* is the TCP port on which the Oracle Net listener on host listens for database connections. If omitted, 1521 is assumed. *service_name* is the database service name. Can be omitted if the Net Services listener configuration on the remote host designates a default service. If no default service is configured, service_name must be supplied.

Here are some examples of connecting to an oracle database server:

- This simple example connects to a local database as user SYSTEM. SQL*Plus prompts for the SYSTEM user password.
  connect system
- This example connects to a local database as user SYS with the SYSDBA privilege. SQL*Plus prompts for the SYS user password.
  connect sys as sysdba
- This example connects locally with operating system authentication.
  connect /
- This example connects locally with the SYSDBA privilege with operating system authentication.
  connect / as sysdba
- This example uses easy connect syntax to connect as user salesadmin to a remote database running on the host db1.mycompany.com. The Oracle Net listener (the listener) is listening on the default port (1521). The database service is sales.mycompany.com. SQL*Plus prompts for the salesadmin user password.
  connect salesadmin@'db1.mycompany.com/sales.mycompany.com'
- This example is identical to previous, except that the listener is listening on the non-default port number 1522.
  connect salesadmin@'db1.mycompany.com:1522/sales.mycompany.com'
- This example connects remotely with the SYSDBA privilege and with external authentication to the database service designated by the net service name sales1.
  connect /@sales1 as sysdba

1.6.3 Identifying Your Oracle Database Software Release

To understand the release nomenclature used by Oracle, examine the example of Figure 7 of an Oracle Database server labeled "Release 10.1.0.1.0".

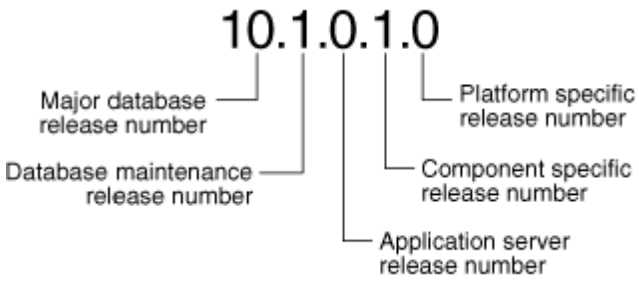

Figure 7: An oracle release details.

- Major Database Release Number: The first digit is the most general identifier. It represents a major new version of the software that contains significant new functionality.
- Database Maintenance Release Number: The second digit represents a maintenance release level. Some new features may also be included.
- Application Server Release Number: The third digit reflects the release level of the Oracle Application Server (OracleAS).
- Component-Specific Release Number: The fourth digit identifies a release level specific to a component. Different components can have different numbers in this position depending upon, for example, component patch sets or interim releases.
- Platform-Specific Release Number: The fifth digit identifies a platform-specific release. Usually this is a patch set. When different platforms require the equivalent patch set, this digit will be the same across the affected platforms.

Chapter 2: Instance startup, shutting down.

2 Starting Up and Shutting Down Oracle Database

To initiate database shutdown, you can use the SQL*Plus SHUTDOWN command. Control is not returned to the session that initiates a database shutdown until shutdown is complete. To shut down a database and instance, you must first connect to the database server as SYSOPER or SYSDBA. There are several modes for shutting down a database. These are discussed in the following sections.

2.1 Overview of Instance and Database Startup

The three steps to starting an Oracle database and making it available for system wide use are:

- Start an instance.
- Mount the database.
- Open the database.

How an Instance Is Started

When Oracle Database starts an instance, it reads the server parameter file (SPFILE) or initialization parameter file (PFILE) to determine the values of initialization parameters. Then, it allocates an SGA, which is a shared area of memory used for database information, and creates background processes. At this point, no database is associated with these memory structures and processes.

How a Database Is Mounted

The instance mounts a database to associate the database with that instance. To mount the database, the instance finds the database control files and opens them. Control files are specified in the CONTROL_FILES initialization parameter in the parameter file used to start the instance. Oracle Database then reads the control files to get the names of the database's datafiles and redo log files.

At this point, the database is still closed and is accessible only to the database administrator. The database administrator can keep the database closed while completing specific maintenance operations. However, the database is not yet available for normal operations.

How a database is opened

Opening a mounted database makes it available for normal database operations. Any valid user can connect to an open database and access its information. Usually, a database administrator opens the database to make it available for general use.

When you open the database, Oracle Database opens the online datafiles and redo log files. If any of the datafiles or redo log files is not present when you attempt to open the database, then Oracle Database returns an error. You must perform recovery on a backup of any damaged or missing files before you can open the database.

2.2 Different Shut-Down commands

Oracle database can be shut down with shutdown oracle command. However, we may need to apply different versions of this command in different situations.

Shutting Down with the NORMAL Clause

To shut down a database in normal situations, use the SHUTDOWN command with the NORMAL clause:

SQL> SHUTDOWN NORMAL

The NORMAL clause is optional, because this is the default shutdown method if no clause is provided. Normal database shutdown proceeds with the following conditions:

- No new connections are allowed after the statement is issued.
- Before the database is shut down, the database waits for all currently connected users to disconnect from the database.
- The next startup of the database will not require any instance recovery procedures.

Shutting Down with the IMMEDIATE Clause

To shut down a database immediately, use the SHUTDOWN command with the IMMEDIATE clause:

SQL> SHUTDOWN IMMEDIATE

Immediate database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- Any uncommitted transactions are rolled back. (If long uncommitted transactions exist, this method of shutdown might not complete quickly, despite its name.)
- Oracle Database does not wait for users currently connected to the database to disconnect. The database implicitly rolls back active transactions and disconnects all connected users.
- The next startup of the database will not require any instance recovery procedures.

Use immediate database shutdown only in the following situations:

- To initiate an automated and unattended backup
- When a power shutdown is going to occur soon
- When the database or one of its applications is functioning irregularly and you cannot contact users to ask them to log off or they are unable to log off

Shutting Down with the TRANSACTIONAL Clause

When you want to perform a planned shutdown of an instance while allowing active transactions to complete first, use the SHUTDOWN command with the TRANSACTIONAL clause:

SQL> SHUTDOWN TRANSACTIONAL

Transactional database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- After all transactions have completed, any client still connected to the instance is disconnected.
- At this point, the instance shuts down just as it would when a SHUTDOWN IMMEDIATE statement is submitted.
- The next startup of the database will not require any instance recovery procedures.
- A transactional shutdown prevents clients from losing work, and at the same time, does not require all users to log off.

Shutting Down with the ABORT Clause

When you must do a database shutdown by aborting transactions and user connections, issue the SHUTDOWN command with the ABORT clause:

SQL> SHUTDOWN ABORT

An aborted database shutdown proceeds with the following conditions:

- No new connections are allowed, nor are new transactions allowed to be started, after the statement is issued.
- Current client SQL statements being processed by Oracle Database are immediately terminated.
- Uncommitted transactions are not rolled back.
- Oracle Database does not wait for users currently connected to the database to disconnect. The database implicitly disconnects all connected users.
- The next startup of the database will require instance recovery procedures.

You can shut down a database instantaneously by aborting the database instance. If possible, perform this type of shutdown only in the following situations:

- The database or one of its applications is functioning irregularly and none of the other types of shutdown works.
- You need to shut down the database instantaneously (for example, if you know a power shutdown is going to occur in one minute).
- You experience problems when starting a database instance.

2.3 Starting Up a Database

When you start up a database, you create an instance of that database and you determine the state of the database. Normally, you start up an instance by mounting and opening the database. Doing so makes the database available for any valid user to connect to and perform typical data access operations. Other options exist, and these are also discussed in this section.

Starting Up a Database Using SQL*Plus

You can start a SQL*Plus session, connect to Oracle Database with administrator privileges, and then issue the STARTUP command. When you issue the SQL*Plus STARTUP command, the database attempts to read the initialization parameters from an SPFILE in a platform-specific default location. If it finds no SPFILE, it searches for a text initialization parameter file (PFILE).

Table 1 describes and illustrates the various states in which you can start up an instance.

SQL COMMAND	Database Behavior
STARTUP	Normal database operation means that an instance is started and the

	database is mounted and open. This mode allows any valid user to connect to the database and perform data access operations. (You can optionally specify a PFILE clause.)
STARTUP NOMOUNT	You can start an instance without mounting a database. Typically, you do so only during database creation.
STARTUP MOUNT	You can start an instance and mount a database without opening it, allowing you to perform specific maintenance operations. For example, the database must be mounted but not open during the following tasks: Enabling and disabling redo log archiving options, Performing full database recovery
STARTUP RESTRICT	You can start an instance, and optionally mount and open a database, in restricted mode so that the instance is available only to administrative personnel (not general database users).

Table 1: STARTUP command illustrated.

Altering Database Availability

You can alter the availability of a database. For example, to mount a database to a previously started, but not opened instance, use the SQL statement ALTER DATABASE with the MOUNT clause as follows:

SQL>ALTER DATABASE MOUNT;

You can make a mounted but closed database available for general use by opening the database. To open a mounted database, use the ALTER DATABASE statement with the OPEN clause:

SQL>ALTER DATABASE OPEN;

Practice Lab 1: Study SHUTDOWN IMMEDIATE and STARTUP NOMOUNT command.

1. Connect to database using SYS account with the SYSDBA privilege.
2. Shutdown your database using SHUTDOWN IMMEDIATE command.
3. Start the database in NOMOUNT state.
4. Query the views V$INSTANCE and V$SGA. [**A view is like a database table. Take help from your instructor to learn about these views! Later, you will be much familiar with these.**]

5. What information are stored in the V$INSTANCE view?
6. What information are stored in the V$SGA view?
7. Attempt to query V$DATABASE and V$CONTROLFILE views. What happens?

Lab Practice 2: Study ALTER DATABASE command. [Do this lab after you have completed Lab practice 1]

1. Mount the database from NOMOUNT stage using ALTER DATABASE command.
2. Execute these queries:
    a. SELECT name FROM V$DATABASE;
    b. SELECT instance_name, status FROM V$INSTANCE;
    c. SELECT * FROM V$CONTROLFILE;
    d. SELECT file#, status FROM V$DATAFILE;
3. What happens in each case? What is the database status column contain in V$INSTANCE?
4. Use ALTER DATABASE to OPEN the database for normal operation.
5. Query V$INSTANCE. What is the database status shown in V$INSTANCE?

Lab Practice 3: Study the effects of SHUTDOWN commands. [Can be done as homework practice]

1. Start two separate SQL*Plus sessions. Connect using SYS AS SYSDBA in one of the sessions. Connect with HR account in the 2nd session. [**Take help from your instructor to learn how to connect using HR account**]
2. In the 1st session (SYSDBA session), enter the SHUTDOWN command with no parameters. What happens?
3. Unblock the shutdown by quitting the HR session. What happens now in the 1st session?
4. In the SYSDBA session, restart the database.
5. Logon in the second session again using HR account and enter the following commands:
    a. create table t1 (c1 number);
    b. insert into t1 values(1);
6. In the SYSDBA session, Shutdown the database with the transactional clause. What happens?
7. Unblock the shutdown by issuing a COMMIT in the Non-SYSDBA session.
8. Do Step 5 and 6 again, however now shutdown the database with immediate clause. What happens?

