**P.A. AZIZ COLLEGE OF ENGINEERING AND TECHNOLOGY (PAACET)**

Green Hills, Karakulam P.O., Thiruvananthapuram, Kerala 695564



**LABORATORY MANUAL**

**ADVANCED ABMS LAB**

**SUBJECT CODE: 20MCA134**

**MASTER OF COMPUTER APPLICATION DEPARTMENT**

**MCA 2nd SEMESTER**

**NAME: ACHSAH MARIYAM GEORGE**

**REGISTER NO: PAA21MCA-2001**

**YEAR: ACADEMIC YEAR 2021-’23**

**P.A. AZIZ COLLEGE OF ENGINEERING AND TECHNOLOGY**

GREEN HILLS, KARAKULAM P.O., THIRUVANANTHAPURAM, KERALA 695564



**CERTIFICATE**

***This is to certified that Mrs. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Register No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_) has satisfactorily completed the course in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_as by the APJ Abdul Kalam University for \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ year, of semester \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ of Master of Computer Applications in the Academic year \_\_\_\_\_\_\_\_\_\_\_\_\_\_.***

**Head of the Department (MCA) Faculty-in-charge**

**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Internal Examiner External Examiner**

**CONTENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl. No.** | **Date** | **Name of Experiment** | **Page No.** | **Remarks** |
| 1 | 22.03.2022 | BASIC SQL QUERIES (CREATE, INSERT, SELECT) |  |  |
| 2 | 29.03.2022 | BASIC JOIN & AGGREGATE FUNCTIONS |  |  |
| 3 | 05.04.2022 | LIKE OPERATOR, INNER QUERIES & GROUP BY |  |  |
| 4 | 12.04.2022 | LABORATORY SYSTEM 1 |  |  |
| 5 | 19.04.2022 | LABORATORY SYSTEM 2 |  |  |
| 6 | 26.04.2022 | PL/SQL (TRIGGERS & PROCEDURES |  |  |
| 7 | 03.05.2022 | PROCEDURES & FUNCTIONS |  |  |
| 8 | 10.05.2022 | CURSOR & CAR DATABASE APPLICATION |  |  |
| 9 | 17.05.2022 | STUDENT DATABASE APPLICATION |  |  |
| 10 | 24.05.2022 | HOSPITAL DATABASE APPLICATION |  |  |
| 11 | 31.05.2022 | MONGODB |  |  |

# INTRODUCTION

A database management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The primary goal of DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

The DBMS a general-purpose software system that facilitates the process of

1. defining,
2. constructing and
3. manipulating the database for various application.

## NEED FOR DATABASE SYSTEMS

DBMS has been developed to overcome the disadvantages of conventional systems like file processing systems

1. Data redundancy and
2. Difficulty in accessing
3. Data isolation inconsistency data
4. Integrity problems
5. Atomicity problems
6. Concurrent-access anomalies
7. Security problems

**THE RELATIONAL DATABASE:**

A relational database uses tables to store information.

**DATABASE OBJECTS:**

A database object is anything defined and stored in a database. Some of the databases objects are:-tables, views, indexes etc

**TABLES**:

Table is the basic unit of data storage in a relational database management system. Every table has a table name and a set of columns and rows in which the data is stored. Each column is given a column name, a datatype or domain type, and a width.

## STRUCTURED QUERY LANGUAGE (SQL)

SQL is the standard relational database language. It includes features for defining the structure of the data, for modifying data in the database and for specifying security constraints. DDL and DML form part of SQL. Data Definition Language (DDL) provides commands for defining the relation schemas.

Data Manipulation Language (DML) includes commands to insert tuples , delete tuples, and modify tuples in the database.

SQL language has the following parts:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Transaction Control Language (TCL)
4. Data Control Language (DCL)

## BASIC SQL OPERATION

SQL statements can be grouped into four general categories: Data definition language (DDL), Data manipulation language (DML), Transaction control language (TCL), Data control language (DCL).

## DATA DEFINITION LANGUAGE (DDL)

The overall design of a database is called the database schema. A database schema is specified by a set of definitions that are expressed using a data definition language. It is used for structuring the database.

We use the data definition language statements CREATE, ALTER, DROP, TRUNCATE to create new objects, alter the structure of existing objects, completely remove objects from system, or delete all rows permanently from the table leaving the structure of the table.

## DDL COMMANDS

**1.1 CREATE TABLE COMMAND**

While naming a table,

1. First letter should be an alphabet
2. Reserved words cannot be used to name a table
3. Maximum length for a table name is 30 character
4. Table name should also be unique
5. Underscore, numerals and letters are allowed but not blank space and single quotes

**Syntax:** Create table<table name> (attribute-1 domaintype-1, attribute-2 domaintype-2….. attribute-n domaintype-n, integrity\_CONSTRAINT<constraint\_name-1><constraints>,

CONSTRAINT<constraint\_name-2> <constraints>, ……, CONSTRAINT<constraint name n> <constraints>);

**Example:** Create table manager (mnane varchar2(15), eno int, mid varchar2(10) primary key);

**Output:** Table created

**COMMAND FOR A TABLE FROM AN EXISTING TABLE**

A new table is created with attributes whose domain type and contents are same as the existing table’s.

**Syntax:** Create table <new table name> (<attribute1….attribute n>) as select attribute-1, attribute-2 … attribute-n from <existing table name>

**Example-1:** Create table branch\_new(branchname, branchno) as select bname,bno from branch\_exist;

**Output:** Table created

**Example-2:** Create table basset(bname,asset) as select bname,assetfrom branch ;

**Output:** Table created

**1.2 DESC COMMAND**

Command to see the structure of a table

**Syntax:** desc<table name>

**Example:** desc branch;

**Output:** Name Null? Type

-------------------- ------------ ---------------

BNAME CHAR(15)

BNO NOT NULL VARCHAR2(15)

MANAGER VARCHAR2(15)

ASSET NUMBER

**1.3** **ALTER TABLE COMMAND**

Command for modifying the fields in a table

This command is used to

1. Add a new column
2. Change the width of a data type or the data type itself
3. Include or drop integrity constraints

**Syntax:** - to add a new field into a table

alter table<table name> add(attribute-1 domintype, attribute-2 domaintype …attribute-n domaintypen);

**Example:** alter table branch add (city varchar(15), state char(15));

**Output:** Table altered.

**Syntax: -** To modify the domain type

alter table<table name>modify(attribute newdomaintype1, attribute2 newdomaintype2)

**Example:** alter table branch modify (state varchar(15),city char(15));

**Output:** Table altered

**Syntax: -** to add integrity constraint alter table add integrity constraint (attribute)

**Example:** alter table add primary key (bno); alter table add unique (name, manager);

**Output:** Table altered

**Syntax: -** to drop integrity constraint alter table<table name>drop integrity constraints;

**Example:** alter table branch drop primary key; alter table branch drop unique (name,manager);

**Output:** Table altered

**1.4. DROP TABLE COMMAND**

Command to delete the entire schema of the table (table with its content is deleted)

**Syntax:** drop table<table name>;

**Example:** drop table basset;

**Output:** Table dropped.

**1.5. TRUNCATE TABLE COMMAND**

This command is to delete the records in the table, while retaining the structure of the table.

**Syntax:** truncate table <table name> ;

**Example:** truncate table basset;

**Output:** Table truncated

## DATA MANIPULATION LANGUAGE (DML)

These commands are used to query and manipulate existing objects like tables. DML commands are: INSERT, SELECT, UPDATE, DELETE DML COMMANDS

**2.1 INSERT STATEMENT**

- To insert values into a table

To insert a single record

**Syntax:** Insert into <table name> (attribute-1,attribute-2,…attribute-n) values (value-1,value-2,…value-n);

**Example:** insert into branch (bname,bno,manager,asset) values (‘tvmbranch’,’b1’,’Smith’,1000);

**Output:** 1 row created.

Mass insertion

**Syntax:** insert into <table name> (attribute-1,attribute-2,…,attribute-n) values(‘&attribute-1’,’&attribute- 2’,…..’&attribute-n’);

**Example:** Insert into branch(bname,bno,manager) values (‘&bname’,’&bno’,’&manager’);

**Output:**

Enter value for name: klmbranch

Enter value for bno: b2

Enter value for manager: Jones

Old 1:insert into branch(bname,bno,manager)

Values(‘&bname’,’&bno’,’&manager’)

New 1:insert into branch(bname,bno,manager)

Values(‘klmbranch’,’b2’,’Jones’)

1 row created

**2.2 SELECT STATEMENT**

Select command is used to retrieve information stored in the table.

**Syntax:** select <column name> from <table name>;

**Example:** Select \* from branch; // Select all the columns of the table

***VARIATIONS OF SELECT STATEMENT***

1. To see the values of specific attributes

**Syntax:** Select attribute-1,attribute-2,….attribute-n from<table name>;

**Example:** Select bname,bno from branch;

1. Selecting distinct rows

Command to list the tuples in a table without duplication

**Syntax:** select distinct attribute-1,attribute-2,…..from<table name> where<condition>;

**Example:** Select distinct bname from branch;

1. Command to list the tuples in a table in a sorted order based on some attributes

**Syntax:** Select attribute-1,attribute-2,……attribute-n from<table name> order by attribute-1 asc; Select attribute-1,attribute-2,……attribute-n from<table name> order by attribute-1 desc;

**Example:** Select \* from branch order by asset asc;

Select \* from branch order by asset desc;

1. Select command with ‘where’ clause

To select specific rows or records which satisfy some specific conditions, we can use ‘where’ clause in the select command.

**Syntax:** select <attribute> from <table name> where <conditions>;

**Example:** Select bname from branch where bno=123;

1. Select command to create a table

User can include a ‘select’ clause in a create table command to create table and copy the records into it.

**Syntax:** create table <table name> as select <column names> from <existing table name>;

**Example:** create table branch\_new as select \* from branch;

‘where’ clause can be included to create table with those records which satisfy the ‘where’ condition

**Example:** create table branch\_new as select \* from branch where bno>200;

1. Select command to insert records

This command either create a table or consider an existing table prior to insertion, with the same structure as the other table and then performs the insertion command.

