DATABASE MANAGEMENT SYSTEMS

A Quick Reference to laboratory course

(Prepared with reference to B.Tech(IT), MS (SE), BCA & MCA syllabus)

by

Prof.C.Ranichandra

&

Prof.N.C.Senthilkumar

School of Information Technology and Engineering

VIT University

Table of Contents

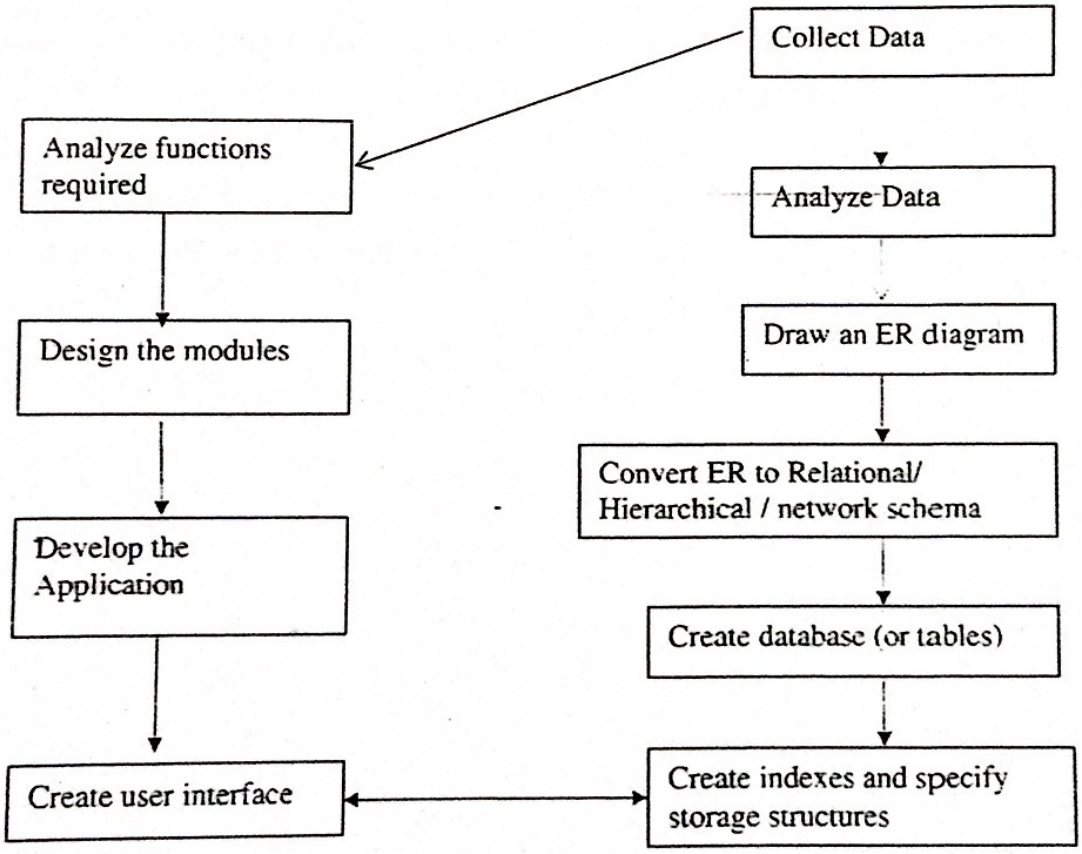
1. DESIGN OF A DBMS APPLICATION

A **database** is a collection of related data. Eg. Library (Book, Member, journal) database, Railway Reservation (Train, station, passenger, ticket) database.

A **Database Management System** is software or a set of programs which can create and manipulate the database. Eg. Oracle, Db2, Sybase, and Informix are some commercial DBMSs.

Railway Reservation System, Library Management System, Online quiz, Stock Management System are some of the DBMS applications which are developed using any commercial DBMS.

For developing a DBMS application collection, analysis and design of schemas at different level is important.



Phases in the development of DBMS application

A DBMS application consists of a back end and a front end. The front and back end are connected using ODBC(open database connectivity) or JDBC(Java Database Connectivity). Back end is the design and implementation of the database. Front end is the API(Application program Interface) or GUI (Graphical User Interface) which gives a friendly environment to query the database in the back end. A back end may be any database server like oracle, sql or DB2. The front end is developed using any IDE (Integrated Development Environment) available for languages like VB, VC++ and Java. For web applications the front end is even developed using HTML, Javascript, VBscript.

This manual gives elaborate explanation about the back end. That is developing the database using Structured Query Language. Developing the front is left to the students as self study on their own interest to apply it for the mini projects.

Some examples of making the front end in VB, VB.net, Java and connecting to oracle database are given in CHAPTER 5.