Chapter 3: Parameter files

3 Parameter files of Oracle Database

3.1 What are Oracle parameter files

When an Oracle Instance is started, the characteristics of the Instance are established by parameters specified within the initialization parameter file. Oracle database configures the instance according to the values specified in parameter file. Several configuration parameter values can be set via this parameter file such as:

- The name of the database to connect to (parameter DB_NAME)
- The database block size (parameter DB_BLOCK_SIZE)
- The size of the database buffer cache (parameter DB_CACHE_SIZE)
- The location and name of all control files (parameter CONTROL_FILES).

Usually, these files are located in some default location; however, the location can be changed by the user.

Starting from Oracle 8i, the Oracle database offers two types of parameter files. These are

- Server Parameter File (SPFILE) – Available for use in newer databases starting from version 8i
- Initialization Parameter File (PFILE) –Used in earlier database

3.1.1 PFILE vs. SPFILE

The SPFILE was introduced in Oracle 8i and is the preferred type of parameter file. In the situation where both types of parameter files exist, the database instance will be started with the SPFILE.

- A PFILE is a static, client-side text file that must be changed with a standard text editor like "notepad" (windows) or "vi" (linux). DBA's commonly refer to this file as the INIT.ORA file.
- An SPFILE on the other hand, is a persistent server-side binary file that can only be modified with the "ALTER SYSTEM SET" oracle command (described later).

To better understand the behavior/use of these files and why the SPFILE is preferred, we will first look at the characteristics of each file.

3.2 Characteristics of the Initialization Parameter File (PFILE)

- A text file
- Naming convention is <initSID.ora>
- Default location: $ORACLE_HOME/dbs (Linux), %ORACLE_HOME%/database (windows)
- Editable by notepad, vi, etc. text editor
- Dynamic changes affect only the currently running instance.
- Permanent changes require the PFILE to be opened, edited and a restart to the database.

3.2.1 Dynamic changes to the PFILE and associated problems

Let us say, the value of the DB_CACHE_SIZE parameter is some value X. To change the value of the DB_CACHE_SIZE to a new value Y, you can issue the command:

SQL>ALTER SYSTEM SET DB_CACHE_SIZE=Y

The above change would come into effect and would affect the currently running instance only. If the database is restarted, the previous value that existed (X) in the parameter file would be used for configuring the size of the database buffer cache.

To permanently change the size of the database buffer cache to Y, you would need to open the PFILE, and edit the initialization parameter to take the value Y. To bring this into effect you would need to restart the database. **There is no other way to change parameter values in a PFILE except editing it with text editors.**

What is the problem in manually editing this PFILE? The parameter file consists of parameters and value and the likelihood of accidentally making an error is very high. If you accidentally make an incorrect change or have a typo or change the wrong parameter, the database will not start up properly. This problem can be solved by using SPFILE (described in later section).

Another problem with PFILE is that if you are connecting to your database from a remote (client) machine, the text parameter file must be located on your client – not on the server.

3.3 Characteristics of the Persistent (Server) Parameter File (spfile)

- A binary file
- Naming convention is <spfileSID.ora>
- Default location: $ORACLE_HOME/dbs (Linux), %ORACLE_HOME%/database (windows)
- **Not Editable by text editors, e.g. notepad, vi.**

- Dynamic changes may affect only the currently running instance, only future instances or both currently running and future instances.

Changes of parameters in SPFILE and Advantages over PFILE

To better understand the advantage of using the SPFILE over the PFILE, let us see how a dynamic change can be made to the SPFILE. To begin let us say the value of the DB_CACHE_SIZE initialization parameter is some value X in the SPFILE. When using the SPFILE you can use the SCOPE keyword with the ALTER SYSTEM SET command. The keyword takes the following options:

- SCOPE=MEMORY => Change affects only current running instance
- SCOPE=SPFILE => Change is written into the spfile and affects only future instances (after database restart)
- SCOPE=BOTH => Change is written into the spfile and also affects the currently running instance. Because the change is made to the spfile, the changed value remains in effect after database restart.

For example, to change the value of the DB_CACHE_SIZE initialization parameter to a new value Y for the currently running instance, you can issue the command:

SQL> ALTER SYSTEM SET DB_CACHE_SIZE=Y SCOPE=MEMORY;

The above change would come into effect and would affect the currently running instance only (due to SCOPE=MEMORY option).

To change the value of the DB_CACHE_SIZE initialization parameter for only future instances you could issue:

SQL> ALTER SYSTEM SET DB_CACHE_SIZE=Y SCOPE=SPFILE

The above change will not affect the currently running instance. However upon database restart the new value will be in effect. This type of changes are very useful that cannot be done while using PFILE.

To change the value of the DB_CACHE_SIZE initialization parameter for the currently running instance and for future instances, you can issue the command:

SQL> ALTER SYSTEM SET DB_CACHE_SIZE=Y SCOPE=BOTH;

The above change would come into effect and would affect the currently running instance as well as future instances. When the database is restarted the new value will be in effect.

Based on the above discussion on the SPFILE you should note that the user can never directly open (in a text editor) and modify SPFILE. Any change to the SPFILE must be made by issuing the ALTER

SYSTEM command which causes Oracle to write into the file. This removes any possibility of user errors created by manually editing the file.

So, in summary, SPFILEs provide the following advantages over PFILEs:

- Reduce human errors. The SPFILE is maintained by the server. Parameters are checked before changes are accepted.
- Eliminate configuration problems (no need to have a PFILE in every remote machine from which you STARTUP your oracle database)
- Easy to find - stored in a central location

3.4 Converting between parameter files in Oracle

You can create a PFILE from an SPFILE or vice versa using the following commands:

SQL> CREATE PFILE FROM SPFILE;

SQL> CREATE SPFILE FROM PFILE;

3.4.2 How database startup when there are both PFILE and SPFILE?

If there are both PFILE and SPFILE in the default location of these files, then oracle searches as follows:

- **Step 1:** Search for a file called: $ORACLE_HOME\DATABASE\SPFILE%ORACLE_SID%.ORA. If found, then Oracle uses this file.
- **Step 2**: If Step 1 fails, oracle next performs the following search for a file called: $ORACLE_HOME\DATABASE\SPFILE.ORA. If found, then Oracle uses this file.
- **Step 3**: If Step 2 fails too, oracle performs the following search: $ORACLE_HOME\DATABASE\PFILE\INIT%ORACLE_SID%.ORA. If found, then Oracle uses this file. If not found, Oracle generates an error.

Remember that, $ORACLE_HOME is the value of the Oracle home directory. The OS paths specified above assume a Windows server.

3.5 Viewing Parameters from SQL Command Line

One can view parameter values using one of the following methods (regardless if they were set via PFILE or SPFILE):

- The "SHOW PARAMETERS" command from SQL*Plus (e.g., SHOW PARAMETERS DB_CACHE_SIZE)
- V$PARAMETER view - display the currently in effect parameter values
- V$SPPARAMETER view - display the current contents of the server parameter file.

Lab Activity 1: Study PFILE and SPFILE.

1. Check if your current instance is using a server parameter file. If it is, proceed to step 4. If not, proceed to step 2. [**Take help from your instructor to learn how to know this**]
2. Create a SPFILE from the PFILE.
3. Restart your instance with the new SPFILE.
4. Reduce the size of the shared pool (SHARED_POOL_SIZE) by 4 Mb. Make this change persistent.
5. Restart the database. Check that the change you made in step 4 is still in effect.
6. Create a text version (PFILE) from SPFILE. View the contents using a text editor (e.g., NOTEPAD in windows).
7. Restart your instance by forcing it to use the PFILE you created. [**Take help from your instructor to learn how to do this**]
8. What is the current value of your SHARED_POOL_SIZE parameter?
9. Shutdown your database.
10. Remove your SPFILE by renaming it. [**Take help from your instructor to learn how to do this**]
11. Startup your database again. Is the SPFILE being used?
12. Shutdown again, then rename your SPFILE to its original name.
13. Startup your database without any parameter. Is the SPFILE being used?

Chapter 4: Alert Log File and Trace files

4 Oracle Diagnostic Files: Alert and Trace Files

The fault, alert and trace aids in preventing, detecting, diagnosing, and resolving problems. The problems that are targeted in particular are critical errors such as those caused by database code bugs, metadata corruption, and customer data corruption.

When a critical error occurs in Oracle database, it is assigned an incident number, and diagnostic data for the error (such as trace files) are immediately captured and tagged with this number. The data is then stored in the Automatic Diagnostic Repository (ADR)—a file-based repository outside the database—where it can later be retrieved by incident number and analyzed. Oracle's log and trace files chronologically records messages and errors arising from the daily database operation.

4.1 Automatic Diagnostic Repository (ADR)

The ADR is a file-based repository for database diagnostic files data such as traces, dumps, the alert log, health monitor reports, and more. The location of ADR root directory (also called ADR base directory) is the value of the -

- DIAGNOSTIC_DEST initialization parameter (if this parameter is specified in SPFILE or PFILE).
- ORACLE_BASE, if DIAGNOSTIC_DEST is not set.

Inside ADR root, there is ADR home directory.

- The location of an ADR home is given by the following path, which starts at the ADR root directory: "*diag/product_type/product_id/instance_id*".
- For example, for a database with a SID and database name both equal to *iac*, and database type is *rdbms*, the ADR home would be in the following location: "*ADR_root/diag/rdbms/iac/iac/"*

Retrieve ADR location information by SQL Command

The V$DIAG_INFO view lists all important ADR locations for the current Oracle Database instance. A sample output is given below for a windows machine (the database name and instance name is *iac*).

```
SQL>SELECT * FROM V$DIAG_INFO;

INST_ID NAME                 VALUE
------- -------------------- ----------------------------------------------------------
      1  Diag Enabled         TRUE
      1  ADR Base             c:\app\oracle
      1  ADR Home             c:\app\oracle\diag\rdbms\iac\iac
      1  Diag Trace           c:\app\oracle\diag\rdbms\iac\iac\trace
      1  Diag Alert           c:\app\oracle\diag\rdbms\iac\iac\alert
      1  Diag Incident        c:\app\oracle\diag\rdbms\iac\iac\incident
      1  Diag Cdump           c:\app\oracle\diag\rdbms\iac\iac\cdump
      1  Health Monitor       c:\app\oracle\diag\rdbms\iac\iac\hm
      1  Default Trace File   c:\app\oracle\diag\rdbms\iac\iac\trace\orcl_ora_22769.trc
      1  Active Problem Count 8
      1  Active Incident Count 20
```

Table 3 describes some of the information displayed by the above command.