**Syntax:** insert in to <table name> (select column\_name from <existing table name>);

**Example:** insert into branch select \* from branch\_usa; insert into branch(branchname,branchno) select bname,bno from branch\_usa;

**2.3 UPDATE STATEMENT**

command to modify the value of records in a table

**Syntax:** update<table-name>set <change-value-of-attribute> where<condition>;

**Example:** update brach set city=’tvm’ where bno=’b1’;

**Output:** 1 row updated

update comfort set midnight = (select temperature from weather where city = 'manchester') where city

= 'walpole' and sampledate = to\_date('22-dec-1999','dd-mon-yyyy');

1 row updated.

**2.4 DELETE STATEMENT**

Command for deletion of rows

1. Deletion of specified rows

**Syntax:** Delete from<table name>where<condition>;

**Example:** delete from branch1 where asset>2000;

**Output:** 0 rows deleted

1. Deletion of entire rows

**Syntax:** delete from<table name>;

**Example:** Delete from branch1;

**Output:** 4 rows deleted

## 3. TRANSACTION CONTROL LANGUAGE (TCL)

The various commands in TCL are

1. COMMIT
2. SAVEPOINT
3. ROLLBACK

* 1. **COMMIT COMMAND**

Transaction changes can be made permanent to the database by the commit command.

**Syntax:** commit work; or commit; user can set auto commit option by using the command ‘set autocommit on’

* 1. **SAVEPOINT COMMAND**

These are markers to divide a very lengthy transaction into smaller ones. They are used to identify a point in a transaction to which we can later roll back. This is used along with rollback.

**Syntax**: savepoint savepoint\_id;

* 1. **ROLLBACK**

It is used to undo the work done in the current transaction. We can either rollback the entire transaction or to a savepoint so that statements after the savepoint are rolled back.

**Syntax:** rollback work;

or rollback;

Syntax to rollback to a particular stage in a transaction:

rollback to savepoint save\_pt; where save\_pt is the savepoint

## 4. DATA CONTROL LANGUAGE (DCL)

Data Control Language is used to manage user access to a database. DCL statements such as GRANT, REVOKE control access to database and affirm or revoke database transactions. It provides the user with privilege commands. The owner of the database can grant privileges or withdraw (revoke) privilege to other database.

**4.1 GRANT**

An object privilege specifies what a user can do with a database object such as a table, a sequence or a view. The privileges are granted using GRANT statement.

**Syntax:** grant privileges on <object\_name> to <username>;

**Example:** grant select, update on branch to new\_branch; grant all on customers to ashraf; // Grants all permissions on the table customers to the user who logs in as 'ashraf'.

grant select on customers to sunil; // Grants SELECT permission on the table customers to the user 'sunil'. User 'sunil' does not have permission to insert, update, delete or perform any other operation on customers table.

grant select on customers to sunil with grant option; // Enables user 'sunil' to give SELECT permission on customers table to other users

**4.2 REVOKE**

The privileges, once granted, can be taken away. REVOKE statement takes privileges not only from the grantee but also from users who are granted privileges by the grantee.

**Syntax:** revoke privileges on <object\_name> from <username>;

**Example:** revoke select, update on branch from mca; revoke delete on customers from ashraf

## OPERATORS IN SQL

The operators supported by SQL are

1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Pattern matching operators
5. **Arithmetic operators**

The various arithmetic operators are Function Definition value1 + value2 Addition value1 - value2 Subtraction value1 \* value2 Multiplication value1 / value2 Division

**Example:** select \* from pnum;

NAME NO1 NO2

---------- ---------------- ----------

os 60 70 nm 70 40 ds 60 90 select name, no1 + no2 as Additon, no1 - no2 as Substraction, no1 / no2 as Division, no1 \* no2 as Multiplication from pnum;

**Output:**

NAME ADDITON SUBSTRACTION DIVISION MULTIPLICATION

-------------------- -------------------------------------------------------------------------------------------

os 130 -10 .857142857 4200 nm 110 30 1.75 2800 ds 150 -30 .666666667 5400

1. **Comparison operators**

The various comparison operators are:

=, !=, <, >, <=, >=, between ( to check between any two values) in (to match with any of the values in the list) //not in – opposite of the in function result.

like (to match a character pattern) is null (to check whether it is Null) // is not null - opposite of the is null function result.

**Example:** Select bname from branch where bno > 100;

Select bname from branch where bno is null;

1. **Logical Operators**

These are used to combine the results of two conditions to produce a single result. The logical operators are: AND, NOT, OR

**Example:** Select regno from car where colour = 'blue' and regno = 1230

1. **Pattern matching operators**

Pattern matching operators are like, %, \_ (under score)

LIKE is useful for slightly more complex queries, in which a non-exact match is required. it can be used to select data in which the column entry matches a pattern containing the wildcards % or \_

 % matches 1 or more characters of any type  \_ matches any single character

**Example:** Select bname,asset from branch where bname like ‘tv%’; Select bname from branch where bname like ‘tv\_%’;

## FUNCTIONS IN SQL

These are built-in functions for performing operations using the Data Manipulation commands.

**Single-Row Functions**

Single-row functions return a single result row for every row of a queried table or view. These functions can appear in select lists, WHERE clauses, START WITH and CONNECT BY clauses, and HAVING clauses.

**Group Functions (Aggregate functions)**

Group Functions Returns a result based on a group of rows.

**Single-Row functions**

1. Numeric function
2. Character function
3. Date function
4. Conversion function

1. **Numeric Functions**

Numeric functions accept numeric input and return numeric values. Most numeric functions that return

NUMBER values that are accurate to 38 decimal digits.

**Function Definition**

ABS(value) ABSolute value

ACOS(value) Arc COSine of value, in radians

ASIN(value) Arc SINe of value, in radians

ATAN(value) Arc TANgent of value, in radians

CEIL(value) Numeric CEILing: the smallest integer larger than or equal to value

COS(value) COSine of value

COSH(value) Hyperbolic COSine of value

EXP(value) e raised to value EXPonent

FLOOR(value) Largest integer smaller than or equal to value

LN(value) Natural Logarithm of value

LOG(value) Base 10 LOGarithm of value

MOD(value, divisor) MODulus

NVL(value, substitute) substitute for value if value is NULL

POWER(value, exponent) value raised to an exponent POWER

ROUND(value, precision) ROUNDing of value to precision

SIGN(value) 1 if value is positive, –1 if negative, 0 if zero

SIN(value) SINe of value

SINH(value) Hyperbolic SINe of value

**Example:** select \* from emp;

ENO SAL COMM

---------- ---------- ---------- 101 4000 2.2 102 8000 1.76

select sal + nvl(comm,0) from emp;

SAL+NVL(COMM,0)

--------------------- 4002.2

8001.76

8000 0

select abs(-123) from dual;

ABS(-123)

------------

123

select ceil(1.9),ceil(2),ceil(1.3),floor(1.4),floor(1.9),floor(2) from dual;

CEIL(1.9) CEIL(2) CEIL(1.3) FLOOR(1.4) FLOOR(1.9) FLOOR(2)

---------- --------- ---------- ------------- ------------- ----------- 2 2 2 1 1 2

SQL> select mod(6,5),mode(6,2) from dual;

MOD(6,5) MOD(6,2)

---------- ----------

1 0 select power(3,2),sqrt(64),round(1.2),round(1.9) from dual;

POWER(3,2) SQRT(64) ROUND(1.2) ROUND(1.9)

------------- ---------- -------------- -------------- 9 8 1 2

1. **Character Functions**

These are functions operated on characters. Input will be character and output will be either character or number values. Character functions that return character values return values of the same datatype as the input argument. The length of the value returned by the function is limited by the maximum length of the datatype returned.

**Function Name Use**

|| Glues or concatenates two strings together. The | symbol is called a vertical bar or pipe.

CONCAT CONCATenates two strings together (same as | |).

INITCAP INITial CAPital. Capitalizes the first letter of a word or series of words.

INSTR Finds the location of a character IN a STRing.

LENGTH Tells the LENGTH of a string.

LOWER Converts every letter in a string to LOWERcase.

LPAD Left PAD. Makes a string a certain length by adding a certain set of characters to the left.

RPAD Right PAD. Makes a string a certain length by adding a certain set of characters to the right.

RTRIM Right TRIM. Trims all the occurrences of any one of a set of characters off of the right side of a string.

SOUNDEX Finds words that SOUND like the EXample specified.

SUBSTR SUBSTRing. Clips out a piece of a string.

UPPER Converts every letter in a string into UPPERcase.

**Examples** select \* from location;

CITY COUNTRY

--------------- -----------

athens greece trivandrum india southwales australia

select upper(city),initcap(country),length(city),city||country from location;

UPPER(CITY) INITCAP(COUNTRY) LENGTH(CITY) CITY||COUNTRY

--------------- ---------------------- ---------------- -----------------------

ATHENS Greece 6 athensgreece

TRIVANDRUM Inda 10 trivandrumindia

SOUTHWALES Australia 10 southwalesaustralia

select lower(city) from location;

LOWER(CITY)

---------------

athens trivandrum southwales

select concat(city,country) from location;

CONCAT(CITY,COUNTRY)

-----------------------------

athensgreece trivandruminda southwalesaustralia

select rpad(city,10,'\*') from location;

RPAD(CITY,10,'\*')

---------------------

athens\*\*\*\* trivandrum southwales

select lpad(city,15,'\*') from location;

LPAD(CITY,15,'\*')

---------------------

\*\*\*\*\*\*\*\*\*athens

\*\*\*\*\*trivandrum

\*\*\*\*\*southwales

select substr(city,2,4) from location;

SUBSTR(CITY,2,4)

---------------------

then riva outh select city,instr(city,'t') from location;

CITY INSTR(CITY,'T')

--------------- ------------------

athens 2 trivandrum 1 southwales 4

select \* from location where soundex(city)=soundex('athans');

CITY COUNTRY

--------------- --------------- athens Greece

soundex – has the unusual ability to find words that sound like other

1. **Date Functions**

Date functions operate on date (DATE), and interval (INTERVAL YEAR TO MONTH) values.

**Function Use**

ADD\_MONTHS(date,count) Adds count months to date.

GREATEST(date1,date2,date3,...) Picks latest date from list of dates.

LEAST(date1,date2,date3,...) Picks earliest date from list of dates.

LAST\_DAY(date) Gives date of last day of month that date is in.

MONTHS\_BETWEEN(date2,date1) Gives date2−date1 in months (can be fractional months).

NEXT\_DAY(date,’day’) Gives date of next day after date, where ‘day’is ‘Monday’, ‘Tuesday’, and so on.

