DATABASE MANAGEMENT SYSTEMS
(CSE2007)

LAB MANUAL

Department of Computer Science and Engineering
VIT-AP University, Amaravati
Andhra Pradesh

DBMS LAB MANUAL

Table of Contents

STRUCTURED QUERY LANGUAGE

        Structured Query Language (SQL) is a non-procedural database language used for storing and retrieving data from the database.

        SQL was invented by IBM in early 1970's. IBM was able to demonstrate how to control relational databases using SQL. The SQL implemented by ORACLE CORPORATION is 100% compliant with ANSI/ISO standard SQL data language.

Oracle's database language is SQL. (Oracle is a software package containing both front end tools and back end tools. The back end tool is SQL*PLUS - It submits SQL and PL/SQL statements to the server for execution.) A table is a primary database object of SQL that is used to store data. A table holds data in the form of rows and columns.

SQL supports the following categories of commands to communicate with the database:-

Commands	Statements	Description
Data Definition Language	Create , Alter Drop, Rename Truncate	Sets up ,changes and removes data structures called tables
Data Manipulation Language	Insert , delete ,update

Select | Adds , removes and changes rows in Db. Retrieves data from Db. |
| Data Control Language | Grant ,revoke | Gives or removes access rights to others. |
| Transaction Control Language | Commit, Rollback, save point | Manages the changes made by DML. |

Oracle Data types

Type Name	Syntax	Description	Range	Valid Data
Character	char (length) ex: char(10)	Fixed length character	1 to 2000 bytes	'1234567890' 'dfee'
Character	varchar2(length) ex: varchar2(5)	Variable length Character string	1 to 4000 bytes	'asqeq' '1234' 'ssf%.sd'
Number	number	Integer of any	Integer range	Any number

		maximum range Only 3 digits	38 digits	123, 789
	number(3)			
	number (4,1)	Float of max 1 decimal place	after decimal -84 to 127	123.4,111.5 12.4
Date	date	Fixed length date -7 bytes for each date, month	Jan 1 ,4712 BC to Dec 31, 4712 AD	'01-jan-01' '31-feb-2005'
Long	Long	To store Variable character length (one table only one long type)	Max 2 GB	'ggfg….'
Raw	Raw	Binary data or byte strings (manipulation of data cannot be done)	Max 2000 bytes	
Long Raw	Long raw	Binary data of Variable length	Max 2GB	
Large Object	CLOB	Stores character Object with single byte Character	Max 4 GB	
	BLOB	Stores large binary objects(Graphics, video clips and sound files)		
	BFILE	Stores file pointers to LOBs managed by file systems external to the Db.		BFILE('dir. Name', 'filename')
Time	Timestamp	Date with time (No separate time type)		'24-sep-75,06:12:12'

SYNTAX for the SQL statements:-

DDL

1. CREATE

a. simple creation

```
CREATE TABLE  < tablename> (
      <column name1 >   < datatype>,
      <column name 2>   < datatype>,
      <column name 3>   < datatype>
      );
```

b. without constraint name

```
CREATE TABLE  < tablename>  (
      <column name 1>   < datatype>,
      <column name 2>   < datatype> unique ,
      <column name 3>   < datatype> ,
       primary key ( <column name2>)
      );
```

c. with constraint name

```
CREATE TABLE  < tablename1>  (
      <column name 1>   < datatype>,
      <column name 2>   < datatype>,
      constraint  < constraint name1 >  primary key ( <column name1>),
      constraint  <constraint name2>  foreign key (<column name2>)
      references  <tablename2> (<column name1>)
      );
```

d. with check constraint

```
CREATE TABLE  < tablename>  (
      <column name1 >   < datatype> ,
       <column name 2>  < datatype>,
       check ( < column name 1  > in  ( values) )
       check ( < column name 2  > between <val1> and  <val2> )
```

e. for sequence creation

```
CREATE SEQUENCE <sequence name>        INCREMENT  /
DECREMENT
BY   <val > START WITH  <val>
```

f. for ROLE creation

```
for grouping together a set of access rights
CREATE  ROLE  <role name>
```

2. ALTER

a. Add –to add new columns

ALTER TABLE <tablename> add (<column name > < datatype>)

b. Modify the datatype or increase / decrease the column width

ALTER TABLE <tablename> modify (<column name > < newdatatype>)

c. drop –delete column or remove constraint

ALTER TABLE <tablename> drop column < column name>;
ALTER TABLE <tablename> drop constraint < constraint name > ;

*** Constraints addition and column changing (datatype or decreasing the width) can be done only if column values are null.