Name	**Description**
ADR Base	Path of ADR base
ADR Home	Path of ADR home for the current database instance
Diag Trace	Location of background process trace files, server process trace files, SQL trace files, and the text-formatted version of the alert log
Diag Alert	Location of the XML-formatted version of the alert log

Table 3: Meaning of V$DIAG_INFO outputs.

4.2 Alert Log File

The alert log is an XML file that is a chronological log of database *messages* and *errors*. This file captures errors and incidents related to database and user activity. It includes *messages* about the following:

- Startups and shutdowns of the instance
- Errors causing trace files.

- Create, alter and drop SQL statements on databases, tablespaces and rollback segments.
- Errors when a materialized view is refreshed.
- ORA-00600 (internal) errors.
- ORA-01578 errors (block corruption)
- ORA-00060 errors (deadlocks)

4.2.1 Location of the ALERT LOG file

Oracle will write the alert-log file to the directory as specified by the BACKGROUND_DUMP_DEST parameter. If this parameter is not set, the alert-log will be created in the ORACLE_HOME/rdbms/trace directory.

```
SQL> show parameter BACKGROUND_DUMP_DEST

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
background_dump_dest                 string      /app/oracle/diag/rdbms/o11gr1/o11gr1/trace
```

4.2.2 View the alert log with a text editor

Connect to the database with SQL*Plus or another query tool, such as SQL Developer. Query the V$DIAG_INFO view. Then you can do one of the following:

- *To view the text-only alert log*, without the XML tags, complete these steps: In the V$DIAG_INFO query results, note the path that corresponds to the *Diag Trace* entry, and change directory to that path. Open file alert_SID.log with a text editor.
- *To view the XML-formatted alert log*, complete these steps: In the V$DIAG_INFO query results, note the path that corresponds to the *Diag Alert* entry, and change directory to that path. Open the file log.xml with a text editor.

4.3 Oracle Trace File

An Oracle Trace file is written when one of the Oracle background processes encounter an exception. An ORA-00600 error encountered when Oracle detects an unexpected condition also produces a trace. Along with ORA-00600, a trace file is written in the trace directory. The name for the trace file written is recorded in the alert.log . The details of trace file are out of scope of this course.

Chapter 5: Oracle Data Dictionary Views

5 Oracle Data Dictionary

A data dictionary is a central component of every database. The database management software stores information about *metadata* in its data dictionary. Metadata is data about data, or data that defines other data. The Oracle data dictionary is metadata about the database. For example, if you create a table in Oracle, metadata about that table is stored in the data dictionary. Such things as column names, length, and other attributes are stored. Thus, the data dictionary contains a great volume of useful information about your database. Pretty much everything you would want to know about your database is contained in the data dictionary in some form.

The data dictionary is managed and updated by the Oracle software whenever a user of the database issues a data definition language statement such as CREATE, ALTER, DROP, TRUNCATE, etc, or a data control language statement such as GRANT or REVOKE (described in later chapters).

A data dictionary may contain information such as the following:

- The definitions of every schema object in the database, including default values for columns and integrity constraint information
- The amount of space allocated for and currently used by the schema objects
- The names of Oracle Database users, privileges and roles granted to users, and auditing information related to users

As a DBA then, you can see why the data dictionary is so important. Since you can't possibly remember everything about your database (like the names of all the tables and columns), Oracle remembers this for you. All you need to do is learn how to find that information.

5.1 Contents of data dictionary

The data dictionary consists of the following types of objects:

- **Base tables**: These underlying tables store information about the database. Only Oracle Database should write to and read these tables. Users rarely access the base tables directly because they are normalized and most data is stored in a cryptic format.

- **Views**: These views decode the base table data into useful information, such as user or table names, using joins and WHERE clauses to simplify the information. These views contain the names and description of all objects in the data dictionary. Some views are accessible to all database users, whereas others are intended for administrators only.

SYS, Owner of the Data Dictionary

The data dictionary is created when oracle database is created. The Oracle Database user SYS owns the data dictionary. No other Oracle Database user should ever alter (UPDATE, DELETE, or INSERT) any rows or schema objects contained in the SYS schema, because such activity can compromise data integrity. The security administrator must keep strict control of this central account.

Storage of the Data Dictionary

The data dictionary base tables are the first objects created in any Oracle database. All data dictionary tables and views for a database are stored in the SYSTEM tablespace. Because the SYSTEM tablespace is always online (i.e., active) when the database is open, the data dictionary is always available when the database is open.

How the Data Dictionary Is Used

The data dictionary has three primary uses:

- Oracle Database accesses the data dictionary to find information about users, schema objects, and storage structures.
- Oracle Database modifies the data dictionary every time that a data definition language (DDL) statement is issued.
- Any Oracle Database user can use the data dictionary as a read-only reference for information about the database.

5.2 How to Use the Data Dictionary

Use can query the data dictionary views with SQL statements. Some views are accessible to all Oracle Database users, and others are intended for database administrators only.

The name of some views start with three different prefixes as shown in Table 1.

Prefix	Example of view	Scope
USER_	USER_TABLES	User's view (what is in the user's schema)

ALL_	ALL_TABLES	Expanded user's view (what the user can access)
DBA_	DBA_TABLES	Database administrator's view (what is in all users' schemas)

Table 1: Some data dictionary view prefixes.

The USER_ view contains information relevant to the current user. It has information about a specific type of object that a user owns. For example, USER_TABLES contains information about the tables that the user querying the view, owns.

The ALL_ views contain information relevant to the current user. It has information about a specific type of object that a user has access to. For example, ALL_TABLES contains information about tables that a user querying the view has access to (i.e. owns and was given permission by another user).

The DBA_ views contain information about all the objects of the database, for example, DBA_TABLES, contains information about all the tables in the database.

Table 2 gives the names of some important DBA_ views required by database administrators frequently.

View Name	**Description**
DBA_TABLES	Describes information about all tables in database.
DBA_TABLESPACES	Describes information about tablespaces.
DBA_DATA_FILES	Describes information about data files.
DBA_VIEWS	Describes information about all views in database.
DBA_SYS_PRIVS	Describes the system privileges granted to user and roles.
DBA_TAB_PRIVS	Describes the object privileges granted to user and roles.
DBA_TRIGGERS	Describes information about all triggers in database.
DBA_USERS	Describes information about users in database.

Table 2: Some important DBA_ views

5.3 Dynamic Performance Tables and Views

Throughout its operation, Oracle Database maintains a set of virtual tables that record dynamic information of database. These views are called dynamic performance views (also called V$ views as their names start with the prefix V$). The information of these views is not permanent and populated each time data base is started.

You may remember that we have used such views in previous chapters (e.g., V$INSTANCE, V$CONTROLFILE, etc).V$ views contain information such as the following:

- System and session parameters
- Memory usage and allocation
- Database instance state and database information
- File states
- Progress of jobs and tasks
- Statistics and metrics

5.3.1 Data Population in the Dynamic Performance Views

The data of the dynamic performance views are dependent on the state of the database and instance. For example, you can query V$INSTANCE and V$BGPROCESS when the database is started. However, you cannot query V$DATAFILE until the database has been mounted.

The names of some important dynamic performance views are given in Table 3.

View Name	Information
V$ARCHIVE_DEST	Describes, for the current instance, all the archive log destinations, their current value, mode, and status.
V$CONTROLFILE	Lists the names of the controlfile.
V$DATABASE	Contains database information from the controlfile.
V$INSTANCE	Displays the state of the current instance.
V$DATAFILE	Contain information of data files of the current

	instance.
V$LOG_FILE	Stores information regarding database log files.

Table 3: Some important dynamic performance views

5.3.2 Some Examples of Using Dynamic Performance Views

To know the current startup state (e.g., started, mounted, open) of database instance, you can query the V$INSTANCE view as follows:

SQL> SELECT STATUS FROM V$INSTANCE;

To know the name of running instance (ORACLE_SID) of database, you can query the V$INSTANCE view as follows:

SQL> SELECT INSTANCE_NAME FROM V$INSTANCE;

To know whether database is in archived log mode or not, you can query the V$DATABASE view as follows (log modes are described in later chapters):

SQL> SELECT LOG_MODE FROM V$DATABASE;

To know the name of running database, you can query the V$DATABASE view as follows:

SQL> SELECT NAME FROM V$DATABASE;

Lap Practice 1: Execute the following two commands and understand their relation.

1. Create two separate sessions with the database. In the 1st session, login as SYSTEM user. In the 2nd session login as HR user. [**Take help from your instructor regarding HR account**]
2. In the HR session, view the tables that are owned by user HR. [**You need to query the USER_TABLES view**. **Take help from your instructor regarding the column you should print to obtain the required information**]
3. In the HR session, view the tables that can be accessed by user HR. [**You need to query the ALL_TABLES view**. **Take help from your instructor regarding the column you should print to obtain the required information**]

4. In the 1st session, execute the following two commands. The first command creates a table, T1 and the second command grants the selects access on table T1 to the user HR. Grants are described in later chapters in more detail.
   SQL> CREATE TABLE T1 (C1 NUMBER);
   SQL> GRANT SELECT ON T1 TO HR;
5. Do Step 2 and 3 again. Observe the difference of output this time and discuss with your instructor.
6. In both sessions, try to query the data dictionary view DBA_TABLES. Observe the output difference between the two sessions and discuss with your instructor.

Lab Practice 2: Find the following information from data dictionary and dynamic performance views. The required view names are given in parenthesis. You need to connect to the database with DBA privileges (SYS or SYSTEM by default are DBAs).

1. What is your database name? (V$DATABASE)
2. What is your instance name? (V$INSTANCE)
3. What data files and tablespaces are currently parts of your database?(DBA_DATA_FILES and DBA_TABLESPACES)
4. Determine the number, location and name of your redo log files (V$LOGFILE).
5. Determine the number, location and name of your archived redo log files, if any (V$ARCHIVED_LOG).
6. What mode is your database running in, NOARCHIVELOG or ARCHIVELOG? (V$DATABASE)
7. How many control files do you have? What are the location and name(s) of the file(s)? (V$CONTROLFILE)
8. What is the name of the temporary tablespace assigned to your username? (DBA_USERS)
9. How many temporary tablespaces have been created in your database? (DBA_TEMP_FILES)
10. Determine the values of the DB_BLOCK_SIZE and DB_CACHE_SIZE parameter (V$PARAMETER).
11. What users have SYSDBA privilege in your database? (V$PWFIlE_USERS)
12. What is the current value of the BACKGROUND_DUMP_DEST parameter?(V$PARAMETER)
13. What is the location of the alert log file and trace files? (V$DIAG_INFO)
14. Display the username and account status of all users on your database.(DBA_USERS)

Chapter 6: Oracle users and security

6 Oracle Users and Security

To perform the administrative tasks of an Oracle Database DBA, you need specific privileges within the database and possibly in the operating system of the server on which the database runs. Access to a database administrator's account should be tightly controlled.