2. 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) varchar(length) ex: varchar2(5)	Variable length Character string	1 to 4000 bytes	'asqeq' '1234' 'ssf%.sd'

Number	number number(3) number (4,1)	Integer of any maximum range Only 3 digits Float of max 1 decimal place	Integer range 38 digits after decimal -84 to 127	Any number 123, 789 123.4,111.5 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 BLOB BFILE	Stores character Object with single byte Character Stores large binary objects(Graphics, video clips and sound files) Stores file pointers to LOBs managed by file systems external to the Db.	Max 4 GB	 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 : Create new tables with columns and constraints

a. simple creation

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

```
SQL> create table emp (
  2  id number(3),
  3  salary number(10,2),
  4  name varchar(30),
  5  dob date,
  6  addr varchar(20));
Table created.
```

b. constraints without constraint name

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

```
SQL> create table emp (
  2  id number(3),
  3  salary number(10,2),
  4  name varchar2(30),
  5  dob date not null,
  6  addr varchar2(20),
  7  primary key (id),
  8  unique(name));
Table created.
```

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>) );
```

```
SQL> create table dept (
  2  dno number(3),
  3  name varchar2(30),
  4  mgrssn number(3),
  5  start_date date,
  6  constraint pk1 primary key (id),
  7  constraint fk1 foreign key(mgrssn) references emp(id));
Table created.
```

Bank
Customer acc
BankBranch.
employeesal
Account
Employee
Customer

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> ) :
```

```
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 ck1 check (salary between 20000 and 30000),
```

8 constraint ck2 check (addr in ('chennai','madurai','Theni','Trichy')),

9 constraint ck2 check (dob > '1-jan-1980'));

Table created.

e. for sequence creation

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

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

Sequence created.

f. for ROLE creation

for grouping together a set of access rights

CREATE ROLE <role name>;

SQL> create role manger;

Role created.

SQL> create role employee:

Role created.

2. ALTER: -Alter the definition of an already created table

a. Add –to add new columns or constraints

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

ALTER TABLE <tablename> add constraint <constraint name> <primary reference/unique/check constraint>

SQL> alter table emp add (dnum number(3));

Table altered.

SQL> alter table emp add (DOJ date);

Table altered.

SQL> alter table emp add constraint fk1

2 foreign key (dnum) references dept(dno) on delete cascade:

Table altered.

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

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

ALTER TABLE <tablename> modify constraint <constraint name> <primary reference/unique/check/not null constraint>;

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

SQL> alter table emp modify (salary number(7,2)):

Table altered.

SQL> alter table dependent modify constraint fk1

2 foreign key (depname) references emp(fname) on delete cascade:

Table altered.

SQL> alter table dept modify name varchar(20) constraint nn1

not null:

Table altered.

c. drop –delete column or remove constraint

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

ALTER TABLE <tablename> drop constraint < constraint name > :

SQL> alter table emp drop column DOJ;

Table altered.

SQL> alter table dependent drop constraint fk1;

Table altered.

d. Rename- change the column name

ALTER TABLE <tablename> rename column < old column name> to <new column name> ;

SQL> alter table emp rename column id to ssn;

Table altered.

3. TRUNCATE

Removes the rows, not the definition

TRUNCATE TABLE <tablename>;

SQL> truncate table emp;

Table truncated.

4. DROP

Removes the rows and table definition

DROP TABLE <tablename>;

SQL> drop table emp;

Table dropped.

5. RENAME

Changes table name

RENAME < old tablename> to < new tablename>;

SQL> rename emp to employee;

Table renamed.

DML

1. INSERT

a. Inserting values from user

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

SQL> insert into emp values (12, 6000,'cra','01-jan-75','vellore');

1 row created.

b. Inserting interactively

INSERT INTO <tablename> VALUES(&<column name1> , &
<columnname2> ...);

SQL> insert into emp values (&id,&sal,'&name','&dob','&addr');

id: 190

sal: 6000

name: Ranichandra

dob: 24-09-75

addr: Sathuvachari

1 row created.

c. Inserting null values

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

SQL> insert into emp values (12,6000,'','01-jan-75','vellore');

1 row created. (give two single quotes without a space for a null value)

SQL> insert into emp values (12,6000,'cra',null,'vellore');

1 row created. (give null for a null value)

d. Inserting a sequence number

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

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

1 row created.

seq.nextval gives 101 since start value is 100. Another property of sequence is currentval

e. Inserting few columns of a table

INSERT INTO <tablename>(<column name1>, <column name2>) values (val1,val2);

SQL> insert into emp (ssn, name, dob) values (12, 'cra','01-jan-75');

1 row created.