3. TRUNCATE

Removes the rows, not the definition
TRUNCATE TABLE <tablename>;

4. DROP

Removes the rows and table definition
DROP TABLE <tablename>;

5. RENAME

Changes table name
RENAME < old tablename> to < new tablename>;

DML

1. INSERT

a. Inserting values from user

INSERT INTO <tablename> VALUES(val1,val2 …);

b. Inserting interactively

INSERT INTO <tablename> VALUES(&<column name1> , & <column
name2> …);

c. Inserting null values

INSERT INTO <tablename> VALUES(val1,' ',' ',val4);

d. Inserting a sequence number

INSERT INTO <tablename> VALUES (.. ,…….,
<sequence name > . NEXTVAL)

2. SELECT

a. Simple select

    SELECT * FROM < tablename>;
    SELECT <col1>, <col2> FROM < tab1>;

b. Alias name

    SELECT <col1> <alias name 1> , <col2> < alias name 2>FROM <
    tab1>;

c. With distinct clause

    SELECT DISTINCT <col2> FROM < tab1>;

d. With where clause

    SELECT <col1>, <col2> FROM < tab1>
    WHERE <conditions>;

e. Select to create table

    CREATE TABLE <tablename> as SELECT <column names > FROM
    <existing table>;

f. Copy only table definition

    CREATE TABLE <tablename> as SELECT <column names > FROM
    <existing table> WHERE 1=2;

g. select to insert

    INSERT INTO < table> (SELECT <column names > FROM <
    existing table>);

3. UPDATE

a. Simple update

    UPDATE < tablename> SET <col> = < new value>;

b. Using where clause

    UPDATE < tablename> SET <col1> = < new value> , <col2> = <
    new
    value> WHERE <conditions>;

4. DELETE

a. Delete all rows

    DELETE FROM <tablename>;

b. Using where clause –delete specific rows

DELETE FROM <tablename> WHERE <conditions>;

TCL

1. COMMIT

a. To permanently save

COMMIT;

2. SAVEPOINT

a. To make markers in a lengthy transaction

SAVEPOINT <savepoint name>;

3. ROLLBACK

a. To undo changes till last commit

ROLLBACK;

b. To undo changes till a marker

ROLLBACK <savepoint name>;

DCL

1.GRANT

a. Grant all privileges

GRANT ALL ON < object name> TO <username>;

b. Grant certain privileges

GRANT <privileges > ON < object name> TO <username>;

c. Grant Execute privilege

GRANT EXECUTE ON <function name> TO <username>;

d. Setting privileges to ROLE

GRANT <any DML command> TO <role name>;
GRANT <role name> TO <user name>;

***User can grant insert , delete , update , select on his tables or views or materialized views and also grant references to columns.

***grant execute permission on procedures, functions, packages, abstract datatypes, libraries, index types and operators.

***grant select ,alter on sequences.

2. REVOKE

a. REVOKE <privileges > on < object name> FROM <username>;

b. REVOKE SELECT ,UPDATE ON <table> FROM <username>;

OPERATORS IN SQL*PLUS

Type	Symbol / Keyword	Where to use
Arithmetic	+ , - , * , /	To manipulate numerical column values, WHERE clause
Comparison	=, !=, <, <=, >, >=, between, not between, in, not in, like, not like	WHERE clause
Logical	and, or, not	WHERE clause, Combining two queries

SQL*PLUS FUNCTIONS

- Single Row Functions
- Group functions

Single Row Functions

- returns only one value for every row.
- Can be used in SELECT command and included in WHERE clause
- Types
  Date functions Numeric
  functions Character
  functions Conversion
  functions Miscellaneous
  functions

Date functions:

syntax	Description
add_months(date,no. of months)	Return the date after adding the number of months
last_day(date)	Returns the last date corresponding to the last day of the month
months_between(date1,date2)	Returns the numeric value of the difference between the months.
round(date, [format])	Format – 'day', 'month' , 'year' rounded to the nearest format specified
next_day(date, day)	Returns the next date of the day
trunc(date, [format])	Format – 'day', 'month' , 'year' Day – previous nearest Sunday Month – start date of the month Year – start date of the year
greatest(date1, date2,…)	Returns the latest date

| new_time(date, 'this', 'other') | New_time(date, 'est', 'yst') converts date from one time zone to another |

Numeric functions:

syntax	Description
abs ( )	Returns the absolute value
ceil ( )	Rounds the argument
cos ( )	Cosine value of argument
cosh( )	Hyperbolic cos
exp ( )	Exponent value
floor( )	Truncated value
power (m,n)	N raised to m
mod (m,n)	Remainder of m / n
round (m,n)	Rounds m's decimal places to n
trunc (m,n)	Truncates m's decimal places to n
sqrt (m)	Square root value

Character Functions:

syntax	Description
initcap (char)	Changes first letter to capital
lower (char)	Changes to lower case
upper (char)	Changes to upper case
ltrim (char, set)	Removes the set from left of char
rtrim (char, set)	Removes the set from right of char
translate(char, from, to)	Translate 'from' anywhere in char to 'to'
replace(char, search string, replace string)	Replaces the search string to new
substring(char, m , n)	Returns chars from m to n length
lpad(char, length, special char)	Pads special char to left of char to Max of length
rpad(char, length, special char)	Pads special char to right of char to Max of length
chr(number)	Returns char equivalent
length(char)	Length of string
decode(columnname, col value, replace value)	Changes column values from specified value to new
||	Concatenation of strings

Conversion functions:

syntax	Description
to_char(date, format)	Converts date to specified format string
to_date(char,format)	Converts string to date format
to_number(char)	Converts a numerical string to number

Miscellaneous functions

syntax	Description
uid	Integer value of login
user	Username
nvl (column name, new value)	Replaces null values to new value in the column specified
vsize(value)	Returns number of bytes in value

Group functions:

Result based on group of rows.