6.1 The Database Administrator's Operating System Account

To perform many of the administrative duties for a database, you must be able to execute operating system commands. Depending on the operating system on which Oracle Database is running, you might need an operating system account or ID to gain access to the operating system. If so, your operating system account might require operating system privileges or access rights that other database users do not require (for example, to perform Oracle Database software installation). Although you do not need the Oracle Database files to be stored in your account, you should have access to them.

6.1.1 Administrative User Accounts of Oracle Database

In previous chapters, you should have used the administrator accounts SYS and SYSTEM. These two administrative user accounts are automatically created when Oracle Database is installed.

Super Powerful Administrative Privileges: SYSDBA and SYSOPER

Administrative privileges that are required for an administrator to perform basic database operations (i.e., shutting down or starting up a database) are granted through two special system privileges, SYSDBA and SYSOPER. You must have one of these privileges granted to you, depending upon the level of authorization you require. The SYSDBA and SYSOPER system privileges allow access to a database instance even when the database is not open. Control of these privileges is totally outside of the database itself.

Activating SYSDBA and SYSOPER Privilege

The unique thing about SYSDBA and SYSOPER privileges is that you must explicitly activate these privileges for your login session by specifying "AS SYSDBA" in the connection string. Otherwise, these

privileges do not become active even though you may have granted the privilege. The following command shows that SYS user is connecting with SYSDBA privilege activated:

SQL> CONNECT SYS AS SYSDBA

6.1.2 SYSDBA vs. SYSOPER

Table 1 shows the operations that are authorized by the SYSDBA and SYSOPER system privileges:

SYSDBA	• Perform STARTUP and SHUTDOWN operations • CREATE DATABASE • DROP DATABASE • ALTER DATABASE OPEN/MOUNT/BACKUP • CREATE SPFILE • ALTER DATABASE ARCHIVELOG • ALTER DATABASE RECOVER • Includes the RESTRICTED SESSION privilege
SYSOPER	• Perform STARTUP and SHUTDOWN operations • ALTER DATABASE OPEN/MOUNT/BACKUP • CREATE SPFILE • ALTER DATABASE ARCHIVELOG • ALTER DATABASE RECOVER • Includes the RESTRICTED SESSION privilege

Table 1: Difference of SYSDBA and SYSOPER privileges.

When you connect with SYSDBA or SYSOPER privileges, you connect with a default schema, not with the schema that is generally associated with your username. For SYSDBA this schema is SYS; for SYSOPER the schema is PUBLIC. Therefore, it is recommended that, you do not create any table while logged in “AS SYSDBA” because the created tables will reside in SYS schema.

By default SYS and SYSTEM accounts are given SYSDBA privilege. However, any other accounts can be granted these privileges by a special technique though oracle password file.

6.2 Oracle Password File

A password file enables login with SYSDBA privilege from remote machines. Otherwise, the notorious ORA-01031 error will be thrown, indicating "insufficient privileges." The Password File stores the passwords for users with administrative privileges, e.g, SYSDBA. The default location of the password file in windows is '%ORACLE_HOME%\database\PWD%ORACLE_SID%.ora'.

The following example shows that SYSTEM account is trying to login the database remotely (from a distant client) with SYSDBA privilege.

SQL> CONNECT SYSTEM@'102.10.5.15:1521/ORCL' AS SYSDBA

Without a password file, the above remote login with SYSDBA will fail. So, before allowing such remote login, we must create a password file in oracle database. Remember that, for local logins, SYSDBA can be enable via OS authentication (in case OS user account is an administrator account) or password file authentication (in case OS user is not administrator, therefore, he has only password-file option for SYSDBA login). However, for remote non-secure connections (via ORACLE NET), password file is the only option for SYSDBA login.

The following three steps show how to create a password file, how to enable remote login as SYSDBA, and how to grant SYSDBA access to a certain user:

Step 1: If not already created, create the password file using the ORAPWD utility:

CMD> ORAPWD FILE='%ORACLE_HOME%\DATABASE\PWDORCL.ORA ENTRIES=25

The above command will create a password file in the default location. The command means the following:

- The password file is created in default password file directory. In windows, this directory is '%ORACLE_HOME%\DATABASE'
- The name of the password file is PWDORCL.ORA.
- At most 25 users can be granted SYSDBA or SYSOPER privilege. This is maximum number of entries that can be stored in password file.

Step 2: If not set already, you need to set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to EXCLUSIVE (In SPFILE or PFILE). To do this for SPFILE, execute the following command:

SQL> ALTER SYSTEM SET REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE SCOPE=SPFILE;

REMOTE_LOGIN_PASSWORDFILE is a static initialization parameter and therefore cannot be changed without restarting the database. **You must restart the database to make this change into effect**.

Step 3: Grant SYSDBA or SYSOPER to HR user as follows:

SQL> GRANT DBA, SYSDBA to HR

The above command grants SYSDBA to HR. Since, password file has been created, this command will be successful and HR user can login to the oracle database as SYSDBA remotely.

6.2.1 Connecting Using Password File Authentication

Administrative users can be connected and authenticated to a local or remote database by using the SQL*Plus CONNECT command. They must connect using their username and password and the AS SYSDBA or AS SYSOPER clause. For example, user HR has been granted the SYSDBA privilege, so HR can connect as follows:

SQL> CONNECT HR@ORCL AS SYSDBA

Here, ORCL is the net service name (discussed in later chapters). It should be used here for verifying SYSDBA login if you are using local machine oracle server.

Note that, *operating system authentication always takes precedence over password file authentication for SYSDBA login*. Therefore, if you are a member of the OSDBA (ORA_DBA in windows) or OSOPER group for the operating system, and you connect as SYSDBA or SYSOPER, you will be connected with associated administrative privileges regardless of the username/password that you specify.

For example, assume that your OS account is a member of ORA_DBA group (in windows) and JOHN account in database has not been granted SYSDBA. Now, you gave the following command for local login:

>CONNECT JOHN AS SYSDBA

Even though, JOHN has no SYSDBA privilege, the above login will pass. Because, OS authentication will be tried first and that will succeed due to OS privilege. JOHN account and its password will not be checked at all!

6.2.2 Verify who has the SYSDBA privilege in Database

You can query the data dictionary view V$PWFILE_USERS to know which users in the database have SYSDBA or SYSOPER privilege. The following command illustrates this:

```
SQL> SELECT * FROM V$PWFILE_USERS;

USERNAME                       SYSDB      SYSOP
------------------------------ ---------- ---------
SYS                            TRUE       TRUE
```

Lab Activity 1: Practice password file and SYSDBA login.

1. Login to Oracle as SYS user with the SYSDBA privilege.
2. Login to Oracle as SYSTEM user with the SYSDBA privilege.
3. Execute the SHOW USER command. What user is returned? [**Discuss with your instructor about the output**]
4. If you are working with a local connection to your Oracle server, login to the database with a local operating system connection. If problems occur, check which OS users are parts of the ORA_DBA OS group. [**If you have only a remote connection, connect as necessary and go on to the Step 6. If you have a local connection, then go on to Step 5 below or use net service name in all connect statements in Step 6-13**]
5. Disable OS authentication (Search the SQLNET.ORA file, open the file, and set the parameter SQLNET.AUTHENTICATION_SERVICES=NONE if otherwise)
6. Create a password file if none exists already in your database.
7. Determine which users have been granted SYSDBA.
8. By default, HR account lacks SYSDBA privilege. So, grant SYSDBA to the HR account.
9. Attempt to connect as HR user without the SYSDBA privilege. What happens?
10. Attempt to connect as HR user with the SYSDBA privilege. What happens?
11. Grant the DBA role to the HR user.

12. Connect to the database as HR user – without SYSDBA.
13. Connect to the database as HR user – with SYSDBA.

6.3 Creating and Managing Users, Roles, and Privileges

A user account is required to connect to the database. When we create a database, some default user accounts are created, e.g., SYSTEM, SYS, etc. However, as applications grow, we need to create the new end users who will be using the database on a day-to-day basis.

6.3.1 Temporary and Default Tablespaces

All users need a TEMPORARY TABLESPACE where they can perform work such as sorting data during SQL execution. Users also need to have a DEFAULT TABLESPACE, where objects (e.g., tables, views) created by the user will be stored. If you don't assign a specific tablespace as the DEFAULT TABLESPACE, the *database default tablespace* becomes your DEFAULT TABLESPACE. If there is no database default, then SYSTEM becomes the default tablespace (this is not recommended at all).

6.3.2 Creating a New User

You use the CREATE USER SQL statement to create a new user. The following example creates a new user JOHN whose password is JOHNPASS. In oracle 11g, passwords can be case sensitive, so be careful when you give the password.

```
SQL> CREATE USER JOHN IDENTIFIED BY JOHNPASS;
        User created.
SQL>
```

Since, we did not specify default and temporary tablespaces in the above CREATE USER SQL command, user JOHN will be assigned *database default temporary* and *default permanent tablespace*. All database users are recorded in the data dictionary. You can query the DBA_USERS view for a complete list of all users. The following query shows the new user's default (permanent) and temporary tablespaces:

```
SQL> SELECT DEFAULT_TABLESPACE, TEMPORARY_TABLESPACE FROM DBA_USERS
WHERE USERNAME='JOHN';
```

```
DEFAULT_TABLESPACE      TEMPORARY_TABLESPACE
------------------      ------------------------
USERS                   TEMPTBS_01
>
```

The new user can't connect to the database, however, because the user doesn't have any privileges to do so. This is what happens when the user JOHN tries to connect using SQL*Plus:

```
SQL>  SQLPLUS JOHN/JOHNPASS

SQL*Plus: Release 11.1.0.6.0 - Production on Fri Mar 21 11:55:38 2008
Copyright (c) 1982, 2007, Oracle. All rights reserved. ERROR:
Ora-01045: user JOHN lacks CREATE SESSION privilege; logon denied
Enter user-name:
```

In order for the JOHN user to connect and start communicating with the database, you must grant the CREATE SESSION system privilege to the new user, as shown below:

```
SQL> GRANT CREATE SESSION TO JOHN;
Grant succeeded.
SQL>
```

It's always advisable to explicitly assign each new user a default tablespace and a temporary tablespace. The following CREATE USER command shows how to do that for user JOHN:

```
 SQL> CREATE USER JOHN IDENTIFIED BY JOHNPASS
    DEFAULT TABLESPACE USERS
    TEMPORARY TABLESPACE TEMP;

User created.
SQL>
```

In the above command, we assume that, database has a permanent tablespace named USERS and a temporary tablespace named TEMP.