2. SELECT

a. Simple select

SELECT * FROM < tablename>;

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

SQL> select * from emp;

SQL> select ssn,name from emp;

b. Alias name

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

SQL> select ssn social_security_number,name emp_name from emp;

c. With distinct clause

SELECT DISTINCT <col2> FROM < tab1>;

SQL> select distinct dob from emp;

d. With where clause

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

SQL> select ssn , name from emp where dob='01-jan-1985';

e. Select to create table

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

SQL> create table employee as (select * from emp);

SQL> create table emp1 as select ssn "emplno",name "ename",sal "salary" from emp;

Table created.

f. Copy only table definition

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

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

Table created

g. select to insert

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

SQL> insert into emp2 (select ssn,name,sal from emp);

3. UPDATE

a. Simple update

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

SQL> update emp set DOJ ='01-jan-2009';

b. Using where clause

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

SQL> update emp set name = 'puvi' where name = 'gpa';

c. Interactive updation

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

SQL> update emp1 set ename='&ename' where eno=&eno:

Enter value for ename: ee

Enter value for eno: 22

old 1: update emp1 set ename='&ename' where eno=&eno

new 1: update emp1 set ename='ee' where eno=22

1 row updated.

4. DELETE

a. Delete all rows

DELETE FROM <tablename>;

SQL> delete from emp;

b. Using where clause –delete specific rows

DELETE FROM <tablename> WHERE <conditions>;

SQL> delete from emp where DOJ < '01-jan-09';

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 to <savepoint name>;

Example-TCL

1. Make some insert
2. SQL> commit
3. Make some insert
4. SQL> savepoint my_ins;
5. Make some update;

6. SQL> rollback;

   insert at 3rd step and update at 5th step are undone (lost)

7. SQL> rollback to my_ins;

   Instead of previous rollback if this is given only update is undone.

DCL

1. GRANT : Grant privileges on your table to other user

a. Grant all privileges

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

SQL> grant all on emp to bce108;

b. Grant certain privileges

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

SQL> grant select on emp, dept to bce108;

c. Grant Execute privilege

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

SQL> grant execute on factorial to bce108;

d. Setting privileges to ROLE

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

GRANT <role name> TO <user name>;

SQL> grant select,insert,update to manager.

SQL> grant manger to bce108;

e. Grant reference privilege t

GRANT REFERENCE(<column name>) ON <table name> TO <username>;

SQL> grant reference(dno) on emp to bce108;

***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: Given privileges can be withdrawn using this command

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

SQL> revoke select on emp from bce108;

b. REVOKE <rolename> FROM <username>;

SQL> revoke manager from bce108;

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

```
SQL> select add_months('18-jan-2001',2) from dual;

ADD_MONTH
---------
18-MAR-01

SQL> select add_months('18-dec-2001',2) from dual;

ADD_MONTH
---------
18-FEB-02

SQL> select last_day('18-jan-2001') from dual;

LAST_DAY(
---------
31-JAN-01

SQL> select last_day('27-feb-2002') from dual;

LAST_DAY(
---------
28-FEB-02
```

```
SQL> select months_between('01-jan-2001','31-jan-2001') from dual;

MONTHS_BETWEEN('01-JAN-2001','31-JAN-2001')
-------------------------------------------
                                  -.9677419

SQL> select round(sysdate,'yy') from dual;

ROUND(SYS
---------
01-JAN-01

SQL> select round(to_date('25-aug-09'),'year') from dual;

ROUND(TO_
---------
01-JAN-10

SQL> select round(sysdate,'mon') from dual;

ROUND(SYS
---------
01-DEC-00

SQL> select round(sysdate,'day') from dual;

ROUND(SYS
---------
17-DEC-00

SQL> select trunc(sysdate,'year') from dual;

TRUNC(SYS
---------
01-JAN-09

SQL> select trunc(sysdate,'month') from dual;

TRUNC(SYS
---------
01-JAN-09
```

```
SQL> select trunc(sysdate,'day') from dual;

TRUNC(SYS
---------
25-JAN-09

SQL> select next_day('13-dec-2002','sunday') from dual;

NEXT_DAY(
---------
15-DEC-02

SQL> select next_day('13-dec-2002','wednesday') from dual;

NEXT_DAY(
---------
18-DEC-02

SQL> select next_day(sysdate,'sunday') from dual;

NEXT_DAY(
---------
17-DEC-00

SQL> select greatest ( '25-jan-2006', '23-feb-2009') from dual; ----wrong

GREATEST('2
----------
25-jan-2006

SQL> select greatest ( to_date('25-jan-2006'), to_date('23-feb-2009')) from dual;
GREATEST(
--------
23-FEB-09
```