Syntax	Description
count (*), count (column name), count (distinct column name)	Returns number of rows
min (column name)	Min value in the column
max (column name)	Max value in the column
avg (column name)	Avg value in the column
sum (column name)	Sum of column values
stdev(column name)	Standard deviation value
variance(column name)	Variance value

SET OPERATORS

 -Combine the results of two queries into single one

 -Rule:

        queries using set operators should have same number of columns and corresponding columns should be of same data types.

1. UNION

        Returns all distinct rows by both queries

        < query 1> UNION < query2>;

2. UNION ALL

        Returns all rows by both the queries including duplicates

< query 1> UNION ALL< query2>;

3. INTERSECT

Returns common rows by both the queries

< query 1> INTERSECT < query2>;

4. MINUS

Returns distinct rows in the first query

< query 1> MINUS < query2>;

JOINS

To combine the datas from many tables.

It is performed by WHERE Clause which combines the specified rows of the tables.

Type	Sub type	Description
Simple join	Equi join (=)	Joins rows using equal value of the column
	Non – equi join (<, <=, >, >=, !=, < >)	Joins rows using other relational operators(except =)
Self join	-- (any relational operators)	Joins rows of same table
Outer join	Left outer join ((+) appended to left operand in join condition)	Rows common in both tables and uncommon rows have null value in left column
	Right outer join ((+) appended to right operand in join condition)	Vice versa

SUB QUERIES

- nesting of queries

- a query containing a query in itself

- innermost sub query will be executed first

- the result of the main query depends on the values return by sub query

- sub query should be enclosed in parenthesis

1. Sub query returning only one value

a. relational operator before subquery.

SELECT … WHERE < column name > < relational op.> < subquery>;

2. Sub query returning more than one value

a. ANY

main query displays values that matches with any of the values returned by
sub query

SELECT .. WHERE < column name > < relational op.> ANY

(<subquery>);

b. ALL

main query displays values that matches with all of the values returned by
sub query

SELECT .. WHERE < column name > < relational op.> ALL

(<subquery>);

c. IN

main query displays values that matches with any of the values returned by
sub query

SELECT … WHERE < column name > IN (<subquery>);

d. NOT IN

main query displays values that does not match with any of the values
returned by sub query

SELECT … WHERE < column name > NOT IN (<subquery>);

e. EXISTS

main query executes only if the sub query returns any few rows

<main query> EXISTS (<sub query>);

f. NOT EXISTS

main query executes only if the sub query does not return any rows

<main query> NOT EXISTS (<sub query>);

g. CONTAINS

A query selects what is selected by its correlated sub query

(<query>) CONTAINS (<query>);

h. EXCEPT

A query selects what is not selected by its correlated sub query

(<query>) EXCEPT (<query>);

i. GROUP BY CLAUSE

used to group same value in a column together and apply a group function to
it

Rule:

Select attributes and group by clause attributes should be same

Select <column1>, <column2> from <table name>
Where <conditions>
GROUP BY <column2>, <column1>;

j. HAVING CLAUSE

used to apply a condition to group by clause

Select <column1>, <column2> from <table name>
Where <conditions>
GROUP BY <column2>, <column1>
HAVING < conditions>;

k. ORDER BY CLAUSE

Used along with where clause to display the specified column in ascending
order or descending order .Default is ascending order

Select <column1>, <column2> from <table name>
Where <conditions>
ORDER BY <columns> DESC / ASC;

VIEW

View is a virtual table

It is a subset of a table derived from the original large table for

convenient manipulation

It restricts the database access and allows data independence

CREATE OR REPLACE VIEW <view name> AS <query>;

CREATE OR REPLACE VIEW <view name>(alias column name) AS

<query>;

Rules:

View derived from single table is updateable (if it has derived the primary

key or any candidate key)

Not updateable if view is derived from multiple tables and also view contains group by, aggregate functions, distinct or reference to pseudo column number

To create read only view:

CREATE OR REPLACE VIEW <view name> AS <query> WITH READ ONLY;

To create an object view:

CREATE OR REPLACE VIEW <view name> (any column name, type name) AS <selection with type name specification>;

ABSTRACT DATA TYPE

-Combining basic data types

-ADT can be nested , refer other ADT

-ADT can be used to create object tables

CREATE TYPE <typename> AS OBJECT (<columnname><data type>….);

To include new type in table

CREATE TABLE <table name> (< column name> <data type> <column name> <type name>………….);

To insert into object table

INSERT INTO <table name > VALUES (values .. ,type name(values..));

-Adding methods to objects

Member functions can be added to objects and body defined separately

CREATE OR REPLACE TYPE <type name> AS OBJECT
(< column name>   <data type> …
 member function <name> (<parameter> IN / OUT <datatype>)
return <data type>);

CREATE OR REPLACE TYPE BODY <type name> AS
MEMBER FUNCTION <name> (<pararmeter> <data type>)
RETURN <data type> IS
      begin
      ….
      end;

To apply Member function

SELECT < variable of type name. function name(arguments)> ….

NESTED TABLES

Adding table within a table

First create type.

Create nested table type

```
CREATE TYPE <nested type>AS TABLE OF <type name>
CREATE TABLE < > ( <name> <nested type>)
NESTED name STORE AS <new name>
```

To Insert

```
INSERT INTO TABLE < > VALUES ( ….name ( type ame ( ..), type name
(..) ………) …);
```

VARYING ARRAYS

Range of values in a single row.

To create an array of same data type.