You must also explicitly allocate tablespace quotas to a user. The quotas determine the maximum space that the user is allowed to use on the corresponding tablespace. The following example shows how you can assign specific quota values for user JOHN on USERS tablespace.

```
SQL>   CREATE USER JOHN IDENTIFIED BY JOHNPASS
    DEFAULT TABLESPACE USERS
    TEMPORARY TABLESPACE TEMP
    QUOTA 10M ON USERS;

User created.
SQL>
```

You can see the individual tablespace quotas allocated to a user by using the DBA_TS_QUOTAS view, as shown in the following example:

```
SQL> SELECT TABLESPACE_NAME, USERNAME, BYTES FROM DBA_TS_QUOTAS;

TABLESPACE USERNAME    BYTES
---------- --------- ----------
SYSAUX     DMSYS      196608
SYSAUX     OLAPSYS    16252928
SYSAUX     WK_TEST    12582912
SYSAUX     SYSMAN     78577664
RMAN_TBSP  RMAN 8585216
SQL>
```

6.3.3 Altering a User (ALTER USER)

The ALTER USER statement can be used to change the attributes of a user. Using this statement, you can do the following:

- Change a user's password.
- Assign and modify tablespace quotas.
- Set and alter default and temporary tablespaces.

- Assign and modify a profile and default roles.

Below, we show user ALTER USER command can be used to change previously created JOHN's password. The new password will be JOHNNEWPASS

```
SQL> ALTER USER JOHN IDENTIFIED BY JOHNEWPASS;

User altered.

SQL>
```

Only a DBA can change passwords of other users with the ALTER USER statement. Users can also change their own passwords with the PASSWORD command in SQL*Plus, as shown here:

```
SQL> PASSWORD
     Changing password for AHMED
     Old password: *********
     New password: *********
     Retype new password: *********
     Password changed
SQL>
```

The following ALTER USER command modifies previously created JOHN user's quota on USERS tablespace and the new quota is set to be UNLIMITED (no bound on space usage).

```
SQL> ALTER USER JOHN QUOTA UNLIMITED ON USERS;

User altered.
SQL>
```

6.3.4 Dropping a User

To drop a user, you use the DROP USER statement, as shown here:

```
SQL> DROP USER JOHN;

User Dropped.
SQL>
```

The DROP USER statement will remove just the user from the database, but all the objects owned by the user will remain intact. If other objects in the database depend on this user, you won't be able to use the simple DROP USER command—you must use the DROP USER . . . CASCADE statement, which drops the user, the user's schema objects, and any dependent objects as well. Here's an example:

```
SQL> DROP USER JOHN CASCADE;

User Dropped.
SQL>
```

If you aren't sure whether you will need a user's objects later, but you want to deny access, simply leave the user and the user's schema intact, but deny the user access to the database by using the following statement:

```
SQL> REVOKE CREATE SESSION FROM JOHN;

Revoke succeeded.
SQL>
```

Lab Activity 1: Working with users.

1. Create an Oracle user account with the following settings :
    a. User Name - yourname2 e.g. geoff2
    b. Password - yourname e.g. geoff
    c. Default tablespace - USERS
    d. Temporary tablespace - TEMP

2. Query the Oracle data dictionary DBA_USERS to ensure that this user has been created correctly.
3. Attempt to make a connection to the database using the user name and password specified.
4. Any ideas as to why (3) does not work? Fix the problem and re-connect. (GRANT CREATE SESSION)
5. Change the user's password to yourname2 e.g. geoff2 .
6. Query the USER_USERS data dictionary view while logged into the new user.

6.3.5 Granting Roles and Privileges

DBA need to assign specific privileges to each created user. These privileges determine what the user can do in the database. Previously, we showed that, a created user cannot login until he is given the CREATE SESSION privilege by the DBA. Similarly, users need to have CREATE TABLE privilege to create tables on his default tablespace. Privileges are granted with the GRANT statement. The following example shows how you can do this:

```
SQL> GRANT CREATE TABLE TO JOHN;
Grant succeeded.
SQL>
```

Oracle provides two types of privileges:

1. **System privileges** – These privileges allow the user to execute some action on the database. For example, logon requires CREATE SESSION privilege; creating a table requires CREATE TABLE privilege. System privileges are not tied to one specific object.
2. **Object privileges** – These privileges are related to one specific database object. Using these privileges, users can be granted to edit, modify any specific table, view, etc.

Previously, we showed how system privilege (e.g., CREATE SESSION) was granted to user JOHN. Now, we will show how an object privilege is granted to user JOHN. Usually, one user cannot access data (table, view, etc) of another user. But, object privilege can be used to allow this type of access. The following example shows that user JOHN has been granted select and updates privilege on another user's (HR) table (EMPLOYEE).

```
SQL> GRANT SELECT, UPDATE ON HR.EMPLOYEE TO JOHN;
Grant succeeded.
```

SQL>

The following data dictionary will give you information about user's system priviliges and roles (described later)

- DBA_SYS_PRIVS - Records all system privileges granted to all users and all roles.
- USER_SYS_PRIVS - System privileges granted to current user
- ROLE_SYS_PRIVS - System privileges granted to roles

Table 1 shows some minimal important privileges that a user should have to work in a database.

System Privilege	Description
CREATE SESSION	To connect to the database.
CREATE TABLE	To create tables on users own default tablespace.
CREATE VIEW	To create views on users' own default tablespace.
CREATE SYNONYM	To create synonyms.
CREATE PROCEDURE	To create procedures, functions, etc.
CREATE TRIGGER	To create database triggers.

Table 1: Some minimal system privileges required for a user to work in database.

Lab Activity 2: Working with privileges.

1. Logon to the JOHN user you created.
2. Create a table "T" with one column "NAME".
3. Login to database with DBA account (e.g., SYSTEM).
4. Create a new user ALICE with password ALICE.
5. Grant all DDL privileges (SELECT, INSERT, DELETE, UPDATE) on JOHN's "T" table to user ALICE
6. Query the data dictionary to determine all recipients of object privileges on your table.(**You can take help from your instructor**)
7. Login to database with user ALICE.
8. Insert some data in "T" table.

9. Re-connect with JOHN and verify inserted data by ALICE.

6.3.6 Working with ROLES

A role is a collection of privileges. Once an Oracle user has been granted the role, all the privileges inherent in the role are automatically available to the user. Creating a role is done with the CREATE ROLE statement, which requires the CREATE ROLE privilege. Just like granting a privilege to a user, we can use the GRANT statement to grant privileges to a role. Then, the role can be granted to a user. The user acquires all privileges within the role upon next logon. Below, we show the steps to create a role, assign privileges to the role, and then granting the role to a user.

```
SQL> CREATE ROLE APP;
Role created.
SQL>
```

Now, grant privileges to role APP.

```
SQL > GRANT CREATE SESSION TO APP;
Grant succeeded.
SQL > GRANT CREATE TABLE TO APP;
Grant succeeded.
SQL > GRANT CREATE VIEW TO APP;
Grant succeeded.
SQL >
```

Now, you can grant the role APP to user JOHN. User JOHN will then receive all privileges assigned to the role APP.

```
SQL > GRANT APP TO JOHN;
Grant succeeded.
SQL >
```

After database creation, there are some default roles such as DBA, CONNECT and RESOURCE. The DBA role contains all required privileges an administrator account should have. So, you can create an administrator account by granting this role to a user. The following examples grants DBA role to user JOHN making JOHN an administrator of database.

```
SQL > GRANT DBA TO JOHN;
Grant succeeded.
SQL >
```

Lab Activity 3: Working with roles.

1. Create a role called ‘MANAGER' and assign that role the ability to be able to create tables and synonyms.
2. Create another role called ‘SALESPERSON'. The ‘SALESPERSON' role should be given the ability to be able to make connections to the database.
3. Assign the ‘SALESPERSON' role to the ‘MANAGER' role since the manager will require all the privileges given to the ‘SALESPERSON' role.
4. Give both roles to the JOHN Oracle user (create the JOHN user if requires).
5. Connect as the JOHN user and determine which roles are present. (Query DBA_ROLES, ROLE_SYS_PRIVS, DBA_ROLE_PRIVS data dictionary views)

6.4 Creating and Using User Profiles

You can set the individual resource limits in Oracle by using what are known as profiles. A profile is a collection of resource-usage and password-related attributes that you can assign to a user. Multiple users can share the same profile, and you can have an unlimited number of profiles in an Oracle database. Profiles set hard limits on resource consumption by the various users in the database and help you limit the number of sessions a user can simultaneously keep open, the length of time these sessions can be maintained, and the usage of CPU and other resources.

```
SQL>  CREATE PROFILE MISER
   LIMIT
   CONNECT_TIME 120
   FAILED_LOGIN_ATTEMPTS 3
   IDLE_TIME 60
   SESSIONS_PER_USER 2;
```

```
Profile created.
>
```

When a user with the miser profile connects, the database will allow the connection to be maintained for a maximum of 120 seconds and will log the user out if he or she is idle for more than 60 seconds. The user is limited to two sessions at any one time. If the user fails to log in within three attempts, the user's accounts will be locked for a specified period or until the DBA manually unlocks them.

6.4.1 Parameters and Limits

Oracle databases enable you to set limits on several parameters within a profile. The following sections provide brief explanations of these parameters, which can be divided into two broad types: resource parameters, which are concerned purely with limiting resource usage, and password parameters, used for enforcing password-related security policies.

Resource Parameters

The main reason for using resource parameters is to ensure that a single user or a set of users doesn't monopolize the database and server resources. Here are the most important resource parameters that you can set within an Oracle Database 11g database:

- CONNECT_TIME: Specifies the total time (in minutes) a session may remain connected to the database.
- CPU_PER_CALL: Limits the CPU time used per each call within a transaction (for the parse, execute, and fetch operations).
- CPU_PER_SESSION: Limits the total CPU time used during a session.
- SESSIONS_PER_USER: Specifies the maximum number of concurrent sessions that can be opened by the user.
- IDLE_TIME: Limits the amount of time a session is idle (which is when nothing is running on its behalf).
- LOGICAL_READS_PER_SESSION: Limits the total number of data blocks read (from the SGA memory area plus disk reads).