**Example:**

select add\_months(sysdate,3) from dual;

ADD\_MONTH

--------------

27-JUL-08

select sysdate,last\_day(sysdate) "last",last\_day(sysdate) - sysdate "days left" from dual;

SYSDATE Last Days Left

------------- ----------- ---------------

30-MAY-01 31-MAY-01 1

select next\_day('02-feb-2001','tuesday') "next day" from dual;

NEXT DAY

-----------

06-FEB-2001

select months\_between (t\_date,n\_date ) from ex;

select greatest('10-mar-2008','2-apr-2008') from dual;

GREATEST('

--------------

2-apr-2008

SQL> select least('10-mar-2008','2-apr-2008') from dual;

LEAST('10-M

---------------

10-mar-2008

Note: SYSDATE

SYSDATE returns the current date and time set for the operating system on which the database resides.

select sysdate from dual;

SYSDATE

-----------

27-APR-08

1. **Conversion Functions**

Conversion functions convert a value from one datatype to another. Generally, the form of the function names follows the convention datatype TO datatype. The first datatype is the input datatype.

The second datatype is the output datatype.

The SQL conversion functions are:

**Function Use**

TO\_CHAR transforms a DATE or NUMBER into a character string.

TO\_DATE transforms a NUMBER, CHAR, or VARCHAR2 into a DATE.

TO\_NUMBER transforms a CHAR or VARCHAR2 into a NUMBER.

**Example:** select to\_char(sysdate,'yy>>MM>>dd') from dual;

TO\_CHAR(SY

---------------

08>>03>>07

select to\_char(sysdate,'dd month year') from dual;

TO\_CHAR(SYSDATE,'DDMONTHYEAR')

-------------------------------------------

07 march two thousand eight

select to\_date('01032008','mm dd yyyy') from dual;

TO\_DATE('

------------

03-JAN-08

**Group-Value Functions**

Group value functions return a single result row based on groups of rows, rather than on single rows. Group-Value functions can appear in select lists and in ORDER BY and HAVING clauses. They are commonly used with the GROUP BY clause in a SELECT statement.

**Function Definition**

AVG(value) Average of value for group of rows

COUNT(value) COUNT of rows for column

GROUPING(expression) Used in conjunction with ROLLUP and CUBE functions to detect

NULLs

MAX(value) MAXimum of all values for group of rows

MIN(value) MINimum of all values for group of rows

STDDEV(value) STanDard DEViation of all values for group of rows

SUM(value) SUM of all values for group of rows

VARIANCE(value) VARIANCE of all values for group of rows

**Example:**

select avg(salary) from employees;

select min(salary) from employees;

select max(salary) from employees;

select sum(salary) from employees;

select count(name) from employees; select count(\*) from emp;

select count(empno) from emp;

Count(distinct attribute) – count all rows except null valued rows but eliminates duplicates while counting. select count(distinct empno) from emp;

**Group By**

GROUP BY is used in conjunction with aggregating functions to group the results by the unaggregated columns.

**Example:** select deptno,sum(sal) from emp group by deptno;

DEPTNO SUM(SAL)

----------- ---------------

1 11000 2 5000

select deptno,sum(sal) from emp group by deptno order by deptno desc;

DEPTNO SUM(SAL)

------------ --------------- 2 5000

1 11000

**Group By With Having**

HAVING is used to perform an action on groups created by GROUP BY similar to that of the WHERE clause on rows in a basic SQL statement. The WHERE clause limits the rows evaluated. The HAVING clause limits the grouped rows returned.

**Example:** select deptno,sum(sal) from emp group by deptno having sum(sal) > 9000;

DEPTNO SUM(SAL)

---------- -----------

1 11000

**SET OPERATORS**

The four set operators union, union all, intersects and minus allows to serially combine more than one select statements. Although more than one select statement will then be present, only one result set is then returned. Operators Use union all selects all rows from all select statements union union all is very similar to union, however, it dismisses duplicate rows found across different select statements intersect returns the rows that are found in all select statements minus minus returns all rows from the first select statements except

those who are duplicated in a following select statement

**Example:**

Select ename, empno from emp Union all Select mname, eno from manager;

Select ename,empno from emp Union Select mname,eno from manager;

Select ename,empno from emp Intersect Select mname,eno from manager;

Select ename,empno from emp Minus Select mname,eno from manager;

**JOIN OPERATION**

An SQL JOIN clause combines records from two tables in a relational database, resulting in a new, temporary table, sometimes called a "joined table".

**Inner join**

1. Equi-join
2. Natural join
3. Cross join

**Outer joins**

1. Left outer join
2. Right outer join
3. Full outer join

An inner join requires each record in the two joined tables to have a matching record. An inner join essentially combines the records from two tables (A and B) based on a given join-predicate.

**Example** of an explicit inner join

select \* from employee inner join department on employee.departmentid =department.departmentid

**Example** of an implicit inner join

select \* from employee, department where employee.departmentid = department.departmentid

**Types of inner joins**

## Equi-join

An equi-join is a specific type of comparator-based join, or theta join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

The query shown above has already provided an example of an equi-join

select \* from employee inner join department on employee.departmentid = department.departmentid

**Natural join**

A natural join offers a further specialization of equi-joins. The join predicate arises implicitly by comparing all columns in both tables that have the same column-name in the joined tables. The resulting joined table contains only one column for each pair of equally-named columns.

The above sample query for inner joins can be expressed as a natural join in the following way:

select \* from employee natural join department

## Cross join

A cross join, cartesian join or product provides the foundation upon which all types of inner joins operate. A cross join returns the cartesian product of the sets of records from the two joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or join-condition is absent in statement.

If A and B are two sets, then cross join = A × B.

Example of an explicit cross join: select \* from employee cross join department Example of an implicit cross join: select \* from employee, department;

**Outer joins**

An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table(s) one retains the rows from (left, right, or both).

**Left outer join**

The result of a left outer join for tables A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B). This means that if the ON clause matches 0 (zero) records in B, the join will still return a row in the result—but with NULL in each column from B. This means that a left outer join returns all the values from the left table, plus matched values from the right table (or NULL in case of no matching join predicate).

**Example:** select \* from employee left outer join department on employee.departmentid = department.departmentid

**Right outer join**

A right outer join closely resembles a left outer join, except with the tables reversed. Every record from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in A.

A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).

**Example:** select \* from employee right outer join department on employee.departmentid = department.departmentid

**Full outer join**

A full outer join combines the results of both left and right outer joins. The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

**Example:** select \* from employee full outer join department on employee.departmentid = department.departmentid

## SUBQUERIES

Nesting of queries, one within another is termed as a subquery. Subqueries allow to realize complex queries which would normally require several queries with storage of the intermediate results.

**Example:** Determine the name, sex and age of the oldest student.

select nometu, cdsexe, ( current\_date-dtnaiss)/365 as age from etudiant where (current\_date-dtnaiss) /365 = ( select max(( current\_date-dtnaiss) /365) from etudiant );

Predicates - IN, ANY, ALL, EXISTS

A subquery can return a subset of zero to n values. According to the conditions which one wants to express, one can use the predicates IN, ANY, ALL or EXISTS.

Predicate Meaning IN The comparison operatror is the equality and the logical operation between values is OR.

ANY Allows to check if at least a value of the list satisfies condition.

ALL Allows to check if condition is realized for all the values of the list.

EXISTS If the subquery returns a result, the value returned is True otherwise the value returned is False.

**Example:** Display the marks of the student number 1 which are equal to the marks of the student number 2.

select note from notes where numetu=1 and note in ( select note from notes where numetu=2);

**Example:** Display the marks of the student number 1 which are greater than the marks of the student number 2.

select note from notes where numetu=1 and note > any ( select note from notes where numetu=2);

**Example:** Display the marks of the student number 1 which are lower than all the marks of the student number 9.

select note from notes where numetu=1 and note < all ( select note from notes where numetu=9);

**Example:** Display all information on the students who do not have any mark.

select \* from etudiant e where not exists ( select numetu from notes where numetu=e.numetu);

## VIEWS

A view is a customized presentation of the data from one or more tables. Views derive their data from tables on which they are based, which are known as base tables. All operations performed on a view actually affect the base tables of the view.

Views can be used for several purposes:

1. To give you an additional level of table security by restricting access to a predetermined set of table rows and columns.
2. To hide data complexity. Oracle 8 databases usually include many tables, and by creating a view combining information from two or more tables,you make it easier for other users

to access information from your database.

1. To present the data in a different perspective from that of the base table.Views provide a means to rename columns without affecting the base table.
2. To store complex queries.

**Syntax:** create [or replace] view <view-name> [(<column(s)>)] as <select-statement> [with check option [constraint <name>]][with read only];

**Example:** create view DEPT20 (ENAME, JOB, ANNUAL SALARY) as select ENAME, JOB, SAL \* 12 from EMP where DEPTNO = 20; OUTPUT:

View created.

To list the view:

Select \* from DEPT20;

ENAME JOB SALARY

------------ ------------- -----------------------

SMITH 111 2000

HAYES 123 7000

JOHNES 221 8000

TURNE 241 1000

**Creating a read only view**

To prevent modifications to the base tables via view, user can use with read only clause of the create view command.

Create or replace view RAIN as select \* from TROUBLE with read only;

**Dropping a VIEW**

**Syntax:** drop view view\_name;

**example:** drop view rain;

**INTEGRITY CONSTRAINTS IN CREATE TABLE COMMAND**

A constraint is a rule that restricts the values for one or more columns in a table. Constraints are of two types. Table constraints and Column constrains.

Some of the constraints are:

1. **Primary key:** It is the minimal combination of attribute in the table, which is used to uniquely identify the rows of the table.
2. **Foreign key:** This is a key to join a table to another table. Foreign in a table is an attribute, which is a primary key of another table. If an attribute A1 in table R1 is a primary key in another table R2, the A1 is the Foreign key in R1 and A1 refers R2.
3. **Unique:** If some attribute (column) combination is said to be unique, then their combination cannot have duplicate values in the table.
4. **Not Null:** If a column is assigned to be not null then it cannot have null values.
5. **Check:** This constraint is used to specify some condition for the columns.
6. **Default:** This constraint is used to specify a default value for a specific column.