Maximum values stored will be the size of the array.

```
CREATE OR REPLACE  TYPE <type name> AS
Varray (6) of varchar2(25);

CREATE TABLE  < table name> ( <name > <data type> …
                                <name > <type name>);
INSERT INTO <name> VALUES ( … type( 3 values));
INSERT INTO <name> VALUES ( … type( 4 values));
```

To select from table containing varray

```
cursor  <name>  is
       select *  from <table name>
begin
       for <record name>  in <cursor name>
       loop
       dbms_output. put_line( ddf);
       for i  in 1 . . rec. type name. count
       loop
               put( rec. type name (i));
       end loop;
end loop;
;
```

TABLE PARTITION

Table can be partitioned and stored in different location to avoid data corruption and facilitate back up and recovery. The single logical table can be split into number of physically separate tables based on arrange of key values.

RULE:

Table containing advanced data types can not be partitioned

```
CREATE TABLE  < tablename> (
        <column name1 >   < datatype>,
        <column name 2>   < datatype>,
        <column name 3>   < datatype>
        ) PARTITION BY RANGE(<column name1 >)
        (PARTITION <partition name1> VALUES LESS THAN (<value>),
        PARTITION <partition name1> VALUES LESS THAN (<value>));
```

ADDING PARTITION

```
ALTER TABLE <table name> ADD PARTITION <partition name>
VALUES LESS THAN (<value>);
```

SPLITTING PARTITION

```
ALTER TABLE <table name> SPLIT PARTITION <old partition name>
AT (<value>) INTO (PARTITION <partition name1>, PARTITION
<partition name2>);
```

DROPPING PARTITION

```
ALTER TABLE <table name> DROP PARTITION <partition name>;
```

INDEX

Indexes are data structures associated with the table to allow fast access to the datas

Index can be created for one or many columns

When an index is created the column values are sorted and stored along with ROW ID

When the rows are updated the index values maintained are automatically changed internally

** primary key and unique columns are index by default

simple index

CREATE INDEX <index name> ON < table name> (<column name>);
CREATE UNIQUE INDEX <index name> ON < table name> (<column name>);

Composite index

CREATE INDEX <index name> ON < table name> (<column name>,<column name>, . . .);

PL/SQL

Procedural language SQL

SQL statements combined with procedural constructs

*** Before running the PL/SQL block the following command should be given to

enable the output

    SQL>SET SERVEROUTPUT ON;

PL/SQL Block

    DECLARE
    <declarations>
    BEGIN
    <executable statements>
    EXCEPTION
    <exception handlers>
    END;

Data types

    Boolean
    Integer
    Real
    Character – varchar2( ), char( )
    Rowid
    Raw
    LOB

Attributes

    %type – used to refer to the database columns
    %rowtype – represents a row in table
    variable name tablename.columnname<attribute>

Control Structures

1. Conditional Control

IF <condition> THEN
<statements>;
END IF;

2. Iterative Control

Simple loop

LOOP
<statements>;
EXIT WHEN <condition>;
END LOOP;

WHILE loop

WHILE <condition>
LOOP
<statements>;
END LOOP;

FOR loop

FOR <variable> IN [REVERSE] <initial value> . . <final value>
LOOP
<statements>;
END LOOP;

CURSOR

Cursor is the pointer to the temporary area (context area) created for

storing the data manipulated by a PL/SQL program

Attributes

To be added in the exit condition

%notfound
%found
%rowcount
%isopen

Block

DECLARE
<declarations>
CURSOR <cursor name> IS <query>

```
BEGIN
OPEN <cursor name>;
LOOP
FETCH <cursor name> INTO <local variables>;
<statements>;
EXIT WHEN <cursor name> <attribute name>;
END LOOP;
CLOSE <cursor name>;
END;
```

SUBPROGRAMS

Subprograms are PL/SQL block that can be created and invoked whenever required

PROCEDURE

Procedure is the subprogram that does not return any value

```
CREATE OR REPLACE PROCEDURE <procedure
name>(<parameters>) IS
<local declarations>;
BEGIN
<executable statements>;
END;
```

```
To Execute
exec <procedure name> (<parameter values>);
```

Parameter specification in list

in   – value passed
out – value returned
inout—value in and out

```
CREATE OR REPLACE PROCEDURE <procedure name>
(<variable> IN <data type>) IS
<local declarations>;
BEGIN
<executable statements>;
END;
```

FUNCTION

Function is a subprogram which returns a value.

CREATE OR REPLACE FUNCTION <function name>(<parameters>)

RETURN TYPE IS
<local declarations>;
BEGIN
<executable statements>;
END;

To Execute function
1.   declare
                Local declarations;
        begin
                <variable> := <function name>(values);
                …
        end;

2.   select function name(Values) from dual;

PACKAGE

It encapsulates subprograms, cursors, variables, constants

Package declaration contains function or procedure declarations

Package body contains body of function or package

CREATE PACKAGE <package name > IS <declarations>
BEGIN
<executable statements>
END <package name>;

CREATE PACKAGE BODY<package name > IS <declarations>
BEGIN
<executable statements>
END <package name>;

To execute package
<Package name > . < subprogram name>;

TRIGGERS

It is a stored procedure which is fired when any manipulation on the

specified table

It is used to enforce complex integrity constraint, security authorizations.