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
ln(m)	Output based on BASE e

```
SQL> select abs(-34) from dual;

 ABS(-34)
---------
       34

SQL> select cos(0) from dual;

   COS(0)
---------
        1

SQL> select cosh(180) from dual;

COSH(180)
---------
7.447E+77


SQL> select exp(10) from dual;

  EXP(10)
---------
22026.466

SQL> select exp(1) from dual;

   EXP(1)
---------
2.7182818

SQL> select power(2,2) from dual;

POWER(2,2)
----------
         4

SQL> select mod(10,2) from dual;

MOD(10,2)
---------
        0
```

```
SQL> select mod(10,3) from dual;
MOD(10,3)
---------
        1


SQL> select ceil(87.7) from dual;

CEIL(87.7)
----------
        88

SQL> select ceil(87.3) from dual;

CEIL(87.3)
----------
        88

SQL> select floor(87.7) from dual;

FLOOR(87.7)
-----------
        87

SQL> select floor(87.3) from dual;

FLOOR(87.3)
-----------
        87

SQL> select round(34.645,2) from dual;

ROUND(34.645,2)
---------------
          34.65

SQL> select round(34.643,2) from dual;

ROUND(34.643,2)
---------------
          34.64

SQL> select round(98.645) from dual;

ROUND(98.645)
-------------
           99
```

```
SQL> select trunc(34.645,2) from dual;

TRUNC(34.645,2)
---------------
          34.64

SQL> select trunc(34.643,2) from dual;

TRUNC(34.643,2)
---------------
          34.64

SQL> select trunc(34.645) from dual;

TRUNC(34.645)
-------------
           34

SQL> select sqrt(100) from dual:

SQRT(100)
---------
       10

SQL> select ln(100) from dual:

  LN(100)
---------
4.6051702

SQL> select sign(3) from dual;

  SIGN(3)
---------
         1

SQL> select sign(-3) from dual;

  SIGN(-3)
---------
        -1
```

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
substr(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

SQL> select initcap('senthil') from dual;

INITCAP

Senthil

SQL> select lower('SENTHIL') from dual;

LOWER('
------
senthil

SQL> select upper('senthil') from dual;

UPPER('

SENTHIL

SQL> select ltrim('senthil','sen') from dual;

LTRI
----
thil

```
SQL> select ltrim('senthil','il') from dual;

LTRIM('
-------
senthil

SQL> select ltrim('sensenthil','sen') from dual;

LTRI
----
thil

SQL> select ltrim('sensenthil','SEN') from dual;

LTRIM('SEN
----------
sensenthil

SQL> select rtrim('senthil','thil') from dual;

RTR
---
sen

SQL> select translate('senthil','s','b') from dual;

TRANSLA
-------
benthil

SQL> select translate('senthil','se','b') from dual;

TRANSL
------
bnthil

SQL> select translate('senthil','sl','b') from dual;

TRANSL
------
benthi

SQL> select replace('senthil','sen','b') from dual;
REPLA
-----
bthil
```

```
SQL> select replace('senthil','sl','b') from dual;

REPLACE
-------
senthil

SQL> select substr('senthil',3,2) from dual;

SU
--
nt

SQL> select lpad('senthil',15,'@') from dual;

LPAD('SENTHIL',
---------------
@@@@@@@@senthil

SQL> select rpad('senthil',15,'@') from dual;

RPAD('SENTHIL',
---------------
senthil@@@@@@@@@

SQL> select decode(sname,'karthi','senthil') from stu where sid=2;

DECODE(
------
senthil

SQL> select (sid || ' is the no of ' || sname || ' whose age is '|| age) sasa from stu;

SASA
------------------------------------
2 is the no of karthi whose age is 26
3 is the no of satheesh whose age is 23

2 rows selected

SQL> select chr(65) from dual;

C
-
A
```

```
SQL> select length('ranichandra') from dual;

LENGTH('RANICHANDRA')
---------------------
                   11
```

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

```
SQL> select to_char(hiredate,'DDth month yyyy') from emp:

TO_CHAR(HIREDATE,'D
-------------------
17TH december  1980
02ND april     1981

SQL> select to_char(hiredate,'DDth fmmonth yyyy') from emp;

TO_CHAR(HIREDATE,'D
-------------------
17TH december 1980
02ND april 1981

SQL> select to_char(hiredate,'DDth Month yyyy') from emp;

TO_CHAR(HIREDATE,'D
-------------------
17TH December  1980
02ND April     1981

SQL> select to_char(hiredate,'ddth MONTH yy') from emp:

TO_CHAR(HIREDATE,
-----------------
17th DECEMBER  80
02nd APRIL     81

SQL> select to_char(sysdate,'dd/mm/yy') from dual:

TO_CHAR(
--------
27/01/09
```

```
SQL> select to_char(sysdate,'MONTH dd yy') from dual;

TO_CHAR(SYSDATE
---------------
JANUARY   27 09

SQL> select to_date('03/mar/03') from dual;

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

SQL> select to_date('27 JANUARY  09') from dual;

TO_DATE('
---------
27-JAN-09

SQL> select to_number('45') from dual;

TO_NUMBER('45')
---------------
            45
```