**Examples:** primary key create table emp(empno number(3 ) primary key, ename varchar2(20)); create table emp(empno number(3),ename varchar2(20), primary key(empno));

NOT NULL: create table emp(empno number(3), ename vrchar2(15) not null);

unique: create table emp(empno number(3) unique,ename varchar(20))

create table emp (empno number(3),ename varchar(20),unique(empno)));

check: create table emp(empno number(3),sal number(8) check(sal>0), ename varchar(20));

create table emp5(empno number(3),sal number(8), ename varchar(20),check(sal>0));

foreign key: cretate table maneger table(mname varchar(15), eno int references emp(empno),mid varchar(10)primary key);

default create table worker(wname varchar(15) default (‘not entered’), dob date,salary numeric(6,2) default(3000));

USAGE OF CONSTRIANT keyword for implementing integrity constraints in a CREATE TABLE command

**Syntax:** Create table<table name>(<attribute-1 domain type-1,….attribute-n domain type-n,

CONSTRAINT<constaintname1>,constraints>,….CONSTRAINT<constraintname2><constraints>, …CONSTRAINT<constaintnamren><constraints>);

**Example:** Create table borrower(loanno varchar(10),amount number(10), CONSTRAINT chk1

check(amount>=0), CONSTRIAINT conpri primary key(loanno));

Note: Here the name of constraint is chk1

To remove the constraint chk1 command is:- alter table borrower drop constraint chk1;

## TRIGGERS

Trigger defines an action the database should take when some database-related event occurs. Triggers are executed by the database when specific types of data manipulation commands are performed on specific tables. Such commands may include inserts, updates, and deletes. Updates of specific columns may also be used as triggering events.

## Types of Triggers

A trigger’s type is defined by the type of triggering transaction and by the level at which the trigger is executed.

## Row-Level Triggers

* Row-level triggers execute once for each row in a transaction.
* Row-level triggers are created using the for each row clause in the create trigger command.

## Statement-Level Triggers

Statement-level triggers execute once for each transaction. Statement-level triggers are the default type of trigger created via the create trigger command.

The general **syntax** of the trigger is:

Create [or replace] trigger Trigger\_name

[Before / after] [Insert/update/delete]

On Table\_name

[For each row]

[When condition]

Declare

Declaration statements

Begin

Executable statements

Exception

Exception handling statements

End;

The **before** and **after** keywords indicate whether the trigger should be executed before or after the triggering transaction.

The **delete, insert**, and **update** keywords (the last of which may include a column list) indicate the type of data manipulation that will constitute a triggering event. When referring to the old and new values of columns, you can use the defaults (“old” and “new”) or you can use the **referencing** clause to specify other names.

When the **for each row** clause is used, the trigger will be a row-level trigger; otherwise, it will be a statement-level trigger.

The **when** clause is used to further restrict when the trigger is executed. The restrictions enforced in the when clause may include checks of old and new data values.

**Example:**

Create or replace trigger trig

Before insert on loan

For each row

Begin

If inserting then

Dbms\_output.put\_line(‘value inserted’);

End if;

End;

----------------------------------------------------------------------------------------------

SQL> insert into employee values('john',122,2000); value insetred

1 row created.

## PL/SQL(PROCEDURAL LANGUAGE/ STRUCTURED QUERY LANGUAGE)

**PL/SQL** stands for Procedural Language/SQL. PL/SQL is super set of the Structured Query Language. Using PL/SQL user can do things like codify business rules through the creation of stored procedures and packages, trigger database events to occur, or add programming logic to the execution of SQL commands.

PL/SQL code is grouped into structures called blocks.

* **Anonymous block** - If the block of PL/SQL code is not given a name then it is called anonymous block.
* **Named block** - block with a code with a name. Procedures and Functions are examples.

## Features of PL/SQL

▪ Block(modular) structures

▪ Flow-control statements and loops

▪ Variables, constants and types

▪ Structured data

▪ Customized error handling

▪ Allows to store compiled code directly in the database

PL/SQL datatypes include all of the valid SQL datatypes as well as complex datatypes based on query structures.

**Section Description**

Declaration Defines and initializes the variable and cursors used in the block

Executable commands Uses flow-control commands(such as if commands and assign values to the declared variables)

Exception Handling Provides customized handling of error conditions

## Structure of a PL/SQL block

declare

<declarations section> begin

<executable commands> exception

<exception handling>

end; /

## Declarations Section

The Declarations section begins a PL/SQL block. The Declarations section starts with the declare keyword, followed by a list of variable and cursor definitions. User can define variables to have constant values, and variables can inherit datatypes from existing columns and query results.

**Example:**

declare pi constant NUMBER(9,7) := 3.1415926; radius INTEGER(5); area NUMBER(14,2); begin

<executable commands>

end;

## %TYPE and %ROWTYPE

Both %TYPE and %ROWTYPE are used to define variables in PL/SQL as it is defined within the database. If the datatype or precision of a column changes, the program automically picks up the new definition from the database without having to make any code changes. The %TYPE and %ROWTYPE constructs allows programs to adapt as the database changes to meet new business needs.

## %TYPE

%TYPE is used to declare a field with the same type as that of a specified table's column.

**Example:**

DECLARE

v\_EmpName emp.ename%TYPE;

BEGIN

SELECT ename INTO v\_EmpName FROM emp WHERE ROWNUM = 1;

DBMS\_OUTPUT.PUT\_LINE('Name = ' || v\_EmpName);

END; /

## %ROWTYPE

%ROWTYPE is used to declare a record with the same types as found in the specified database table, view or cursor.

**Examples:**

DECLARE

v\_emp emp%ROWTYPE; BEGIN

v\_emp.empno := 10;

v\_emp.ename := 'JOY';

END; /

## Executable Commands Section

In the Executable Commands section, user can manipulate the variables and cursors declared in the Declarations section of your PL/SQL block. The Executable Commands section always starts with the keyword begin.

**Example:**

declare pi constant NUMBER(9,7) := 3.1415926; radius INTEGER(5); area NUMBER(14,2);

begin radius := 3;

area := pi\*power(radius,2); insert into AREAS values (radius, area); end;

/

## Exception Handling Section

When user-defined or system-related exceptions (errors) are encountered, the control of the PL/SQL block shifts to the Exception Handling section. Within the Exception Handling section, the when clause is used to evaluate which exception is to be “raised”—that is, executed.

If an exception is raised within the Executable Commands section of your PL/SQL block, the flow of commands immediately leaves the Executable Commands section and searches the Exception Handling section for an exception matching the error encountered. PL/SQL provides a set of system-defined exceptions and allows you to add your own exceptions.

Every exception in PL/SQL has an error number and error message; some exceptions also have names.

## Predefined Exception

**Syntax:**

Begin

Sequence of statements;

Exception

When <exception\_name> then

Sequence of statements;

End

Declaring Exceptions

declare mark student.mark1%type;

begin select mark1 into mark from student where mark1>95;

dbms\_output.put\_line('current' : || mark ); exception

when no\_data\_found then dbms\_output.put\_line('no such student'); end; /

## Raising Exceptions

An exception can be raised in three ways:

▪ By the PL/SQL runtime engine

▪ By an explicit RAISE statement in your code

▪ By a call to the built-in function RAISE\_APPLICATION\_ERROR

The **syntax** for the RAISE statement is:

RAISE exception\_name;

declare lo\_mark exception; mark student.mark1%type;

begin select mark1 into mark from student where sname='johny'; if mark < 50 then raise lo\_mark; end if;

exception when lo\_mark then dbms\_output.put\_line('student has got less mark'); end;

/

**Example:**

Raise\_application\_error

It is used to create user defined error messages.

**Syntax:**

raise\_application\_error( error number, error message); where , error number – between –20,000 and –20,999.

Error message – text associated with this error.

**Example:**

exception lo\_bal then raise\_application\_error(-20001, ‘ balance is low’);

--

end;

## Conditional Logic

Within PL/SQL, user can use if, else, and elsif commands to control the flow of commands within the Executable Commands section

**Syntax:**

if <some condition> then <some command> elsif <some condition>

then <some command> else <some command>

end if;

**Example:**

if area >30 then insert into AREAS values (rad\_val.radius, area); end if;

## LOOPS

User can use loops to process multiple records within a single PL/SQL block. PL/SQL supports three types of loops

**Simple loops:** A loop that keeps repeating until an exit or exit when statement is reached within the loop. The loop is started by the loop keyword, and the exit when clause determines when the loop should be exited. An end loop clause signals the end of the loop.

**Example:**

declare pi constant NUMBER(9,7) := 3.1415926; radius INTEGER(5); area NUMBER(14,2);

begin radius := 3; loop

area := pi\*power(radius,2); insert into AREAS values (radius, area); radius := radius+1; exit when area >100; end loop;

end;

/

## FOR Loops

A loop that repeats a specified number of times. The FOR loop’s start is indicated by the keyword for, followed by the criteria used to determine when the processing should exit the loop. Since the number of times the loop is executed is set when the loop is begun, an exit command isn’t needed within the loop.

**Example:**

declare pi constant NUMBER(9,7) := 3.1415926;

radius INTEGER(5);

area NUMBER(14,2);

begin for radius in 1..7 loop area := pi\*power(radius,2);

insert into AREAS values (radius, area);

end loop;

end;

/

## WHILE Loops

In a WHILE loop, the loop is processed until an exit condition is met. Instead of specifying the exit condition via an exit command within the loop, the exit condition is specified in the while command that initiates the loop.

**Example:**

declare pi constant NUMBER(9,7) := 3.1415926; radius INTEGER(5); area NUMBER(14,2);

begin radius := 3; while radius<=7 loop

area := pi\*power(radius,2); insert into AREAS values (radius, area); radius := radius+1; end loop;

end;

/

## Cursor

A cursor is handle or pointer to the context area. The PL/SQL program can control the context area by using the cursor. There can be either static cursors, whose SQL statement is determined at compile time, or dynamic cursors, whose SQL statement is determined at runtime.

**Types of Cursors**

PL/SQL uses two types of cursors: explicit and implicit

## Explicit Cursors

Explicit cursors are SELECT statements that are DECLAREd explicitly in the declaration section of the current block or in a package specification. Use OPEN, FETCH, and CLOSE in the execution or exception sections of your programs.