CREATE OR REPLACE TRIGGER <name>
[BEFORE/AFTER] [INSERT/UPDATE/DELETE] ON <table name>
[FOR EACH STATEMENT/ FOR EACH ROW] WHEN <condition>;

CREATE OR REPLACE TRIGGER <name> [BEFORE/AFTER]
[INSERT/UPDATE/DELETE] ON <table name>

```
[FOR EACH STATEMENT/ FOR EACH ROW]
DECLARE
        <local>
BEGIN
        <statements>
END;/
```

   *For each row/statement specifies that the trigger fires once per row

Variable names used in triggers
*two are used :old and :new
*old specifies the row value before change and new specifies the value after change

Disabling and Enabling Triggers

```
ALTER TRIGGER <trigger name> DISABLE;
ALTER TABLE <name> DISABLE <trigger name>;
ALTER TABLE <name> DISABLE ALL TRIGGERS;
ALTER TABLE <name> ENABLE <trigger name>;
ALTER TABLE <name> ENABLE ALL TRIGGERS;
```

Dropping Triggers

DROP TRIGGER <trigger name>;

EXAMPLES:

DDL
```
SQL> create table emp (
  2  id number(3),
  3  salary number(10,2),
  4  name varchar2(30),
  5  dob date,
  6  addr varchar2(20),
  7  constraint pk1 primary key (id),
  8  constraint ck1 check (salary>5000),
  9  constraint uk1 unique(name));
Table created.

SQL> create table dependent
  2  (did number(3),
  3  dname varchar2(20),
  4  eid number(3),
  5  constraint pk4 primary key(did, eid),
  6  constraint fk1 foreign key (eid) references emp(id));
Table created.

SQL> alter table dependent drop constraint fk1;
Table altered.

SQL> alter table dependent add constraint fk1
  2  foreign key (eid) references emp(id) on delete cascade;
Table altered.

SQL> create sequence seq increment by 1 start with 100;
Sequence created.

SQL> insert into emp values (seq.nextval,6000,'cra','01-jan-75','amravati');
1 row created.

SQL> create sequence seq1 start with 100;
Sequence created.
```

DML
SQL> insert into emp values (seq.nextval,6000,'ncs','01-jan-75',' amravati ');
1 row created.

SQL> select * from dependent;

DID	DNAME	EID
12	raj	101
13	sk	102

SQL> select * from emp;

ID	SALARY	NAME	DOB	ADDR
102	6000	cra	01-JAN-75	amravati
104	6000	ncs	01-JAN-75	amravati
100	6000	ncs1	01-JAN-75	amravati
101	6000	ncs2	01-JAN-75	amravati

SQL> select id from emp
        intersect
        select did from dependent;
no rows selected

SQL> select id from emp
        union
        select did from dependent;
    ID

    12
    13
    100
    101
    102
    104
6 rows selected.

SQL> select id from emp
  2 union all
  3 select did from dependent;
    ID

    102
    104
    100
    101

```
        12
        13
6 rows selected.

SQL>  select id from emp
         minus
         select did from dependent;
      ID
----------
      100
      101
      102
      104

SQL>   select * from emp,dependent
 2    where id=did(+);

  ID   SALARY        NAME   DOB     ADDR    DID  DNAME    EID

 102    6000            cra   01-JAN-75amravati
 104    6000            ncs   01-JAN-75amaravat
 100    6000            ncs1  01-JAN-75amravati
 101    6000            ncs2  01-JAN-75amravati

SQL>  select * from emp,dependent
        where id(+)=did;

  ID    SALARY    NAME    DOB      ADDR    DID    DNAME    EID
                                                   12             raj
                          101
                                                   13             sk
                          102



SQL>  select * from emp
 2   where exists( select * from dependent where id = eid and id=101);

 ID    SALARY     NAME                 DOB            ADDR
 101    6000            ncs2                  01-JAN-75 amravati

SQL>  select * from emp
 2   where not exists( select * from dependent where id = eid );

   ID    SALARY NAME                 DOB            ADDR
   104    6000   ncs                  01-JAN-75    amravati
   100    6000   ncs1                 01-JAN-75    amravati
```

```
SQL> select addr from emp
  2  group by addr
  3  having count(*) > 2;
ADDR
--------------------
amravati


SQL>  select * from emp order by name desc;
 ID    SALARY  NAME                 DOB      ADDR        DOJ
104    6000    ncs                  01-JAN-75  amravati
102    6000    cra                  01-JAN-75  amravati

SQL>  select * from emp order by name asc;
ID    SALARY  NAME                  DOB      ADDR        DOJ
102    6000    cra                  01-JAN-75  amravati
104    6000    ncs                  01-JAN-75  amravati
```