- LOGICAL_READS_PER_CALL: Limits the total logical reads per each session call (parse, execute, and fetch).
- PRIVATE_SGA: Specifies a session's limits on the space it allocated in the shared pool component of the SGA (applicable only to shared server architecture systems).
- COMPOSITE_LIMIT: Sets an overall limit on resource use. A composite limit is a limit on the sum of several of the previously described resource parameters, measured in service units. These resources are weighted by their importance. Oracle takes into account four parameters to compute a weighted

COMPOSITE_LIMIT: CPU_PER_SESSION, CONNECT_TIME, LOGICAL_READS_PER_SESSION, and PRIVATE_SGA. You can set a weight for each of these four parameters by using the ALTER RESOURCE COST statement, as shown in the following example:

```
SQL > ALTER RESOURCE COST
  CPU_PER_SESSION 200
  CONNECT_TIME 2;

Resource cost altered.
SQL >
```

Password Parameters

Oracle provides you with a wide variety of parameters to manage user passwords. You can set the following password-related profile parameters to enforce your security policies:

- FAILED_LOGIN_ATTEMPTS: Specifies the number of consecutive login attempts a user can make before being locked out.
- PASSWORD_LIFE_TIME: Sets the time limit for using a particular password. If you don't change the password within this specified time, the password expires.
- PASSWORD_GRACE_TIME: Sets the time period during which you'll be warned that your password has expired. After the grace period is exhausted, you can't connect to the database with that password.

- PASSWORD_LOCK_TIME: Specifies the number of days a user will be locked out after reaching the maximum number of unsuccessful login attempts.
- PASSWORD_REUSE_TIME: Specifies the number of days that must pass before you can reuse the same password.
- PASSWORD_REUSE_MAX: Determines how many times you need to change your password before you can reuse a particular password.
- PASSWORD_VERIFY_FUNCTION: Lets you specify an Oracle-provided password-verification function to set up an automatic password-verification mechanism.

The Default Profile

If you create a user and don't explicitly assign any profile to the user, the user will inherit the default profile, as shown here:

```
SQL> SELECT profile FROM dba_users
        WHERE username = 'AHMED'
        PROFILE
        -----------
        DEFAULT
```

The default profile, unfortunately, isn't very limiting at all—virtually all the resource limits are set to UNLIMITED, meaning there's no limit on resource usage whatsoever.

```
SQL> SELECT DISTINCT resource_name, limit
        FROM dba_profiles
        WHERE profile='DEFAULT';
        RESOURCE_NAME    LIMIT
        -------------------------------        ----------
        PASSWORD_LOCK_TIME     1
        IDLE_TIME     UNLIMITED
        CONNECT_TIME       UNLIMITED
        PASSWORD_GRACE_TIME   7
        LOGICAL_READS_PER_SESSION     UNLIMITED
```

```
        PRIVATE_SGA          UNLIMITED
        LOGICAL_READS_PER_CALL          UNLIMITED
        SESSIONS_PER_USER          UNLIMITED
        CPU_PER_SESSION    UNLIMITED
        FAILED_LOGIN_ATTEMPTS 10
        PASSWORD_LIFE_TIME       180
        PASSWORD_VERIFY_FUNCTION     NULL
        PASSWORD_REUSE_TIME    UNLIMITED
        PASSWORD_REUSE_MAX    UNLIMITED
        COMPOSITE_LIMIT    UNLIMITED
        CPU_PER_CALL         UNLIMITED
        16 rows selected.
SQL>
```

6.4.2 Assigning a User Profile

You can assign a user a profile when you create the user. Here's an example:

```
SQL> CREATE USER ahmed IDENTIFIED BY sammyy1
        TEMPORARY TABLESPACE TEMPTBS01
        DEFAULT TABLESPACE USERS
        GRANT QUOTA 500M ON USERS;
        PROFILE 'prod_user';
        User created.
SQL>
```

You can also assign a profile to a user any time by using the ALTER USER statement, as shown here:

```
SQL> ALTER USER ahmed
        PROFILE test;
        User altered.
SQL>
```

You can use the ALTER USER statement to assign an initial profile or to replace the current profile with another.

Altering a User Profile

You can alter a profile by using the ALTER PROFILE statement, as follows:

```
SQL> ALTER PROFILE test
        LIMIT
        sessions_per_user 4
        failed_login_attempts 4;
        Profile altered.
SQL>
```

Password Management Function

You can use the Oracle provided script named utlpwdmg.sql (from the $ORACLE_HOME/rdbms/admin directory) to implement various password management features such as setting the default password resource limits. This script lets you create a password verification function named verify_function_11g. The verify_function_11g function helps you implement password complexity in your database and lets you customize it for ensuring complex password checking. The verify_function_11g function lets you implement password protection features such as the following:

- Ensuring that all passwords are at least eight characters long
- Ensuring that every password contains at least one number and one alphabetic character
- Ensuring that a password differs from the previous password by at least three characters
- Checking to make sure that passwords aren't simply a reverse of the usernames
- Checking to make sure the passwords aren't the same as or similar to the name of the server
- Checking to make sure that a password isn't in a set of well-known and common passwords such as "welcome1" or "database1"

```
SQL >ALTER PROFILE DEFAULT LIMITPASSWORD_LIFE_TIME 180
        PASSWORD_GRACE_TIME 7
        PASSWORD_REUSE_TIME UNLIMITED
        PASSWORD_REUSE_MAX UNLIMITED
        FAILED_LOGIN_ATTEMPTS 10
```

PASSWORD_LOCK_TIME 1

PASSWORD_VERIFY_FUNCTION verify_function_11G;

Dropping a User Profile

Dropping a profile is straightforward. Here's how you would drop the test profile:

```
SQL> DROP PROFILE test CASCADE;
```

Profile dropped.

```
SQL>
```

Chapter 7: Oracle Network Configuration - Server

7 Oracle Networking Concepts - Server

Nearly every Oracle database is maintained on a network. The database is located on what we would call as the Oracle Server, and the applications are maintained on client machines. The client machines access the remote database using the network. Typically the Oracle server is maintained remotely from the client.

Oracle Net Software

Oracle Net software must be installed correctly for network connections. It uses a special protocol named "transparent network substrate" (TNS) protocol. Oracle Net software interfaces between the database and the network protocol, for example TCP/IP. So the database itself does not and is not capable of directly communicating with the network protocol - that is the job of Oracle Net Software.

Oracle Net is the new name for the Oracle networking software. It started as SQL*Net (many people still call it that), then Net8 and now Oracle Net.

7.1 Listener Configuration

On the server-side, the most important component is the Listener process. This process, which by default is called LISTENER, should be running, in order to listen for incoming client connection requests. When a client tries to connect to the database from a remote machine, the client will sent details such as the username, password of the schema along with the details pertaining to listener. The listener process on the other hand is on the server, and listens for requests on behalf of specific databases. Databases must register themselves with the listener upon their startup. If the listener recognizes the database that the client is trying to connect to, the connection will be established and the client will be able to communicate further with the database.

7.1.1 Listener Behaviors upon Client Connection Request

Upon receiving a connection request from a client, the Listener can do one of two things.

- Either it will spawn a new Server Process, and redirect the client to talk directly to that Server Process at which point, the Listener drops out of the picture altogether, and continues to listen for

connection requests from other clients. This is known as 'bequeathing' the Server Process to the client, in the sense of 'making a gift' –and the client is then said to have a Bequeath Session.

- Or it will inform the client of the network address of a Server Process which has already been created when the Instance was started (a "pre-spawned Server Process), and the client is then able to make direct contact with that Server Process. Note again, however, that once the connection is established between the client and the Server Process, the Listener simply continues to listen for new connection requests. This is known as 'redirecting' the client to the Server Process, and hence the client is said to have a Redirect Session

7.1.2 Characteristics of listener

Every listener has the following characteristics:

- Has a name
- Has a associated port number
- Configured to understand specific protocols (their protocol address)
- Database services that they listen on behalf of

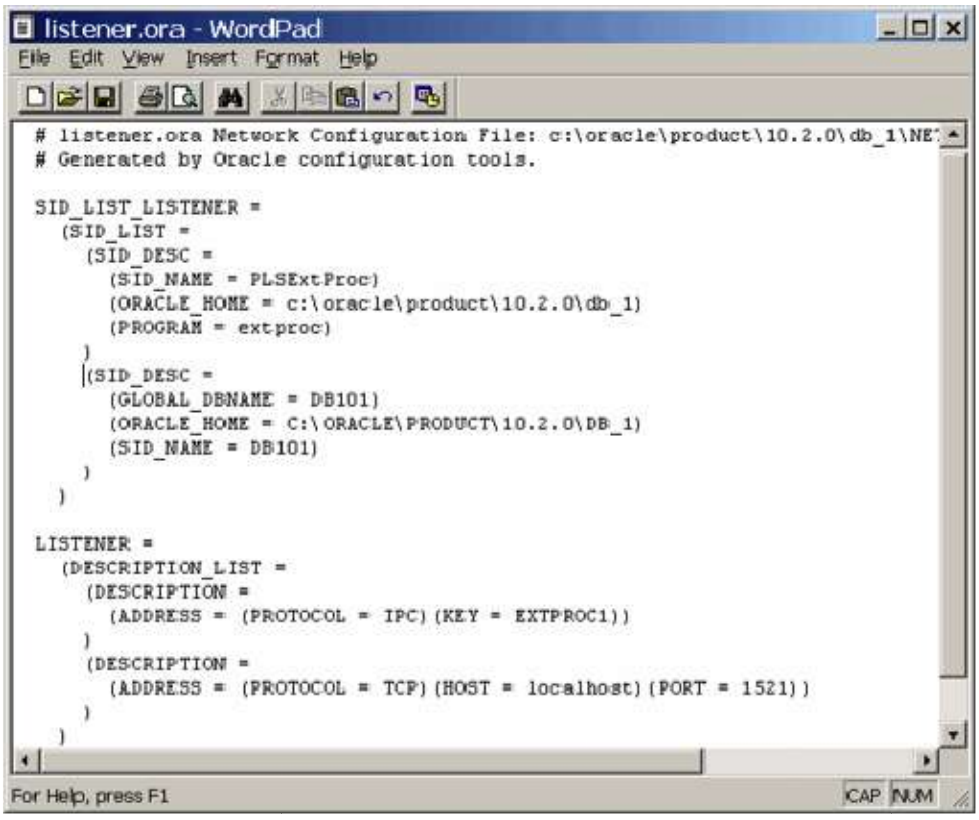

Figure 1: A typical listener.ora file.

7.1.3 LISTENER.ORA file

The listener is configured using a file called the LISTENER.ORA file. This file is located along with other network related files, in the location specified by the TNS_ADMIN environment variable, whose default value is $ORACLE_HOME/network/admin. You can edit this file to configure listeners.

A listener is configured with one or more listening protocol addresses, information about supported services, and parameters that control its runtime behavior. This configuration is stored in listener.ora. Because all of the configuration parameters have default values, it is possible to start and use a listener with no configuration. This default listener has a name of LISTENER, supports no services upon startup, and listens on the following TCP/IP protocol address:

```
(ADDRESS=(PROTOCOL=tcp)(HOST=host_name)(PORT=1521))
```

Figure 1 shows a typical Listener.ora file. Although, it is for an Oracle 10g database, the Oracle 11g listener.ora file would look similar. The name of the listener is LISTENER, it is configured to listen on behalf of external procedures as well as for a database called DB101. It is listener on a host machine called LOCALHOST, at the port 1521.