## Declaring explicit cursors

To use an explicit cursor, first declare it in the declaration section of a block or package. There are three types of explicit cursor declarations: A cursor without parameters *cursor company\_cur is select company\_id from company;* A cursor that accepts arguments through a parameter list *cursor company\_cur (id\_in in number) is select name from company where company\_id = id\_in;*

A cursor **header** that contains a RETURN clause in place of the SELECT statement:

*cursor company\_cur (id\_in in number) return company%rowtype is select \* from company;*

## Opening explicit cursors

To open a cursor, use the following syntax: *OPEN cursor\_name [(argument [,argument ...])];*

where cursor\_name is the name of the cursor as declared in the declaration section. The arguments are required if the definition of the cursor contains a parameter list.

DECLARE

CURSOR c1 IS SELECT ename, job FROM employee WHERE sal < 7000;

...

BEGIN OPEN c1;

...

END;

## Fetching from explicit cursors

The FETCH statement places the contents of the current row into local variables. To retrieve all rows in a result set, each row needs to be fetched.

The **syntax** for a FETCH statement is:

*FETCH cursor\_name INTO record\_or\_variable\_list;* where cursor\_name is the name of the cursor as declared and opened. Closing explicit cursors

The **syntax** of the CLOSE statement is: *CLOSE cursor\_name;*

where cursor\_name is the name of the cursor declared and opened.

**Explicit cursor attributes**

**Attribute Description**

%ISOPEN TRUE if cursor is open. FALSE if cursor is not open.

%FOUND Returns true if the last fetch returned a row else a false

%NOTFOUND Returns true if the last fetch did not return a row else a false

%ROWCOUNTThe number of rows fetched from the cursor. INVALID\_CURSOR if cursor has been CLOSED

**Example:**

declare pi constant NUMBER(9,7) := 3.1415926;

area NUMBER(14,2); cursor rad\_cursor is select \* from RADIUS\_VALS;

rad\_val rad\_cursor%ROWTYPE;

begin open rad\_cursor;

loop

fetch rad\_cursor into rad\_val;

exit when rad\_cursor%NOTFOUND;

area := pi\*power(rad\_val.radius,2);

insert into AREAS values (rad\_val.radius, area);

end loop;

close rad\_cursor; end;

/

## Implicit Cursors

Whenever a SQL statement is directly in the execution or exception section of a PL/SQL block, you are working with implicit cursors. These statements include INSERT, UPDATE, DELETE, and SELECT INTO statements. Unlike explicit cursors, implicit cursors do not need to be declared, OPENed, FETCHed, or CLOSEd.

SELECT statements handle the %FOUND and %NOTFOUND attributes differently from explicit cursors. When an implicit SELECT statement does not return any rows, PL/SQL immediately raises the NO\_DATA\_FOUND exception and control passes to the exception section. When an implicit SELECT returns more than one row, PL/SQL immediately raises the TOO\_MANY\_ROWS exception and control passes to the exception section.

**Implicit cursor attributes**

**Attribute Description**

%ISOPEN Always false as it is closed immediately after executing its associated SQL statement

%FOUND Returns true if the last DML statement returned a row else a false

%NOTFOUND Returns true if the last DML statement did not return any row else a false

%ROWCOUNT Returns the total number of rows returned.

## Cursor FOR loop

In a Cursor FOR loop, the results of a query are used to dynamically determine the number of times the loop is executed. In a Cursor FOR loop, the opening, fetching, and closing of cursors is performed implicitly.

**Example:**

declare pi constant NUMBER(9,7) := 3.1415926;

area NUMBER(14,2); cursor rad\_cursor is select \* from RADIUS\_VALS;

begin for rad\_val in rad\_cursor loop

area := pi\*power(rad\_val.radius,2); insert into AREAS values (rad\_val.radius, area); end loop;

end;

/

## Ref Cursors

To create cursor variables, you take two steps. First, you define a REF CURSOR type, then declare cursor variables of that type. You can define REF CURSOR types in any PL/SQL block, subprogram, or package using the syntax

*TYPE ref\_type\_name IS REF CURSOR RETURN return\_type;*

## Declaring Cursor Variables

Once you define a REF CURSOR type, you can declare cursor variables of that type in any

PL/SQL block or subprogram.

In the following example, you declare the cursor variable dept\_cv:

DECLARE

TYPE DeptCurTyp IS REF CURSOR RETURN dept%ROWTYPE;

dept\_cv DeptCurTyp; -- declare cursor variable

## Opening a Cursor Variable

The OPEN-FOR statement associates a cursor variable with a multi-row query, executes the query, and identifies the result set.

Here is the **syntax:**

OPEN {cursor\_variable\_name | :host\_cursor\_variable\_name} FOR select\_statement;

where host\_cursor\_variable\_name identifies a cursor variable declared in a PL/SQL host environment such as an OCI or Pro\*C program.

## Closing a Cursor Variable

The CLOSE statement disables a cursor variable. After that, the associated result set is undefined.

Here is the **syntax**:

CLOSE {cursor\_variable\_name | :host\_cursor\_variable\_name);

**Varying Arrays**

A varying array allows you to store repeating attributes of a record in a single row.

## Creating a Varying Array

SQL> create or replace type mark as varray(5) of number;

Type created.

SQL> create table varr(no number,marks mark);

Table created.

SQL> desc varr;

Name Null? Type

---------------- -------- --------------------------

NO NUMBER

MARKS MARK

SQL> select \* from varr;

NO MARKS

------------ --------------------------------

101 MARK(33, 44, 55, 66)

**Example:**

declare cursor borrower\_cursor is select \* from varr;

begin for borrower\_rec in borrower\_cursor loop for i in 1..borrower\_rec.marks.count loop dbms\_output.put\_line(borrower\_rec.marks(i)); end loop; end loop;

end;

/

33

44

55

66

PL/SQL procedure successfully completed.

## Named Program Units

The PL/SQL programming language allows you to create a variety of named program units they include:

Procedure - A program that executes one or more statements

Function - A program that returns a value

Package - A container for procedures, functions, and data structures

Triggers - Programs that execute in response to database changes

## PROCEDURES

Procedures are program units that execute one or more statements and can receive or return zero or more values through their parameter lists.

The **syntax** of a procedure is:

CREATE [OR REPLACE] PROCEDURE procedure\_name

[ (parameter [,parameter]) ]

IS

[declaration\_section]

BEGIN

executable\_section

[EXCEPTION

exception\_section]

END [procedure\_name];

**Example:**

create procedure NEW\_WORKER (Person\_Name varchar2)

AS

BEGIN

insert into WORKER(Name, Age, Lodging) values (Person\_Name, null, null);

END; /

## Parameter modes

When you create a procedure or function, you may define parameters. There are three types of parameters that can be declared:

**Calling a Procedure**

**IN** Used to pass values to the subprogram when invoked. It acts like a constant and cannot be assigned a value

**OUT** Used to return values to the caller of a subprogram. It can be assigned some values.

**IN OUT** Used to pass initial values to the subprogram when invoked and it also returns updated values to the caller. Can be assigned to other variables or to itself

A call to the procedure is made through an executable PL/SQL statement. The procedure has the following syntax:

*Procedure\_Name [(parameters)];*

The procedure can be executed from SQL environment with the execute command as follows:

*EXECUTE Procedure\_Name [ (parameters) ];*

## Dropping procedure

To drop a procedure, use the drop procedure command, as follows:

*drop procedure NEW\_WORKER;*

## FUNCTIONS

Unlike procedures, functions can return a value to the caller (procedures cannot return values). This value is returned through the use of the return keyword within the function.

A function is characterized as follows:

* A function can be with one, more or no parameters.
* A function must have an explicit RETURN statement in the executable section to return a value.
* The data type of the return value must be declared in the function’s header.
* A function cannot be executed as a standalone program.

The **syntax** for a function is:

CREATE [OR REPLACE] FUNCTION function\_name

[ (parameter [,parameter]) ]

RETURN return\_datatype

IS | AS

[declaration\_section]

BEGIN

executable\_section

[EXCEPTION

exception\_section] END [function\_name];

**Example:**

create or replace function sqrtno(no number) return number

is

begin return no\*no; end;

/

Function created.

SQL> select sqrtno(5) from dual;

SQRTNO(5)

--------------

25

## Dropping function

To drop a function, use the drop function command, as follows: *drop function BALANCE\_CHECK;*

**PROGRAM NO: 1**

**AIM:** BOOK(BOOKID,TITLE,PAGES,PRICE)

AUTHOR(AUTHORID,AUTHOR\_NAME,ADDRESS,BOOKID)

STUDENT\_CARD(ROLLNO,BRANCH,AGE,BOOKID,AUTHORID)

Give SQL Expressions for the following

1. Find details of author who wrote the book which has max. no. of pages.
2. Find the details of students who did not take ‘OS’ book.
3. Find the average age of students in a specified branch.
4. Find the details of students who took the books of authors in ‘Chennai’.

**SOURCE CODE:**

SQL> create table book (bookid int primary key,title varchar(10),pages int,price int);

**OUTPUT**

Table created.

SQL> desc book

Name Null? Type

-------------- ------ ----------

BOOKID NOT NULL NUMBER(38)

TITLE VARCHAR2(10)

PAGES NUMBER(38)

PRICE NUMBER(38)

**SOURCE CODE:**

SQL> create table author(authorid int primary key, authorname varchar(10), address varchar(10), bookid int references book(bookid));

**OUTPUT**

Table created.

SQL> desc author

Name Null? Type

------------- -------- --------------------

AUTHORID NOT NULL NUMBER(38)

AUTHORNAME VARCHAR2(10)

ADDRESS VARCHAR2(10)

BOOKID NUMBER(38)

**SOURCE CODE:**

SQL> create table studentcard(rollno int primary key,branch varchar(5),age int,bookid int references book(bookid),authorid references author(authorid));

**OUTPUT**

Table created.