ABSTRACT DATA TYPES AND NESTED TABLES

```
SQL> create type addr as object
  2  ( s varchar2(25),
  3   c varchar2(25));
  4  /
Type created.

SQL> alter table a
  2  add ad addr;
Table altered.
SQL> desc a
 Name                                   Null?        Type
 --------------------------------------- -------- --------------------------
 N                                                 NUMBER
 F                                                 NUMBER(4,2)
 C                                      VARCHAR2(10)
 D                                       DATE
 AD                                      ADDR

SQL> insert into a values(1,1.2,'dsd','01-jan-05',addr('arrr','sdfsf'));

1 row created.
SQL>  select * from a;

  N      F C     D                   AD(S, C)
---------- ---------- ---------- ---------------------------------------------------------------
1     1.2 dsd     01-JAN-05       ADDR('arrr', 'sdfsf')

SQL> create type t as object
```

```
  2  (n number);
  3  /

Type created.

SQL> create type nt as table of t;
  2  /

Type created.
SQL>  create table person
  2   (n number,
  3   d nt)
  4  nested table d store as nt_TAB;

Table created.
SQL>  insert into person values( 1,nt(t(1)));
1 row created.

SQL>  insert into person values( 1,nt(t(1),t(2),t(3)));
1 row created.

SQL> select * from person;

  N          D(N)

  1          NT(T(1))
  1           NT(T(1), T(2), T(3))


TABLE PARTITIONS
SQL> create table r (e number(3)) partition by range (e)
  2  (partition p1 values less than (10),partition p2 values less than (20));

Table created.


SQL> select * from r;

       E
----------
       1
       4
      10
      16


SQL> select * from r partition (p1);
```

```
        E
----------
        1
        4

SQL> select * from r partition (p2);

        E
----------
       10
       16
```

VARRAY

```
SQL> create type vr AS
  2  Varray (6) of varchar2(25);
  3  /
Type created.

SQL> alter table emp add new vr;
Table altered.

SQL> desc emp;
 Name                              Null?    Type
 ---------------------------------- -------- -----------------
 ID                                NOT NULL NUMBER(3)
 SALARY                                     NUMBER(10,2)
 NAME                                       VARCHAR2(30)
 DOB                                        DATE
 ADDR                                       VARCHAR2(20)
 DOJ                                        TIMESTAMP(6)
 VR                                         NUMBER(3)
 NEW                                        VR

SQL>  insert into emp values(123,7899,'vvs','09-feb-90','arumbarathi','09-sep-
90,12:34:45',89,vr('asd','lll'));
1 row created.

SQL>  insert into emp values(129,7899,'gpa','09-feb-90','arumbarathi','09-sep-
90,12:34:45',89,vr('asd','lll','pop','push'));
1 row created.
```

ID	SALARY	NAME	DOB	ADDR
129	7899	gpa	09-FEB-90	arumbarathi

```
DOJ                              NEW    VR
09-SEP-90 12.34.45.000000 PM    89        VR('asd', 'lll', 'pop', 'push')
```

MORE DDLs

1. Create new table from existing table with all records

```
SQL> create table z as(select * from t);
        Table created.
```

2. Create table with different column names from existing column

```
SQL> create table emp1 as select empno "emplno",ename "name",sal
"salary" from emp;
        Table created.
```

3. Copy the structure of an existing table

```
SQL> create table emp2 as (select * from emp1 where 1=2);
Table created.
```

MORE DMLs

```
SQL> insert into student values('Len',27,'01-may-1974','Mech','exc');
SQL> delete from student where name = 'Chandru';
SQL> update student set age = 23 where name =  'ravi';
SQL> select * from student;
        NAME          AGE DOB      DEPT COND
        ---------- --------- --------- ---- ----
        Len           27 01-MAY-74 Mech  exc
        Priya          21 24-APR-83 cse  Good
        Vino           21 12-MAR-83 med  exc


SQL> select * from student where age>21;

SQL> select * from student where age=21 and dept = 'med';
        NAME       AGE   DOB      DEPT COND
        ---------- --------- --------- ---- ----
        Vino           21 12-MAR-83 med  exc
        Parkavi        21 15-JUL-83  med  exc


SQL> select * from student where dob is null;
        NAME          AGE DOB      DEPT COND
        ---------- --------- --------- ---- ----
```

SQL> select * from student where name like'%i';
SQL> select * from student order by age;
SQL> select distinct dept from student;
SQL> select age+10 from student;
SQL> select dept from student group by dept;
     DEPT

     CSE
     Mech
SQL> select dept from student where age=21 group by dept;
     DEPT
     ----
     cse
     med

SQL> select dept from student group by dept having dept='med';
SQL> select name from student where age=any(select age from student
         where age >21);
SQL> select * from student a,student b where a.name=b.name;

1. Display the details of al person name with j and ending with n

    SQL> select name from t where name like 'j%n';
    NAME
    ----
    jon

2. Display details of all persons three letter names

    SQL> select * from t where name like '___';

NO	NAME	SAL	HRA	DA	NET	DOJ
20	jon	200				30-AUG-98
30	sam	450				10-JAN-97
40	tow	100				23-DEC-67

3. List name of persons getting sal above 3000 and below 4000

    SQL> select * from t where sal between 300 and 400 ;

NO	NAME	SAL	HRA	DA	NET	DOJ
10	rash	320				12-SEP-89

4. List the details of persons who work in either dept 30 or 10

    SQL> select * from t where dpn=30 or dpn=10;

```
    NO NAME       SAL  HRA DA  NET   DOJ       DPN
    ----- ----  --------- --------- --------- --------- --------- --------
    20 jon        200                30-AUG-98  30
    30 sam        450                10-JAN-97  10
```

5. Display .5% salary of all persons

```
        SQL> select name ,sal,(.5*sal)"half sal" from t;
        NAME        SAL  half sal
        ---------- --------- ---------
        rash        320     160
        jon         200     100
        sam         450     225
```