To store and retrieve Time data in date data type

```
SQL> insert into emp7(bdate) values (to_date('&dd','hh:mi:ss'));
Enter value for dd: 12:55:54
old   1: insert into emp7(bdate) values (to_date('&dd','hh:mi:ss'))
new   1: insert into emp7(bdate) values (to_date('12:55:54','hh:mi:ss'))
1 row created.
SQL> /
Enter value for dd: 10:17:17
old   1: insert into emp7(bdate) values (to_date('&dd','HH:mi:ss'))
new   1: insert into emp7(bdate) values (to_date('10:17:17','HH:mi:ss'))
1 row created.
```

SQL> select * from emp7;

ENO	ENAME	BDATE
hhhddddddddd	lkjhggss	
		01-JAN-10
		01-JAN-10

SQL> select to_char(bdate,'hh:mi:ss') from emp7;

TO_CHAR(

12:55:54

10:17:17

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

SQL> select uid from dual;

UID
66

SQL> select user from dual:

USER
SCOTT

SQL> select vsize('scott') from dual;

VSIZE('SCOTT')

5

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

SQL> select sal from emp;

SAL

800
1600
1250
2975
1250
2850
2450
3000
5000
1500
1100
SAL

950
3000
1300

14 rows selected.

SQL> select min(sal) from emp;

MIN(SAL)

800

```
SQL> select max(sal) from emp;

 MAX(SAL)
---------
     5000

SQL> select avg(sal) from emp;

 AVG(SAL)
---------
2073.21429

SQL> select sum(sal) from emp;

 SUM(SAL)
---------
    29025

SQL> select stddev(sal) from emp;

STDDEV(SAL)
-----------
 1182.50322

SQL> select variance(sal) from emp;

VARIANCE(SAL)
-------------
  1398313.87

SQL> select count(*) from emp;

 COUNT(*)
---------
       14

SQL> select count(sal) from emp;

COUNT(SAL)
----------
       14

SQL> seleet count(distinct sal) from emp;

COUNT(DISTINCTSAL)
------------------
                12
```

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>;

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

2. UNION ALL

Returns all rows by both the queries including duplicates

< query 1> UNION ALL< query2>;

```
SQL>  select id from emp
      union all
      select did from dependent;
```

3. INTERSECT

Returns common rows by both the queries

< query 1> INTERSECT < query2>;

```
SQL> select id from emp
     intersect
     select did from dependent;
```

4. MINUS

Returns distinct rows in the first query

< query 1> MINUS < query2>;

```
SQL> select id from emp
     minus
     select did from dependent;
```

JOINS

To combine the data 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 right operand in join condition)	Rows common in both tables and uncommon rows are padded with null values in left column
	Right outer join ((+) appended to left operand in join condition)	Vice versa
	Full outer join – Only key word used. **No' +' symbol**	Rows common in both tables and uncommon rows are padded with null values in both columns

Example-JOINS

```
SQL>  select * from emp, dept
 2   where dnum=dno;

SQL>  select * from emp e,emp s
 2   where e.ssn=s.superssn;

SQL> select eno,ename,emp1.dno,dname
 2  from emp1,dept1
 3   where emp1.dno=dept1.dno(+);

SQL> select eno,ename,emp1.dno,dname
 2  from emp1 left outer join dept1
 3  on emp1.dno=dept1.dno;

SQL> select eno,ename,emp1.dno,dname
 2  from emp1,dept1
 3  where emp1.dno(+)=dept1.dno;
```

```
SQL> select eno,ename,emp1.dno,dname
 2  from emp1 right outer join dept1
 3  on emp1.dno=dept1.dno;

SQL> select eno,ename,emp1.dno,dname
 2  from emp1 full outer join dept1
 3  on emp1.dno=dept1.dno;
```

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>;

```
SQL>  select * from emp where sal < (select sal from emp e.dept d where
e.ssn=d.mgrssn and d.name='SITE' );
```

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>);

```
SQL> select name from emp where DOJ=any(select start_date from dept);
```

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>);

```
SQL> select name from emp where DOJ > all (select start_date from dept);
```

c. IN

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

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

```
SQL> select name from emp where ssn in(select mgrssn from dept);
```

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>);

```
SQL> select name from emp where ssn not in(select mgrssn from dept);
```

e. EXISTS

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

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

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

f. NOT EXISTS

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

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

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

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>);

*** contains, except is implemented as not exists, minus in SQL

```
SQL>select e.fname from employee e
2 where not exists
3 (select id from project where did = 5
4 minus
5 select pid from works_on where essn = e.ssn);
```

i. GROUP BY CLAUSE

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

Rule:

Selected attributes and group by clause attributes should be same

Select <column1>, <column2> from <table name>

Where <conditions>

GROUP BY <column2>, <column1>;

```
SQL> select dnum from emp group by dnum;
SQL> select id, name, count(*)
2 from project p join works_on w on p.id = w.pid
3 group by id, name;
```

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>;

```
SQL>select id, name, count(*)
  2 from project p join works_on w on p.id = w.pid
  3 group by id, name
  4 having count(*) >= 2;
```

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;

```
SQL> select * from emp order by name desc;
```

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

Syntax:

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>;

Example-View

```
SQL> create view num_proj as
     1 (select id, name, count(*)
     2 from project p join works_on w on p.id = w.pid
     3 GROUP BY id, name);
     View created
```

*** This view is not updatable since it is taken from two tables

A join query can also be given as in the above syntax

<table name1> join <table name2> on <join condition>

<table name1> left outer join <table name2> on <join condition>

<table name 1> full outer join <table name 2> on <join condition>

```
SQL> create view some_emp as
     1 (select ssn,name from emp);
     View created
```

*** This view is updatable since it is derived from one table and contains the primary key. If emp does not contain any other not null column this view is updatable.

```
SQL> create view empofdept2 as
     1 (select * from emp where dnum=2);
     View created
```

SQL> grant select ,insert, delete on empofdept2 to manager;

*** A grant command can be used to give select permission on a subset of a table

SQL> create view tv as select * from emp;

View created.

i) Insert into view

SQL> insert into tv values (50,600,null,'18-mar-75','vlr',2);

1 row created.

ii) Delete from view

SQL> delete from tv where dnum= 10;

1 row deleted.

iii) Modify the view

SQL> update tv set sal = 400 where dnum = 20;

1 row updated.

ABSTRACT DATA TYPE

-Combining basic data types to create new datatypes for manipulating advanced databases

-ADT can be nested, refer other ADT

-ADT can be used to create object tables

To create a new data type

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..));

Methods can be added to the new type to bring object oriented concept in relational dbms.

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)> ….

Examples- ADT

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

SQL> alter table emp
 2  modify addr address;
Table altered
```

SQL> desc emp;

Name	Null?	Type
SSN		NUMBER(3)
SAL		NUMBER(10,2)
NAME		VARCHAR(20)
DOB		DATE
ADDR		ADDRESS
DNUM		NUMBER(3)

SQL> insert into emp values(1,1.2,'dsd','01-jan-05',addr('tn','India'),2);
1 row created.

SQL> select * from emp;

SSN	SAL	NAME	DOB	ADDR(S, C)
1	1.2	dsd	01-JAN-05	ADDRESS('arrr', 'sdfsf')

NESTED TABLES

Adding table-within a table-used for creating complex attributes or 1:N tables

First create type.

Then create nested table type.

A table space is required for storing nested table, which cannot be done in normal user permission.

CREATE TYPE <nested type>AS TABLE OF < type name>

CREATE TABLE < > (<name> <nested type>)

NESTED name STORE AS <new name>

To insert into nested table

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

……) …);

Example-Nested table

```
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))
```

VARYING ARRAYS

Range of values in a single row-multivalued attribute.

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;
;
```

Examples-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)
 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')
```

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>;

Example-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
```

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>, ----);

Example-Index

SQL> create index clus_emp on emp (dnum);

3. 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;

Example-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

```
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.
```

FOR LOOP

```
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.
```

CURSOR

Cursor is the pointer to the temporary area (context area) created for storing the data manipulated by a PL/SQL program. A cursor is created in a PL/SQL block or a procedure or function if a query returns more than one row.

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;
```

```
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.

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;

Example-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.

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;

Example-FUNCTION

```
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.
```

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 is done.

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>:

Example-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>

**An example trigger of the exercise

```
SQL> create or replace trigger passg
  1     after insert on passenger for each row
  2     declare
  3     allotted integer;
  4     M integer;
  5     begin
  6     select max(seatno) into allotted from passenger where trno=:new.trno
  7     and kind=:new.kind;
  8     select maxm into M from train_seat where trno=:new.trno
  9     and kind=:new.kind;
 10     if allotted = M then
 11     update passenger set seat_no=null, status='RAC' where trno=:new.trno;
 12     else
 13     update passenger set seat_no=allotted+1, status='reserved' where
        trno=:new.trno;
 14     end if;
 15     end;
 16     /
```

4. CATALOG QUERIES

A Catalog contains metadata. Metadata is nothing but data about data or description of the database. The description about the database is again stored as tables in the catalog. Users can query the catalog to find what constraints, procedures, functions they have made. Even the body of a trigger procedure or a function can be viewed from the catalog.