SQL> desc studentcard

Name Null? Type

-------------------- -------- ----------

ROLLNO NOT NULL NUMBER(38)

BRANCH VARCHAR2(5)

AGE NUMBER(38)

BOOKID NUMBER(38)

AUTHORID NUMBER(38)

**SOURCE CODE**:

SQL> insert into BOOK values(&BOOKID,"&TITTLE",&PAGES,&PRICE);

Enter value for bookid: 101

Enter value for tittle: Database

Enter value for pages: 522

Enter value for price: 799

SQL> /

Enter value for bookid: 102

Enter value for tittle: Java

Enter value for pages: 655

Enter value for price: 999

SQL> /

Enter value for bookid: 103

Enter value for tittle: Os

Enter value for pages: 788

Enter value for price: 855

SQL> /

Enter value for bookid: 104

Enter value for tittle: Acn

Enter value for pages: 699

Enter value for price: 756

SQL> /

Enter value for bookid: 105

Enter value for tittle: Python

Enter value for pages: 455

Enter value for price: 399

**OUTPUT**

5 rows created

**SOURCE CODE:**

SQL> select \* from book;

**OUTPUT**

BOOKID TITLE PAGES PRICE

-------- ---------- ---------- ----------

1 OS 500 750

2 subtle 100 150

3 aadu 500 550

4 godfather 1000 2400

5 AI 459 499

**SOURCE CODE:**

SQL> select \* from author;

**OUTPUT**

AUTHORID AUTHORNAME ADDRESS BOOKID

-------- ---------- ---------- ----------

11 Mark Chicago 2

12 Jhon Chennai 4

13 Steve Chennai 1

14 David Montreal 3

15 Lewis UK 5

**SOURCE CODE:**

SQL> select \* from studentcard;

**OUTPUT**

ROLLNO BRANC AGE BOOKID AUTHORID

--------- ----- ------- ------ ----------

101 MCA 22 1 13

102 MCA 21 4 12

103 MBA 22 2 11

104 MBA 22 3 14

105 MCA 23 5 15

**SOURCE CODE:**

SQL> select author.authorid,authorname, address, book.bookid, pages from author, book where author.bookid=book.bookid and pages=(select max(pages)from book);

**OUTPUT**

AUTHORID AUTHORNAME ADDRESS BOOKID PAGES

-------- ---------- ---------- ---------- ----------

12 Jhon Chennai 4 1000

**SOURCE CODE:**

SQL> select rollno, age, branch from studentcard, book where studentcard.bookid=book.bookid and title!=’OS’;

**OUTPUT**

ROLLNO AGE BRANC

---------- ------ -----

102 21 MCA

103 22 MBA

104 22 MBA

105 23 MCA

**SOURCE CODE:**

SQL> select branch,avg(age)from studentcard group by branch;

**OUTPUT**

BRANC AVG(AGE)

----- ----------

MBA 22

MCA 22

**SOURCE CODE:**

SQL> select rollno,age,branch from studentcard where authorid in (select authorid from author where address=’Chennai’);

**OUTPUT**

ROLLNO AGE BRANC

---------- -------- -----

101 22 MCA

102 21 MCA

**PROGRAM NO: 2**

## AIM: BASIC JOIN & AGGREGATE FUNCTIONS

A Library maintains a database about Books, Authors, Users and Circulation by the following tables

BOOKS(BOOKID,TITLE,COST)

AUTHORS(BOOKID,AUTHOR)

USERS(USERID,NAME,CATEGORY)

CIRCULATION(USERID,BOOKID,ISSUEDATE,RECEIPTDATE)

**Give SQL statements for following**

1)Find the titles of books where ULLMAN is an author

2)Find the usernames of all users who have not returned books costing above rs 300

3)Find titles of books issued to users of category 3 costing avbove rs 300

4)Find titles of books due to be returned. RECEIPTDATE can be 15 days after ISSUEDATE.

**SOURCE CODE:**

SQL> create table BOOKS(BOOKID INT PRIMARY KEY, TITLE VARCHAR2(20), COST NUMBER(7,2) CHECK (COST>0));

Table created.

SQL> CREATE TABLE AUTHORS(BOOKID REFERENCES BOOKS(BOOKID), AUTHOR VARCHAR2(10));

Table created.

SQL> CREATE TABLE USERS(USERID INT PRIMARY KEY, NAME VARCHAR2(10), CATEGORY INT);

Table created.

SQL> CREATE TABLE CIRCULATION(USERID REFERENCES USERS(USERID),BOOKID REFERENCES BOOKS(BOOKID), ISSUEDATE DATE, RECEIPTDATE DATE);

Table created.

SQL> INSERT INTO "DD"."BOOKS" (BOOKID, TITLE, COST) VALUES ('1001', 'Harry Potter', '500')

1 row created.

SQL> INSERT INTO "DD"."BOOKS" (BOOKID, TITLE, COST) VALUES ('1002', 'Goosebumps', '285')

1 row created.

SQL> INSERT INTO "DD"."BOOKS" (BOOKID, TITLE, COST) VALUES ('1003', 'Fighting Fantasy', '625')

1 row created.

SQL> INSERT INTO "DD"."BOOKS" (BOOKID, TITLE, COST) VALUES ('1004', 'Sherlock Holmes', '267')

1 row created.

SQL> INSERT INTO "DD"."AUTHORS" (BOOKID, AUTHOR) VALUES ('1003', 'Ullman')

1 row created.

SQL> INSERT INTO "DD"."AUTHORS" (BOOKID, AUTHOR) VALUES ('1004', 'Ullman')

1 row created.

SQL> INSERT INTO "DD"."AUTHORS" (BOOKID, AUTHOR) VALUES ('1001', 'JK Rowling')

1 row created.

SQL> INSERT INTO "DD"."AUTHORS" (BOOKID, AUTHOR) VALUES ('1002', 'Steve')

1 row created.

SQL> INSERT INTO "DD"."USERS" (USERID, NAME, CATEGORY) VALUES ('2001', 'Dalvin',

'Thriller')

1 row created.

SQL> INSERT INTO "DD"."USERS" (USERID, NAME, CATEGORY) VALUES ('2002', 'Don', 'Fantasy')

1 row created.

SQL> INSERT INTO "DD"."USERS" (USERID, NAME, CATEGORY) VALUES ('2003', 'Richu', 'Suspense')

1 row created.

SQL> INSERT INTO "DD"."CIRCULATION" (USERID, BOOKID, ISSUEDATE, RECEIPTDATE)

VALUES ('2003', '1005', TO\_DATE('11-Jun-2022', 'DD-MON-RR'), TO\_DATE('09-JUL2022', 'DD-MON-RR'))

1 row created.

SQL> INSERT INTO "DD"."CIRCULATION" (USERID, BOOKID, ISSUEDATE, RECEIPTDATE)

VALUES ('2001', '1003', TO\_DATE('12-JUN-2022', 'DD-MON-RR'), TO\_DATE('08-JUL22', 'DD-MON-RR'))

1 row created.

SQL> INSERT INTO "DD"."CIRCULATION" (USERID, BOOKID, ISSUEDATE, RECEIPTDATE)

VALUES ('2002', '1004', TO\_DATE('13-JUN-22', 'DD-MON-RR'), TO\_DATE('15-JUN-22', 'DD-MON-RR'))

1 row created.

SQL> SELECT TITLE FROM BOOKS WHERE BOOKID IN (SELECT BOOKID FROM

AUTHORS WHERE AUTHOR='Ullman');

TITLE

--------------------

Fighting Fantasy

Sherlock Holmes

SQL> SELECT NAME FROM USERS U, CIRCULATION C,BOOKS B WHERE

U.USERID=C.USERID AND C.BOOKID=B.BOOKID AND COST > 300 AND RECEIPTDATE IS NULL;

NAME

----------

Richu

SQL> select TITLE FROM BOOKS B, USERS U,CIRCULATION C WHERE

B.BOOKID=C.BOOKID AND C.USERID=U.USERID AND CATEGORY=3 AND COST > 300; TITLE

--------------------

Harry Potter

SQL> select TITLE FROM BOOKS B,CIRCULATION C WHERE B.BOOKID=C.BOOKID AND ISSUEDATE + 15 < SYSDATE AND RECEIPTDATE IS NULL;

TITLE

--------------------

Harry Potter

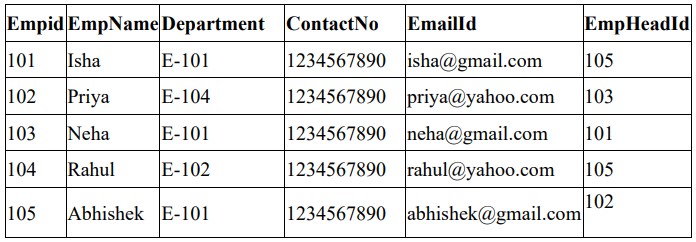
Sherlock Holmes

**PROGRAM NO: 3**

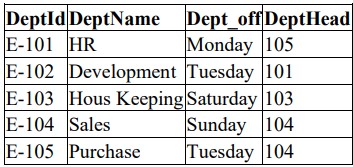
**AIM:** LIKE OPERATOR, INNER QUERIES & GROUP BY

**Section 1:**

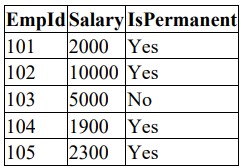
**Table Name: Employee**



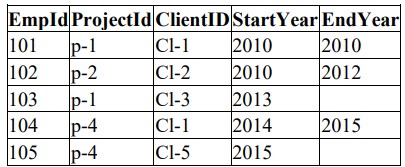
**Table Name: EmpDept**

****

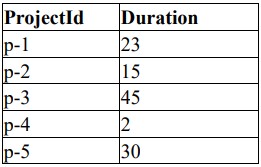
**Table Name: EmpSalary**



**Table Name: Project**



**Table Name: EmpProject**



1. Select the detail of the employee whose name start with P.
2. How many permanent candidate take salary more than 5000.
3. Select the detail of employee whose emailId is in gmail.
4. Select the details of the employee who work either for department E-104 or E102.
5. What is the department name for DeptID E-102?
6. What is total salary that is paid to permanent employees?
7. List name of all employees whose name ends with a.
8. How many project started in year 2010.
9. How many project started and finished in the same year.
10. Select the name of the employee whose name's 3rd character is 'h'
11. Select the department name of the company which is assigned to the employee whose employee id is greater than 103.
12. Select the name of the employee who is working under Abhishek.
13. Select the name of the employee who is department head of HR.
14. Select the employee whose department off is Monday.
15. Select the details of all employees working in development department.