6. Display the name of all persons joined after smith
```
        SQL> select  * from t where doj >(select doj from t where name='rash');
        NO NAME   SAL  HRA DA NET   DOJ        DPN
        -- ---- ------ ----- --------- --------- -------- --------
        20 jon        200                30-AUG-98    30
        30 sam        450                10-JAN-97    10
```

7. Display the salary of smith and who receive higher salary.

```
        SQL>  select sal from emp where sal >=
          2  (select sal from emp where ename='smith');

           SAL
        ---------
           800
          1600
          1250
```

<u>USING SQL FUNCTIONS</u>

```
SQL> select nvl(sum(age),0) from student;
     NVL(SUM(AGE),0)
     ---------------
               113
SQL> select max(age) from student;
SQL> select to_char(121) from dual;
SQL> select months_between('01-may-83','01-jan-83') from dual;
     MONTHS_BETWEEN('01-MAY-83','01-JAN-83')
     ---------------------------------------
                                           4
```

1. Find the sum and avg sal of given employees

```
SQL> select sum(sal),avg(sal) from t;

SUM(SAL)  AVG(SAL)
--------- ---------
    1070    267.5
```

2. Select name from t order by substr(name, instr(name,',,'))

```
SQL> select name from t order by substr(name,instr(name,',,'))
ENAME
----------
ADAMS
ALLEN
```

3. Select translate('abcdabcdabcdabcdabcd','abcd','a') form dual

```
SQL> select translate('abcdabcdabcdabcdabcd','abcd','a') from dual;
TRANS
-----
aaaaa
```

4. Select name , sal lpad('x',round(sal/1000,0)) from t where sal is not null order by sal

```
SQL> select name, mark, lpad('x',round(mark/100,0)) from stud where
mark is not null order by mark;
NAME              MARK    LPAD('X',ROUND(MARK/100,0))
-------------------- --------- --------------------------
sita               56      X
rita               67      X
```

5. select to_char(sysdate,'dd/mm/yy') from dual;

```
TO_CHAR(
--------
19/12/02
```

6. select to_date('03/mar/03') from dual;

```
TO_DATE('
---------
03-MAR-03
```

PL SQL

a. To create PL/SQL code with exception

```
SQL>  declare
  2   r student.rollno%type;
  3  n student.name%type;
  4  begin
  5  select rollno,name into r,n from student where mark=34;
  6  exception
  7  when no_data_found then
  8  dbms_output.put_line('such an item not available');
  9  end;
 10  /
such an item not available

PL/SQL procedure successfully completed.
```

b. To create PL/SQL code using control statement

IF LOOP

```
SQL>  declare
  2   name student.name%type;
  3  begin
  4  select name into name from student where rollno=5;
  5  if name='anya' then
  6  update student set mark=90;
  7  end if;
  8  end;
  9  /
PL/SQL procedure successfully completed.
```

WHILE LOOP

```
SQL> declare
  2  a number:=0;
  3  j number:=0;
  4  begin
  5  while a<=100 loop
  6  j:=j+1;
  7  a:=a+20;
  8  end loop;
  9  dbms_output.put_line(a);
 10  end;
 11  /
120
```

PL/SQL procedure successfully completed.

<u>FOR LOOP</u>

```
SQL> declare
  2  i integer;
  3  begin
  4  for i in 1..3
  5  loop
  6  update stud set name='c' where rollno=4;
  7  end loop;
  8  end;
  9  /
```

PL/SQL procedure successfully completed.

c. To create cursor and work on that

```
SQL> select * from student;
```

NAME	AGE	DOB	DEPT	COND	NO
Len	27	01-MAY-74	Mech	exc	
Malar			CSE	GOOD	
Priya	21	24-APR-83	cse	Good	

3 rows selected.

```
SQL> declare
  2  cursor c1 is select name,dob,dept from student  order by dob desc;
  3  trank number :=0;
  4  n student.name%type;
  5  d student.dob%type;
  6  dep student.dept%type;
```

```
 7  begin
 8  open c1;
 9  loop
10  fetch c1 into n,d,dep;
11  exit when c1%notfound;
12  trank:=trank+1;
13  update student set no=trank where name=n and dept=dep;
14  end loop;
15  close c1;
16  end;
17  /
```

PL/SQL procedure successfully completed.

<u>VIEWS</u>

SQL> create view tv as select * from t;
View created.
SQL> select * from tv;

```
    NO NAME  SAL  HRA  DA NET  DOJ      DPN
--------- ----- --------- ------- --------- ---- --------- --------
    10 rash    320                 12-SEP-89  15
    20 jon     200                 30-AUG-98  30
```

i) Insert into view

SQL> insert into tv values (50,'wad',600,null,null,null,'18-mar-75',25);
1 row created.

ii) Delete from view

SQL> delete from tv where no = 10;
1 row deleted.

iii) Modify the view

SQL> update tv set sal = 400 where no = 20;
1 row updated.

<u>FUNCTION</u>

SQL> create or replace function fact (num number)
 2 return number is

```
 3  i number;
 4  f number;
 5  begin
 6  f:=1;
 7  for i in 1 .. num
 8  loop
 9  f := f*i;
10  end loop;
11  return f;
12  end;
13  /
```

Function created.