7.1.4 Services Supported by a listener process

Supported services, that is, the services to which the listener forwards client requests, can be configured in the listener.ora file or this information can be dynamically registered with the listener. This dynamic registration feature is called *service registration* and is used by newer oracle versions. The registration is performed by the PMON process--an instance background process--of each database instance that has the necessary configuration in the database initialization parameter file. **Dynamic service registration does not require any configuration of service names in the listener.ora file.**

7.2 Use of Listener Configuration Tools

The tools that can be used to configure a listener - in terms of creating the listener, setting / modifying its parameters can be done using:

- The Listener Control Utility (LSNRCTL) - (command line tool)
- Network Configuration Assistant (NETCA) - (graphical tool)

- Network Manager (NETMGR) - (graphical tool)
- Enterprise Manager (browser based graphical interface)

We will now perform some tasks related to listeners, using the graphical tool called *netmgr*.

7.2.1 Task 1: Creating a new listener

1. Run the *Net Manager* tool.

2. A navigation tree, will be displayed, from which Select LOCAL -> LISTENERS

3. To create a new listener called LISTENER1, from the top menu select:

Edit -> Create ->

4. When prompted for the name of the listener, type LISTENER1

5. Now enter the protocol address details as shown below. You might have to choose an appropriate port address.

Add Address button → Protocol: TCP/IP

Host name: LOCALHOST

Port Number: 1681

6. To save the configuration settings, from the top menu select:

File ->Save Network Configuration

7.2.2 Task 2: Add Database Service to LISTENER1

We will now view and enable some attributes for the LISTENER1.

1. From the navigation tree, Select LISTENER1
2. On the right side, from the drop down select: Database Services
3. Add your running database (e.g., ORCL) to the service list of LISTENER1
4. File -> Save Network Configuration

Opening the listener.ora file

You can see the details of the changes you have made by looking at the listener.ora file.

1. Go to the OS directory: '%ORACLE_HOME%\network\admin'

2. See the entries for LISTENER1 and verify your configurations such as, address, port no., database service, etc.

7.2.3 Task 3: Stop current listener and start newly created LISTENER1

This will be done using the listener control utility LSNRCTL command line tool.

The Listener Control Utility (lsnrctl)

We will now use the command line tool called Listener Control to manage the listener.

From a command prompt issue the command:

CMD> lsnrctl

The LSNRCTL prompt will be displayed. From here you can type commands that would affect a specific listener. If you don't specifically identify the listener, the default listener called LISTENER will be affected.

1. To see the status of the current listener:

LSNRCTL> status – (If protocol errors are displayed the listener has not been started).

2. To see the name of the current listener:

LSNRCTL> show current_listener

3. To stop the current listener:

LSNRCTL> stop

4. To change the current listener to LISTENER1:

LSNRCTL> set current_listener LISTENER1

5. To see the status of the listener:

LSNRCTL> status

6. If you see an error, indicating it has not been started, start it:

LSNRCTL> start

7. The listener will be started. To see the database services, the listener responds requests to, issue the following command

LSNRCTL> services

If you don’t see any service supported by the listener at this time, restart the database or restart the listener again.

8. Exit the listener control utility.

LSNRCTL> exit

Chapter 8: Oracle Network Configuration - Client

8 Oracle Networking Concepts - Client

8.1 Client-side configuration

When a client tries to connect to a remote database, in addition to the username and password, the client will also need to mention a service name. This is an alias for the protocol address of the listener which is on the server, listening on behalf of the database. There are a number of different methods available for client-side connectivity. The ones that will be discussed in this article are the Easy Connect Method and the Local Naming Method.

8.2 Easy Connect Method

In the Easy Connect Method, all the information necessary to connect to is mentioned as part of the connect request itself. There is no configuration necessary. A typical easy connect request would contain:

SQL> SQLPLUS /nolog

SQL> CONNECT username/password@'host[:port][/service_name]'

For example:

SQL> CONNECT hr/hr@'www.dbserver.com:1521/mydb'

In the example above:

- Schema you are connecting to be *hr* and the password is *hr*.
- The host name of the machine on the network is: *www.dbserver.com*
- The port on which the listener is listening is: 1521
- The database service/instance you are connecting to is: *mydb*

As you can see, the easy connect naming method requires no configuration in client side.

The other method available is called the Local Naming Method. This method needs some amount of configuration.

8.3 Local Naming Method

If you are using the local naming method, a typical connection request from a client tool such as SQL*Plus would be:

SQL> CONNECT hr/hr@xyz

Now the xyz, is what would be the "Net Service Name". Although it is not typically called xyz, the name can be anything that you created for the service name. The xyz resolves into the details of the listener that listens at a specific port on a specific host machine (on the network), on behalf of a database.

For example, the xyz would have stand for information that looks like:

xyz =

(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=SALESSERVER)(PORT=1521))

(CONNECT_DATA=(SERVICE_NAME=sales1)

(INSTANCE_NAME=sales1)))

This is what all the information means:

- xyz=> alias
- Protocol => TCP protocol
- Host machine name on which listener is listening => SALESSERVER
- Port on which the listener is listening => 1521
- Service Name => The name of the database service
- Instance Name => The name of the database instance

8.4 Use of tools for client side configuration

As a database administration, if you were to configure the client for database connectivity, knowing how to create a net service name such as XYZ will be necessary. The net service name contains all the details necessary to indicate the protocol being used, the address of the remote database host, the listener port number and the database you wish to connect to.

To configure such an alias the tools you can use are:

- Net Manager (graphical tool)
- Network Configuration Assistant (graphical tool)

- Enterprise Manager (browser based graphical tool)
- Manually editing a file called the tnsnames.ora file (not recommended).

Any changes you make using a tool is reflected in the **tnsnames.ora** file. The file is located with other Oracle network-related files in the location defined by the $TNS_ADMIN environment variable, whose default value is '%ORACLE_HOME\network\admin'.

We will now see how to create a net service name called MYDB using network configuration assistant and view the changes in the tnsnames.ora file. We will also configure the sqlnet.ora file to use the local naming method as the preferred method for name resolution.

8.4.1 Configuring the CLIENT side for Local Naming

1. Run the Network configuration assistant utility (NETCA).
2. In the window, select "Local Net service name configuration"
3. Then select "ADD"
4. Type: ORCL (this is the name of the database service you want to connect to, change it according to your configuration)
5. Select the Protocol for the client: TCP
6. Against HOSTNAME type LOCALHOST (this is the name of the host machine on which the listener is listening)
7. Choose: PORT 1521 (this is the port at which the listener is configured to listen)
8. Perform a test, if unsuccessful, Change Login and make sure you specify the current password for the SYSTEM user. (By default the test connection is tried using the username SYSTEM with a password of MANAGER).
9. Then give a Net Service Name: Type: MYDB (this is the name of the net service you are trying to create)

Viewing the TNSNAMES.ORA file and using the net service name

1. Go to '%ORACLE_HOME\network\admin'
2. Open the tnsnames.ora file and make sure you see and entry for a network alias called MYDB.

It might look something like:

mydb=

(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=localhost)(PORT=1521))

(CONNECT_DATA=(SERVICE_NAME=ORCL)(INSTANCE_NAME=ORCL)))

You can now open SQL*Plus and try to connect as the user hr using the net alias mydb.
SQL> CONNECT hr/hr@mydb

The above should work successfully to log you in the database using your net service name, i.e., mydb.

Practice Lab 1: Configuring manually TNSNAMES.ORA file.

1. Backup your TNSNAMES.ORA file
2. Create a new connect descriptor in your local TNSNAMES.ORA file. The net service name should be "CLASS" and it should connect you to the database you have been using in your machine.
3. Test by using the descriptor name in a connect string (e.g. CONNECT DAVE/DAVE@CLASS). Make sure it works!
4. Check the status of the listener on your machine.
5. Stop the listener using LSNRCTL utility.
6. Attempt connections using the TNSNAMES entry created in step 2. Does the connection work? Why?
7. Re-Start the listener.
8. Insure that you can connect to the database.
9. Restore your original TNSNAMES.ORA file.

Chapter 9: Export Import

9 Data Pump Export Import

Data Pump is a server-based data export and import utility. Data pump includes two important utilities: export (*expdp*) and import (*impdp*). Exports are used to take logical backups of database objects. They are called "logical" backups, because logical database objects such as a table are backed up. Export writes output to a binary file (often called a dump file), which can be read by the Import utility.

The Export utility (*expdp*) creates a binary dump file containing the object definitions (DDL) and data. The file is in an Oracle proprietary format; the Import utility (*impdp*) must be used to restore this file. Export utility is run from operating system command line by executing *expdp* command (along with some parameters described below).

The following two steps (10.1 and 10.2) are required to work with export (and import) utility.

9.1 Creating Directory and Granting Permissions, if Necessary.

Export (and import) utility requires you to create an Oracle Directory for the datafiles and log files it will create and read. This directory actually stores the physical path of the directory where data files and log files will be saved by the *expdp* and *impdp* utilities.

The CREATE DIRECTORY command is used to create the directory pointer within Oracle to the external directory you will use. Users who will access the Data Pump files must have the READ and WRITE privileges on the directory. However, oracle by default creates a directory 'DATA_PUMP_DIR' which is used as the default data pump directory. In that case, you need not give the directory argument to the datapump utilities. Below, we show how directories are created and read/write permissions are given to specific users.

The following command creates an oracle directory, i.e., DP_DIR which points to physical operating system directory ‘d:\oradump’.

```
SQL> CREATE DIRECTORY DP_DIR AS 'd:\oradump';
```

To view the details of the newly created directory objects, you can query the data dictionary view DBA_DIRECTORIES:

```
SQL> SELECT OWNER, DIRECTORY_NAME, DIRECTORY_PATH FROM DBA_DIRECTORIES;
```

Usually, the created directory is owned by SYS and DBA accounts can read/write to this directory. However, if you want to give permissions to read/write this directory to any other user who is not DBA than issue the following command (These privileges must be explicitly granted, rather than assigned through the use of roles).

The following command grants user HR to read, write access on the created directory DP_DIR.

SQL> GRANT READ, WRITE ON DIRECTORY DP_DIR TO HR;

In the above case user HR can use the DP_DIR directory for executing *expdp*, *impdp* utilities.

9.2: [Exporting data] Use *expdp* Utility to Export database objects

The Data Pump Export utility is invoked using the *expdp* command. The characteristics of the export operation are determined by the Export parameters you specify.