**SOURCE CODE:**

SQL> select \* from employee where empname like 'p%';

**OUTPUT**

EMPID EMPNAME DEPARTMENT CONTACTNO EMAILID EMPHEADID

----- ------- ---------- --------- ------- ---------

102 priya e-104 1234567890 priya@yahoo.com 103

**SOURCE CODE:**

SQL> select count(ispermanent) from empsalary where salary>5000;

**OUTPUT**

COUNT(ISPERMANENT)

------------------

1

**SOURCE CODE:**

SQL> select \* from employee where emailid like '%gmail%';

**OUTPUT**

EMPID EMPNAME DEPARTMENT CONTACTNO EMAILID EMPHEADID

----- ------- ---------- --------- ---------- ----------

101 isha e-101 1234567890 isha@gmail.com 105

103 neha e-101 1234567890 neha@gmail.com 101

105 abhishek e-101 1234567890 abhishek@gmail.com 102

**SOURCE CODE:**

SQL> select \* from employee where department in ('e-102','e104');

**OUTPUT**

EMPID EMPNAME DEPARTMENT CONTACTNO EMAILID EMPHEADID

----- ------- ----------- ----------- -------------- -----------

102 priya e-104 1234567890 priya@yahoo.com 103

104 rahul e-102 1234567890 rahul@yahoo.com 105

**SOURCE CODE:**

SQL> select deptname from empdept where deptid='e-102';

**OUTPUT**

DEPTNAME

---------------

development

**SOURCE CODE:**

SQL> select sum(salary) from empsalary where ispermanent='yes';

**OUTPUT**

SUM(SALARY)

-----------

16200

**SOURCE CODE:**

SQL> select \* from employee where empname like '%a';

**OUTPUT**

EMPIDEMPNAMEDEPARTMENTCONTACTNOEMAILIDEMPHEADID

**-------- --------- ---------- ---------- --------------- ---------**

101 ishae-1011234567890isha@gmail.com105

102 priyae-1041234567890priya@yahoo.com103

103 neha e-101 1234567890 neha@gmail.com 101

**SOURCE CODE:**

SQL> select count(rojected) from empproject where startyear=2010;

**OUTPUT**

COUNT(PROJECTID)

----------------

2

**SOURCE CODE:**

SQL> select count(projectid) from empproject where startyear=endyear;

**OUTPUT**

COUNT(PROJECTID)

----------------

1

**SOURCE CODE:**

SQL> select \* from employee where empname like '\_\_h%';

**OUTPUT**

EMPID EMPNAME DEPARTMENT CONTACTNO EMAILID EMPHEADID

------ -------- ---------- ---------- ------------------ ---------

101 isha e-101 1234567890 isha@gmail.com 105

103 neha e-101 1234567890 neha@gmail.com 101

104 rahul e-102 1234567890 rahul@yahoo.com 105

105 abhishek e-101 1234567890 abhishek@gmail.com 102

**SOURCE CODE:**

SQL> select deptname from empdept where deptid in (select department from employee where empid>103);

**OUTPUT**

DEPTNAME

---------------

development

hr

**SOURCE CODE:**

SQL> select empname from employee where empheadid=(select empid from employee where empname='abhishek');

**OUTPUT**

EMPNAME

---------------

isha

Rahul

**SOURCE CODE:**

SQL> select empname from employee where empid=(select depthead from empdept where deptname='hr');

**OUTPUT**

EMPNAME

---------------

abhishek

**SOURCE CODE:**

SQL> select \* from employee where department in (select deptid from empdept where dept\_off='monday');

**OUTPUT**

EMPID EMPNAME DEPARTMENT CONTACTNO EMAILID EMPHEADID

------ -------- ---------- ---------- ------------------ ---------

101 isha e-101 1234567890 isha@gmail.com 105

103 neha e-101 1234567890 neha@gmail.com 101

105 abhishek e-101 1234567890 abhishek@gmail.com 102

**SOURCE CODE:**

SQL> select \* from employee where department in (select deptid from empdept where deptname='development');

**OUTPUT**

EMPID EMPNAME DEPARTMENT CONTACTNO EMAILID EMPHEADID

-------- --------- ---------- ---------- --------------- ---------

104 rahul e-102 1234567890 rahul@yahoo.com 105

**Section: 2 – INNER QUERIES**

I. Table Name: Employee

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | JOHN | 20 | US | 2000.0 |
| 2 | STEPHAN | 26 | DUBAI | 1500.0 |
| 3 | DAVID | 27 | BANGKOK | 2000.0 |
| 4 | ALINA | 29 | UK | 6500.0 |
| 5 | KATHRIN | 34 | BANGALORE | 8500.0 |
| 6 | HARRY | 42 | CHINA | 4500.0 |
| 7 | JACKSON | 25 | MIZORAM | 10000.0 |

1. Write a SQL statement to display the details of employees who are getting more than 4500 as SALARY.
2. Write a SQL statement to copy the complete Employee table in to newly created Employee\_BKP table.
3. Write a SQL statement to update the Employee\_BKP table in which Salary by 0.25 times in the Employee table for all employee whose Age is greater than or equal to 29.
4. Write a SQL statement to delete the records from Employee table for all Employee whose age is greater than or equal to 29.

**SOURCE CODE:**

SQL> SELECT\*FROM employee2 WHERE id IN (SELECT id FROM employee2 WHERE salary > 4500);

**OUTPUT**

ID NAME AGE ADDRESS SALARY

---------- ---------- --------- ------------ -----------

4 alina 29 uk 6500

5 kathrin 34 banglore 8500

7 jackson 25 mizoram 10000

**SOURCE CODE:**

SQL> create table employee\_bkp(id int,name varchar(15),age int,address varchar(12),salary int);

**OUTPUT**

Table created.

SQL> INSERT INTO employee\_bkp SELECT \* FROM employee2;

7 rows created

**SOURCE CODE:**

SQL> select \* from employee\_bkp;

**OUTPUT**

ID NAME AGE ADDRESS SALARY

-- ---------- ------- --------------- ------------

1 john 20 us 2000

2 stephan 26 dubai 1500

3 david 27 bangkok 2000

4 alina 29 uk 6500

5 kathrin 34 bangalore 8500

6 harry 42 china 4500

7 jackson 25 mizoram 10000

7 rows selected.

**SOURCE CODE:**

SQL> UPDATE employee\_bkp SET SALARY = SALARY \* 0.25 WHERE age

IN (SELECT age FROM employee2 WHERE age >= 29);

**OUTPUT**

3 rows updated.

SQL> select \* from employee\_bkp;

ID NAME AGE ADDRESS SALARY

-- ---------- ------- --------------- ------------

1 john 20 us 2000

2 stephan 26 dubai 1500

3 david 27 bangkok 2000

4 alina 29 uk 6500

5 kathrin 34 bangalore 8500

6 harry 42 china 4500

7 jackson 25 mizoram 10000

7 rows selected.

**SOURCE CODE:**

SQL> DELETE FROM employee2 WHERE age IN (SELECT age FROM employee\_bkp WHERE age =< 29 );

**OUTPUT**

3 rows deleted.

SQL> select \* from employee2;

ID NAME AGE ADDRESS SALARY

-- ---------- ------- --------------- ------------

1 john 20 us 2000

2 stephan 26 dubai 1500

3 david 27 bangkok 2000

4 alina 29 uk 6500

7 jackson 25 mizoram 10000

### Section: 3 – GROUP BY

Table Name: Officers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Officer\_id** | **Officer\_nam**  **e** | **address** | **Working\_ho**  **urs** | **Date** |
| 1 | Ajeet | Mau | 12 | 2021-01-24 |
| 2 | Deepika | Lucknow | 10 | 2021-01-24 |
| 3 | Vimal | Faizabad | 5 | 2021-01-25 |
| 4 | Rahul | Lucknow | 4 | 2021-01-25 |
| 5 | Ajeet | Mau | 9 | 2021-01-26 |
| 6 | Deepika | Lucknow | 5 | 2021-01-26 |
| 7 | Vimal | Faizabad | 12 | 2021-01-27 |
| 8 | Rahul | Lucknow | 10 | 2021-01-27 |

1. Write a SQL statement to display how many officers are from the same city.
2. Write a SQL statement to display officer\_name and total working hours of each officers from officers table.
3. Write a SQL statement to display minimum working hours of each officers from officer table.
4. Write a SQL statement to display maximum working hours of each officers from officer table.
5. Write a SQL statement to display average working hours of each officers from officer table.
6. Write a SQL statement to display Total working hours of each officers having more than 5 hours from officer table.

**SOURCE CODE:**

SQL> SELECT address, COUNT(\*) FROM officers GROUP BY address;

ADDRESS COUNT(\*)

------------- ----------

lucknow 4

mau 2

faizabad 2

**SOURCE CODE:**

SQL> SELECT officername, SUM(workinghours) AS "Total working hours" FROM officers GROUP by officername;

**OUTPUT**

OFFICERNAME Total working hours

--------------- -------------------

Rahul 14

Deepika 15

Vimal 17

Ajeet 21

**SOURCE CODE:**

SQL> SELECT officername, MIN(workinghours) AS "Minimum working hour" FROM officers GROUP BY officername;

**OUTPUT**

OFFICERNAME minimum working hours

--------------- ---------------------

Rahul 4

Deepika 5

Vimal 5

Ajeet 9

**SOURCE CODE:**

SQL> SELECT officername, MAX(workinghours) AS "Maximum working hour" FROM officers GROUP BY officername;

OUTPUT

OFFICERNAME Maximum working hours

--------------- ----------------------

Rahul 10

Deepika 10

Vimal 12

Ajeet 12

**SOURCE CODE:**

SQL> SELECT officername, AVG(workinghours) AS "Avg working hour" FROM officers GROUP BY officername;

**OUTPUT**

OFFICERNAME Avg working hours

--------------- --------------------

Rahul 7

Deepika 7.5

Vimal 8.5

Ajeet 10.5

**SOURCE CODE:**

SQL> SELECT officername, SUM(workinghours) AS "Total working

hours" FROM officers GROUP BY officername HAVING SUM(workinghours) > 5;

**OUTPUT**

OFFICERNAME Total working hours

--------------- -------------------

Rahul 14

Deepika 15

Vimal 17

Ajeet 21

**PROGRAM NO:4**

**AIM:** LABORATORY SYSTEM 1

**CREATE THE TABLE DESCRIBED BELOW**

1. TableName: Client\_master(Client\_no,Name,Address1,Adress2,City,Pincode,State, Bal\_due)

2. TableName

:Product\_Master(Product\_no, Description, Profit\_percent, Unit\_measure, Qty \_on\_hand, Reorder\_lvl, Sell\_price, Cost\_price)

3.TableName:

Salesman\_master(Salesman\_no, Salesman\_name, Address1, City, Pincode, State, Sal\_amt, Tgt\_to\_get, Ytd\_sales, Remarks)

4. TableName:

Sales\_order(Order\_no, Order\_date, Client\_no, Delay\_addr, Salesman\_no, Dely\_type, Billed\_yn,Dely\_date,Order\_status)

5. TableName: Sales\_order\_detail(Order\_no,Product\_no,Qty\_ordered,Qty\_disp,Product \_rate)

**EXERCISES ON COMPUTATIONS ON TABLE DATA**

1. Find the names of all clients having 'b' as the second letter in their names?
2. Find the clients who stay in a city whose second letter in 'o'?
3. Find the list of all clients who stay in 'Bombay' or 'Kottayam'?
4. Print the list of clients whose bal\_due is greater than value 15000?
5. Print the information from Sales\_order table for enters places in the month of 'March'?
6. Display the order information for client\_no 'C001' and 'C002'?
7. Find the products whose selling price is greater than 2000 and less than or equal to 5000?
8. Find products whose selling price is more than 50000. Calculate a new selling\_price=original selling price \*.15.Rename the new column in the above query as new\_price?
9. List the names ,city and state of clients who are not in the state of 'Kerala'?
10. Count the total number of orders?
11. Calculate the average price of all the products?
12. Determine the maximum and mnimum product prices.Rename the output as max\_p;rice and min\_price respectively?
13. Count the number of products having price greater than or equal to 25000?
14. Find all the product whose Qty\_on\_hand is less than recorder level?