To execute the function

```
SQL> declare
 2  a number;
 3  begin
 4  a:=fact(&n);
 5  dbms_output.put_line('The factorial of the given number is '||a);
 6  end;
 7  /
Enter value for n: 4
old   4: a:=fact(&n);
new   4: a:=fact(4);
The factorial of the given number is 24
```

PL/SQL procedure successfully completed.

PROCEDURE

```
SQL> create or replace procedure fib (n number) is
 2  f1 number;
 3  f2 number;
 4  i number;
 5  c number;
 6  begin
 7  f1 := 0;
 8  f2 := 1;
 9  dbms_output.put_line(f1);
10  dbms_output.put_line(f2);
11  for i in 1 .. n
12  loop
13  c := f1+f2;
14  dbms_output.put_line(c);
15  f1 := f2;
```

```
16  f2 := c;
17  end loop;
18  end;
19  /
```

Procedure created.

To execute the procedure

```
SQL> exec fib(&n);
Enter value for n: 3
0
1
1
2
3
```

PL/SQL procedure successfully completed.

TRIGGERS

Trigger applied in same table:

```
SQL> create or replace trigger stu
  2  before insert on mark for each row
  3  begin
  4  :new.percentage := (:new.mark1 + :new.mark2 + :new.mark3)/3;
  5  end;
  6  /
```

Trigger created.

```
SQL> insert into mark values (110,40,50,60,'');
1 row created.
```

```
SQL> select * from mark;
```

REGNO	MARK1	MARK2	MARK3	PERCENTAGE
110	40	50	60	50

Trigger applied in two different tables:

First table   –   Parent Table (Primary key)
Second Table   –   Child Table (Foreign key)

```
SQL> select * from stu;  (First table)
```

```
     SID SNAME      AGE
-------- ----------      ---------
     1   senthil       24
     2   karthi        26
2 rows selected.

SQL> select * from dept;  (Second Table)
     SID DEPT
      1 cse
      2 spic
2 rows selected.

SQL> create or replace trigger del
  2  before delete on stu for each row
  3  begin
  4  delete from dept where sid = :old.sid;
  5  end;
  6  /
Trigger created.

SQL> delete from stu where sid=1;
1 row deleted.

SQL> select * from stu;
     SID SNAME      AGE
-------- ---------- ---------
      2    karthi        26
1 row selected.

SQL> select * from dept;
     SID DEPT
      2 spic
1 row selected.
```

To drop the trigger:

SQL> drop trigger <trigger_name>

Exercise:

Sample Database: Southern Railways

<u>I. Create tables for the following requirements</u>

Train:

Number, name, source, destination, start_time, reach_time

Passenger:

PNR No, Serial no., Name, Sex, Address, Age, Date of Journey, Status, kind of seat, seat no, Train number

<u>II. Insert necessary values into the tables.</u>

<u>III. Constraints</u>

1. Add a primary key constraint to train, Passenger.
2. Add a referential key constraint to passenger.
3. Add a check constraint to insert source and destination in 3 letters
4. Add a check constraint to enter a valid kind of seat while a Passenger record is added for a train.

<u>IV. Write queries for the following:</u>

1. List all train details.
2. List all passenger details.
3. Give a list of trains in ascending order of number.
4. Find out the number of passengers booked for a particular Train.
5. List the number of waiting lists in a train "x".
6. List the number of female passengers who have booked for trains.(train name wise).
7. List all three letter word passengers.
8. List the passenger names with a vowel in it.
9. List the trains from within a range of numbers.
10. List the details of trains from "x1" to "x2".
11. List the train numbers for which passengers had made some reservation.
12. List the train names for which reservation had not been made so for.
13. List the passengers whose journey is scheduled two weeks from today.
14. List the details of passengers who has reserved next to "Mr. x".
15. List the train names for which largest number of passengers have booked.

<u>V. Write Procedures for the following:</u>

1. Details of train numbers between a source and destination.
2. Details of all trains from one source.

3. PNR No. of a passenger for a given source and a destination.

VI. Write Functions for the following:

1. To the know Status of a passenger
2. Full journey time of any "x" train.

VII. Write a Cursor:

Retrieve the passenger details for "x" train number and given journey date.

VIII. Write a Trigger for the following:

1. When a train is cancelled passenger records should be deleted.
2. When a passenger record is inserted seat no. and status should be automatically updated.

IX. ALTER TABLE:

1. Add the following columns to train table either in the same table or by creating relationships.

No. of intermediate stations ,name of the intermediate station , arrival time, departure time, kind of seats, number of seats in each category, daily or weekly train, day.

X. Write nested queries for the following:

1. Train stopping in more than two intermediate station.
2. The previous station of the destination station of the passenger.
3. Name of weekly trains.
4. Name of trains from source to destination in "x" day.
5. Number of waiting list of passengers on Tuesday trains.
6. Passengers who are not booked for Friday trains.
7. List of train names, kind of seats and number available in each.
8. Details of currently available seats in any train.
9. Details of available seats in any train from a source to destination(source and destination can be any of the intermediate stopping).
10. Trains passing through any station between a time duration.

XI. Using LOBs

1. Add a column photo to the passenger and store it in the Db. Retrieve and show it in the front end. (Hint: Use tablespace while altering the table)

XII. Data report

Design a front end to give a report of passenger who have booked for Tuesday trains but don't travel through "X" station. Store the report as HTML or TEXT document
(Hint: Use Data report utility)