To experiment with export command, first create a test table *persons* in HR schema. Insert some dummy records in the table. Then, using the following command, export the *persons* table:

CMD> EXPDP HR/HR DIRECTORY=DP_DIR TABLES=PERSON DUMPFILE=EXP.DMP LOGFILE=EXP.LOG

The above command -

- Exports the *persons* table of *hr* schema to a dumpfile *exp.dmp*.
- The tables name in the dump file will be *hr.persons.* (contain schema information)
- The activities of export operation are logged in the logfile *exp.log*.
- HR is the HR user's password.
- The export file (exp.dmp) will be saved in the Oracle directory DP_DIR (this directory was created in Step 1 above)
- Since, the user HR is exporting the table and HR owns the table, table creation metadata will be copied in the dump file. This has significance during import of this dump file.

Note that, all the parameters of *expdp* command are not mandatory, and some of them can be un-specified to use default parameter values.

9.3 Use *impdp* Utility to Import database objects

One of the goals of exporting data is to recreate those data into another environment. Data pump *impdp* utility uses an exported dump file and log file, to re-create the database objects into another environment.

You can start a Data Pump Import job via the *impdp* executable from command line. Use the command-line parameters to specify the import mode and the locations for all the files. Below, we describe the steps to be performed for importing objects using *impdp* utility.

Like the export command, Date pump *impdp* utility also require directory objects to be created and all dump and log files must be there. Appropriate permissions must be given to the user who is executing *impdp* utility. Since, this topic has been discussed earlier, we skip it here.

Now, to experiment with *impdp*, drop the table *persons* (previously exported using *expdp*) of user HR from the same database. Connect as user HR and drop table persons using following SQL command:

SQL> DROP TABLE PERSONS PURGE;

Now, use *impdp* command to import the table previously exported as follows.

CMD> IMPDP HR/HR DIRECTORY=DP_DIR DUMPFILE=EXP.DMP LOGFILE=EXP.LOG

Note that following for *impdp* operation

- The above command imports everything in the *exp.dmp* file (in this case dupfile contains only a table, but it could contain many).
- Table is imported into HR schema, because in the dump file, table name is *hr.persons*. It does not matter whether HR or SYSTEM is doing the import operation. Table from dump file will always be imported according to its name in dump file. So, if the table name was *jack.persons* rather than *hr.persons*, then table would have been imported in jack schema!!! Of course, this requires that, HR is a DBA, because a normal user cannot load a table into another user's schema! But, DBA can do this!
- If HR schema was absent in the database then the above command would fail to load the schema.

You can also list the contents of dump file, without actually importing it by following command.

CMD> IMPDP HR/HR DIRECTORY=DP_DIR DUMPFILE=EXP.DMP SQLFILE=EXP.SQL

9.4 Several exporting modes

There are various export modes available. They are described below.

1. Full Database Export Mode:

A full export is specified using the FULL parameter. In a full database export, the entire database is unloaded. This mode requires that you have the EXP_FULL_DATABASE role.

The following command shows that user SYSTEM is taking a full database backup using export utility:

CMD> EXPDP SYSTEM/<PASSWORD> DIRECTORY=DP_DIR DUMPFILE=EXPFULL.DMP FULL=Y LOGFILE=EXPFULL.LOG

2. Schema Export Mode:

The schema export mode is invoked using the SCHEMAS parameter. This is the default mode of expdp command, when no parameters are specified. If you have no EXP_FULL_DATABASE role, you can only export your own schema. If you have EXP_FULL_DATABASE role, you can export several schemas in one go. Optionally, you can include the system privilege grants as well.

In the following example, HR user's schema is exported in SCHEMA_EXP.DMP dumpfile.

CMD> EXPDP HR/HR DIRECTORY=DP_DIR DUMPFILE=SCHEMA_EXP.DMP SCHEMAS=HR

3. Table Export Mode:

This export mode is specified using the TABLES parameter. In this mode, only the specified tables, partitions and their dependents are exported. If you do not have the EXP_FULL_DATABASE role, you can export only tables in your own schema. You can only specify tables in the same schema. In the following example, three tables, EMPLOYEES, JOBS, and DEPARTMENTS are exported. These tables will be exported from HR schema.

CMD> EXPDP HR/HR DIRECTORY=DP_DIR DUMPFILE=TABLES_EXP.DMP TABLES=EMPLOYEES, JOBS, DEPARTMENTS

9.5 Several import modes

When the source of the import operation is a dump file set, specifying a mode is optional. If no mode is specified, then Import attempts to load the entire dump file *set in the mode in which the export operation was run*. You may sometimes want not to import entire contents of the dump file. In that case, you can optionally specify modes to import specific tables, schemas, etc. from the dumpfile. The available modes are described below.

1. **Full Import Mode**:

The full import mode loads the entire contents of the source (export) dump file to the target database. However, you must have been granted the IMP_FULL_DATABASE role on the target database. The data pump import is invoked using the *impdp* command in the command line with the FULL parameter specified in the same command line as follows.

The following command imports the EXPFULL.DMP dump file created earlier.

CMD> IMPDP SYSTEM/<PASSWORD>DIRECTORY=DP_DIR DUMPFILE=EXPFULL.DMP FULL=Y LOGFILE=IMPFULL.LOG

Note that, specifying FULL = y is not required, since by default *impdp* loads entire dumpfile.

2. **Schema Import Mode**:

The schema import mode is invoked using the SCHEMAS parameter. Only the contents of the specified schemas are loaded into the target database. The source dump file can be a full, schema-mode, table, or tablespace mode export files. If you have an IMP_FULL_DATABASE role, you can specify a list of schemas to load into the target database. In this case -

- First, the schemas themselves are created (if they do not already exist), including system and role grants, password history, and so on. Then all objects contained within the schemas are imported.
- Non-privileged users can specify only their own schemas or schemas **remapped to their own schemas**. In that case, no information about the schema definition is imported, only the objects are imported.

CMD> IMPDP HR/HR DIRECTORY=DP_DIR DUMPFILE=EXPFULL.DMP SCHEMAS=HR

In the above command HR user loads the HR schema objects from the dumpfile. HR schema must have the IMP_FULL_DATABASE role.

3. **Table Import Mode**:

This export mode is specified using the TABLES parameter. In this mode, only the specified tables, partitions and their dependents are imported. If you do not have the IMP_FULL_DATABASE role, you can import only tables in your own schema. If you do not supply a schema_name in the TABLES parameter, it defaults to that of the current user. To specify a schema other than your own, you must either have the IMP_FULL_DATABASE role or remap the schema to the current user.

CMD> IMPDP HR/HR DIRECTORY=DP_DIR DUMPFILE=EXPFULL.DMP TABLES=EMPLOYEES, JOBS, DEPARTMENTS

The above command imports three tables EMPLOYEES, JOBS, DEPARTMENTS of HR schema (of the source database). The objects are imported into HR schema of the target database.

Below, we describe few other parameters that are important to understand for working with import, export.

9.6 Explanation of impdp and expdp parameters

9.6.1 TABLE_EXISTS_ACTION parameter

TABLE_EXISTS_ACTION parameter tells *impdp* what to do if the table it is trying to create already exists. It is specified in the command line as follows along with other import parameters:

TABLE_EXISTS_ACTION= {SKIP | APPEND | TRUNCATE | REPLACE}

Default value: SKIP (Note that if CONTENT=DATA_ONLY is specified, the default is APPEND, not SKIP.)

The possible values have the following effects:

- SKIP leaves the table as is and moves on to the next object. This is not a valid option if the CONTENT parameter is set to DATA_ONLY.
- APPEND loads rows from the source and leaves existing rows unchanged.
- TRUNCATE deletes existing rows and then loads rows from the source.
- REPLACE drops the existing table and then creates and loads it from the source. This is not a valid option if the CONTENT parameter is set to DATA_ONLY.

The following is an example of using the TABLE_EXISTS_ACTION parameter.

CMD> IMPDP HR/HR TABLES=EMPLOYEES DIRECTORY=DP_DIR DUMPFILE=EXPFULL.DMP TABLE_EXISTS_ACTION=REPLACE

9.6.2 CONTENT parameter

- CONTENT parameter enables you to filter what is loaded during the import operation. Its values are:

CONTENT = {ALL | DATA_ONLY | METADATA_ONLY}

- ALL loads any data and metadata contained in the source. This is the default.
- DATA_ONLY loads only table row data into existing tables; no database objects are created.

- METADATA_ONLY loads only database object definitions; no table row data is loaded.

The following is an example of using the CONTENT parameter.

CMD> IMPDP SYSTEZM/<PASSWORD> DIRECTORY=DP_DIR DUMPFILE=EXPFULL.DMP CONTENT=METADATA_ONLY

The above command will execute a full import that will load only the metadata in the expfull.dmp dump file. It executes a full import because that is the default for file-based imports in which no import mode is specified.

9.6.3 REMAP_SCHEMA parameter

REMAP_SCHEMA loads all objects from the source schema into a target schema. It is specified along with other parameters as follows:

REMAP_SCHEMA=source_schema:target_schema

If the schema you are remapping to does not already exist, then the import operation creates it, provided that the dump file set contains the necessary CREATE USER metadata for the source schema, and provided that you are importing with enough privileges.

For example, entering the following Export commands creates the dump file sets with the necessary metadata to create a schema, because the user SYSTEM has the necessary privileges:

CMD> EXPDP SYSTEM/<PASSWORD> SCHEMAS=HR

If your dump file set does not contain the metadata necessary to create a schema, or if you do not have privileges, then the target schema must be created before the import operation is performed. This is because the unprivileged dump files do not contain the necessary information for the import to create the schema automatically.

If the import operation does create the schema, then after the import is complete, you must assign it a valid password in order to connect to it. The SQL statement to do this, which requires privileges, is:

SQL>ALTER USER SCHEMA_NAME IDENTIFIED BY NEW_PASSWORD

Unprivileged users can perform schema remaps only if their schema is the target schema of the remap. (Privileged users can perform unrestricted schema remaps). For example, SCOTT can remap BLAKE's objects to SCOTT, but SCOTT cannot remap SCOTT's objects to BLAKE.

Examples

Suppose that, as user SYSTEM, you execute the following Export and Import commands to remap the hr schema into the scott schema:

CMD> EXPDP SYSTEM SCHEMAS=HR DIRECTORY=DP_DIR DUMPFILE=HR.DMP

CMD> IMPDP SYSTEM DIRECTORY=DPUMP_DIR1 DUMPFILE=HR.DMP
REMAP_SCHEMA=HR: JOHN

In this example, if user JOHN already exists before the import, then the Import REMAP_SCHEMA command will add objects from the HR schema into the existing JOHN schema. **You can connect to the john schema after the import by using the existing password (without resetting it).**

If user JOHN does not exist before you execute the import operation, Import automatically creates it with an unusable password. This is possible because the dump file, hr.dmp, was created by SYSTEM, which has the privileges necessary to create a dump file that contains the metadata needed to create a schema. However, you cannot connect to JOHN on completion of the import, unless you reset the password for JOHN on the target database after the import completes.