To know the constraint name and type

```
SQL>select CONSTRAINT_NAME, CONSTRAINT_TYPE     from user_constraints
where TABLE_NAME='DEPT';

CONSTRAINT_NAME C
------------------------------ -
PK_DEPT                P
SQL>select CONSTRAINT_NAME,CONSTRAINT_TYPE     from user_constraints
where TABLE_NAME='EMP7';

CONSTRAINT_NAME   C
------------------------------ -
PK1                    P
FK1                    R
```

To know the constraint column

```
SQL> select COLUMN_NAME . CONSTRAINT_NAME from
user_cons_columns where TABLE_NAME='EMP7';

COLUMN_NAME  CONSTRAINT_NAME
------------------------------
DNO          FK1
ENO          PK1
```

To know the sequence details

```
SQL> select SEQUENCE_NAME , LAST_NUMBER  from user_sequences;

SEQUENCE_NAME                  LAST_NUMBER
------------------------------ -----------
SSN                                    100
```

To know the trigger name, type and body

```
SQL> select  TRIGGER_NAME , TRIGGER_TYPE  from user_triggers;
TRIGGER_NAME                   TRIGGER_TYPE
------------------------------ ----------------
DEL1                           BEFORE EACH ROW
TOT                            BEFORE EACH ROW

SQL> select  TRIGGER_BODY  from user_triggers where
TRIGGER_NAME='DEL1';
TRIGGER_BODY
--------------------------------------------------------------------------------
begin
delete from emp1
where dno = :old.dno;
end;
```

To know the procedure, function names and body

```
SQL> select name from user_source where type='PROCEDURE';

NAME
------------------------------
INFO
INFO1
FACT
```

SQL> select name,type from user_source where type='PROCEDURE';

NAME	TYPE
INFO	PROCEDURE
INFO1	PROCEDURE

SQL> select text from user_source where name='GRT';

TEXT

```
function grt(a number)
return number is
m1 number;
m2 number;
begin
select c1,c2 into m1,m2 from stu where regno=a;
if m1>m2 then
return m1:
else
return m2:
end if:
end;
```

12 rows selected.

5. DATA BASE CONNECTIVITY EXAMPLES

DAO CONNECTIVITY

(In general)

```
Dim db As Database
Dim ds As Recordset
```

(In command button click)

```
Private Sub Command1_Click()
ds.MoveFirst
While Not ds.EOF
Text1.Text = ds(0)
ds.MoveNext
Wend
End Sub
```

(In form load)

```
Private Sub Form_Load()
Set db = OpenDatabase("dbms", False, False, "ODBC;UID=rani;pwd=cra;DSN=dbms")
Set ds = db.OpenRecordset("select id from emp")
End Sub
```

ADODB CONECTIVITY

(In General)

```
Dim CONN As adodb.Connection
Dim rs As adodb.Recordset
Dim CM As adodb.Command
```

(In Form load)

```
Set CONN = New adodb.Connection
CONN.ConnectionString = "PROVIDER=MSDAORA.1;USER
ID=cra;PASSWORD=ncs;DATA SOURCE=dbms;"
CONN.Open
```

(To select from database)

```
Set rs = New adodb.Recordset
rs.ActiveConnection = CONN
rs.Open "SELECT * FROM EMP"
rs.MoveFirst
While Not rs.EOF
Combo1.AddItem rs("regno")
rs.MoveNext
Wend
```

(To insert into database)

```
Set CM = New adodb.Command
CM.ActiveConnection = CONN
CM.CommandText = "insert into emp values('" & Trim(UCase(Combo1.Text))
& "','" & Trim(UCase(Text1.Text)) & "','" & Trim(UCase(Text2.Text)) & "','" &
Trim(UCase(Text5.Text)) & "','" & Trim(UCase(Combo4.Text)) & "','" &
```

```
Trim(Text7.Text) & "'," & Trim(Combo2.Text) & "'," & Format(Now, "DD-
MMM-YYYY") & "'," & Val(Text9.Text) & ")"
CM.Execute
```

(To Call a Procedure)

```
Set CM = New adodb.Command
CM.ActiveConnection = CONN
CM.CommandText = "CALL UPDVIEW('" & Trim(Combo2.Text) & "')"
CM.Execute
CM.CommandText = "commit"
CM.Execute
```

(To delete a record)

```
Set CM = New adodb.Command
CM.ActiveConnection = CONN
CM.CommandText = "DELETE FROM ADMIN WHERE regNO='" &
Trim(Combo1.Text) & "'"
Set cm1 = New adodb.Command
cm1.ActiveConnection = CONN
cm1.CommandText = "DELETE FROM ADMIN1 WHERE regNO='" &
Trim(Combo1.Text) & "'"
msg = MsgBox("Are you sure to delete the record?", vbQuestion + vbYesNo,
"V I T -Admissions")
If msg = vbYes Then
cm1.Execute
```

VISUAL BASIC. NET Oracle CONNECTIVITY

```
'Name space

Imports System.Data.OracleClient

'Class
Public Class Form1
   Inherits System.Windows.Forms.Form

 'Form Load
 ' To retrieve a record set in data grid

 Private Sub Form1_Load (ByVal sender As System. Object. ByVal e As
 System.EventArgs) Handles MyBase.Load

      Dim conn As OracleConnection = New OracleConnection ("Data source=tssccora;
      UID=rani; password=cra ;")
      Dim da As OracleDataAdapter = New OracleDataAdapter
                                    ("select * from pop", conn)
      Dim ds As DataSet = New DataSet
      da.Fill(ds)
      DataGrid1.DataSource = ds
      conn.Close()

   End Sub

 'Button click to insert a record

 Private Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
 Handles Button1.Click
```

```
        Dim conn As OracleConnection = New OracleConnection ("Data
        source=tssccora; UID=rani; password=cra ;")
        Dim qrp As String = New String ("insert into pop values
                              (" + TextBox1.Text + "," + TextBox2.Text + "')")
      Try
        conn.Open()
        Dim cmd As New OracleCommand (qrp, conn)
        cmd.ExecuteNonQuery()
        Dim da As OracleDataAdapter = New OracleDataAdapter
                                          ("select * from pop", conn)
        Dim ds As DataSet = New DataSet
        da.Fill(ds)
        DataGrid1.DataSource = ds
      Catch ex As Exception
        MsgBox(ex.Message)
        conn.Close()
      End Try
    End Sub

'Button click to delete a record

Private Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Button2.Click

        Dim conn As OracleConnection = New OracleConnection("Data
        source=tssccora;UID=rani;password=cra;")
        Dim qrp As String = New String("delete from pop
                        where no =" + TextBox1.Text + "")
        Try
          conn.Open()
```

```
        Dim cmd As New OracleCommand(qrp, conn)
        cmd.ExecuteNonQuery()
        Dim da As OracleDataAdapter = New OracleDataAdapter
                                        ("select * from pop", conn)

        Dim ds As DataSet = New DataSet
        da.Fill(ds)
        DataGrid1.DataSource = ds
      Catch ex As Exception
        MsgBox(ex.Message)
        conn.Close()
      End Try
    End Sub


    'Button click to update  a record

    Private Sub Button3_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Handles Button3.Click

        Dim conn As OracleConnection = New OracleConnection("Data
        source=tsccora;UID=05mcs065;password=05mcs065;")
        Dim qrp As String = New String("update pop set
          name= " + TextBox2.Text + " where no =" + TextBox1.Text + "")
        Try
          conn.Open()
          Dim cmd As New OracleCommand(qrp, conn)
          cmd.ExecuteNonQuery()
          Dim da As OracleDataAdapter = New OracleDataAdapter
                                          ("select * from pop", conn)
          Dim ds As DataSet = New DataSet
          da.Fill(ds)
```

```
        DataGrid1.DataSource = ds
    Catch ex As Exception
        MsgBox(ex.Message)
        conn.Close()
    End Try
  End Sub
End Class
```

Java Oracle CONNECTIVITY

```
import java.sql.*;

class database
{
        static String url="jdbc:odbc:dbms";
        static private Connection con;
        static Statement st;
        static ResultSet res;
        static String n,m;

public static void main(String args[])
{
        try
        {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con=DriverManager.getConnection(url,"ncs","ncs");
        }
        catch(ClassNotFoundException err){}
        catch(SQLException err){}
        try
        {
```

```
//statement object
st=con.createStatement();
res=st.executeQuery("select * from DEPT");
int i=0;
        while(res.next())
        {
        n=res.getString(1);
        m=res.getString(2);
        System.out.println("row" + i++ + " :" +n + " " + m);
        }
        st.close();
  // callable statement object
String qry="{ CALL pop(?,?,?) }";
   CallableStatement cst=con.prepareCall(qry);
cst.setInt(1,12);
cst.setInt(2,34);
cst.registerOutParameter(3,Types.VARCHAR);
cst.execute();
String s=cst.getString(3);
System.out.println(s);
   cst.close();
  // prepared statement object
  PreparedStatement ps=con.prepareStatement("select * from dept where
dnum=?");
  ps.setString(1,"ece");
  res=ps.executeQuery();
  res.next();
  System.out.println("Interactive query"+ res.getString(2));
  ps.close();     }catch(SQLException err){ System.out.println( err);}}
                                        }
```

6. Exercise

Sample Database: Southern Railways

I. Create tables for the following requirements

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

II. Insert necessary values into the tables.

III. Constraints

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.

IV. Write queries for the following:

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.

V. Write Procedures for the following:

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)

XIII. Connecting to external data

Design an Excel sheet to represent a real railways ticket format.
Develop a front end to access the data in Excel sheet and show it in DB Grid. (Use OLE)

**********Best of Luck**********