**SOURCE CODE:**

CREATE TABLE client\_master (client\_no varchar(10) primary key, name varchar(20), address1 varchar(30), address2 varchar(30), city varchar(15), pincode number(8), state varchar(15), bal\_due number(7,2));

CREATE TABLE product\_master (product\_no varchar(10) primary key, description varchar(15), profit\_percent number(4,2), unit\_measure varchar(10), qty\_on\_hand number(8), reorder\_lv1 number(8), sell\_price number(8,2), cost\_price number(8,2));

CREATE TABLE salesman\_master (salesman\_no varchar(10) primary key, address1 varchar(30), city varchar(20), pincode number(8), state varchar(20), sal\_amt number(8,2), tgt\_to\_get number(6,2), ytd\_sales number(6,2), remarks varchar(60));

CREATE TABLE sales\_order (order\_no varchar(10) primary key, order\_date date, client\_no varchar(10), references client\_master(client\_no), delay\_addr varchar(25), salesman\_no varchar(10), references salesman\_master(salesman\_no), delytype char(1), billed\_yn char(1), dely\_date date, order\_status varchar(10));

CREATE TABLE sales\_order\_details (order\_no varchar(10), references sales\_order(order\_no), product\_no varchar(10), references product\_master(product\_no), qty\_ordered number(8), oty\_disp number(8), product\_rate number(10,2));

desc client\_master;

desc product\_master;

desc salesman\_master;

desc sales\_order;

desc sales\_order\_details;

INSERT INTO client\_master (client\_no, name, address1, address2, city, pincode, state, bal\_due) VALUES ('C001', 'Abin Thomas', 'abc', 'abc', 'Kottayam', 686537 ,'Kerala', 17895.58), ('C003', 'Antony Adam', 'xyz', 'zyz', 'Kottayam', 686537

,'Kerala', 1000),('C004', 'Visakh Vijayan', 'wer', 'wer', 'Kollam', 578496 ,'Kerala', 100.50), ('C006', 'Sooraj Krishna', 'qre', 'qre', 'Bombay', 102345,'Maharashtra',24321.12), ('C002', 'Don J', 'iop', 'iop', 'Chennai',789563 ,'Tamilnadu', 15889), ('C005', 'Abhirami A J', 'uip', 'uip', 'Bangalore',600010 ,'Karnataka', 19555.95);

SELECT \* FROM client\_master;

INSERT INTO product\_master (product\_no, description, profit\_percent, unit\_measure, qty\_on\_hand, reorder\_lv1, sell\_price, cost\_price) VALUES

('P001', 'Keyboard', 5, 'per', '5', 2, 1200, 999),

('P002', 'Mouse', 5, 'per', '5', 2, 999, 800),

('P003', 'Graphics Card', 15, 'per', '10', 12, 32000, 26000),

('P004', 'Monitor', 10, 'per', '7', 16, 20000, 15000),

('P005', 'X-box', 20, 'per', '9', 20, 50000, 39999),

('P006', 'Laptop', 15, 'per', '9', 20, 62000, 52452);

SELECT \* FROM product\_master;

INSERT INTO salesman\_master (salesman\_no, address1, city, pincode, state, sal\_amt, tgt\_to\_get, ytd\_sales, remarks) VALUES

('S001', 'aqk', 'Kottayam', 686537, 'Kerala', 15000, 45, 6, 'Good'),

('S002', 'ppp', 'Kollam', 578496, 'Kerala', 15000, 20, 13, 'Fine'),

('S003', 'attt', 'Banglore', 600010, 'Karnataka', 25000, 33, 3, 'Very good'),

('S004', 'awek', 'Bombay', 102345, 'Maharashtra', 22000, 27, 1, 'Good'),

('S005', 'fd', 'Chennai', 789563, 'Tamilnadu', 35000, 15, 10, 'Nice'),

('S006', 'afgjk', 'Banglore', 600011, 'Karnataka', 25000, 22, 5, 'Good');

SELECT \* FROM salesman\_master;

INSERT INTO sales\_order (order\_no, order\_date, client\_no, delay\_addr, salesman\_no, delytype, billed\_yn, dely\_date, order\_status) VALUES

('OR101', '02-MAR-2014', 'C001', 'abc', 'S001', 'F', 'Y', '08-MAR-2014', 'Fullfilled'), ('OR102', '18-MAR-2014', 'C004', 'wer', 'S003', 'F', 'Y', '22-MAR-2014', 'Fullfilled'), ('OR103', '14-FEB-2014', 'C002', 'iop', 'S004', 'F', 'Y', '1-MAR-2014', 'Fullfilled'), ('OR104', '15-FEB-2014', 'C001', 'abc', 'S005', 'P', 'N', '10-MAR-2014', 'In Process'), ('OR105', '22-JAN-2022', 'C005', 'uip', 'S006', 'F', 'Y', '04-FEB-2022', 'Fullfilled'), ('OR106', '1-JUN-2022', 'C003', 'xyz', 'S001', 'P', 'N', '08-JAN-2022', 'In Process');

SELECT \* FROM sales\_order;

INSERT INTO sales\_order\_details (order\_no, product\_no, qty\_ordered, oty\_disp, product\_rate) VALUES ('OR101', 'P001', 30, 27, 8),('OR106', 'P004', 45, 45, 9),

('OR103', 'P003', 10, 9, 7),('OR102', 'P005', 12, 11, 6),('OR103', 'P001', 10, 10, 5), ('OR106', 'P003', 11, 5, 8),('OR105', 'P006', 63, 50, 9);

SELECT \* FROM sales\_order\_details;

-- OUTPUT --

SQL> SELECT name FROM client\_master WHERE name LIKE '\_b%';

NAME

---------------

Abin Thomas

Abhirami A J

SQL> SELECT client\_no, name, city FROM client\_master WHERE city LIKE '\_o%';

CLIENT NAME CITY

------ --------------- ---------------

C001 Abin Thomas Kottayam

C003 Antony Adam Kottayam

C004 Visakh Vijayan Kollam

C006 Sooraj Krishna Bombay

SQL> SELECT client\_no, name, city FROM client\_master WHERE city='Bombay' OR city='Kottayam';

CLIENT NAME CITY

------ --------------- ---------------

C001 Abin Thomas Kottayam

C003 Antony Adam Kottayam

C006 Sooraj Krishna Bombay

SQL> SELECT client\_no, name, bal\_due FROM client\_master WHERE bal\_due>15000;

CLIENT NAME BAL\_DUE

------ --------------- ----------

C001 Abin Thomas 17895.58

C006 Sooraj Krishna 24321.12

C002 Don J 15889

C005 Abhirami A J 19555.95

SQL> SELECT order\_date, salesman\_no, order\_status FROM sales\_order WHERE order\_date LIKE '%MAR%';

ORDER\_DAT SALESMAN\_N ORDER\_STAT

--------- ---------- ----------

02-MAR-14 S001 Fullfilled

18-MAR-14 S003 Fullfilled

SQL> SELECT \* FROM sales\_order WHERE client\_no IN ('COO1','C002');

ORDER\_NO ORDER\_DAT CLIENT\_NO DELAY\_ADDR SALESMAN\_N D B DELY\_DATE ORDER\_STAT

--------- --------- ---------- ---------- ---------- - - --------- ----------

OR103 14-FEB-14 C002 iop S004 F Y 01-MAR-14 Fullfilled

SQL> SELECT product\_no, description, sell\_price FROM product\_master WHERE sell\_price>2000 AND sell\_price<=25000;

PRODUCT\_NO DESCRIPTION SELL\_PRICE

---------- --------------- ----------

P004 Monitor 20000

SQL> SELECT product\_no, description, sell\_price sell\_price+sell\_price\*15 AS new\_pr FROM product\_master WHERE sell\_price>50000;

PRODUCT\_NO DESCRIPTION SELL\_PRICE NEW\_PR

---------- --------------- ---------- ----------

P006 Laptop 62000 992000

SQL> SELECT name, city, state FROM client\_master WHERE state NOT LIKE 'Kerala';

NAME CITY STATE

--------------- --------------- ---------------

Sooraj Krishna Bombay Maharashtra

Don J Chennai Tamilnadu

Abhirami A J bangalore Karnataka

SQL> SELECT COUNT(order\_no) FROM sales\_order;

COUNT(ORDER\_NO)

---------------

6

SQL> SELECT AVG(sell\_price), AVG(cost\_price) FROM product\_master;

AVG(SELL\_PRICE) AVG(COST\_PRICE)

--------------- ---------------

27699.8333 22541.6667

SQL> SELECT MAX(sell\_price) AS max\_price, MIN(sell\_price) AS min\_price FROM product\_master;

MAX\_PRICE MIN\_PRICE

---------- ----------

62000 999

SQL> SELECT COUNT(product\_no) FROM product\_master WHERE sell\_price>=25000;

COUNT(PRODUCT\_NO)

-----------------

3

SQL> SELECT \* FROM product\_master WHERE qty\_on\_hand<reorder\_lv1;

PRODUCT\_NO|DESCRIPTION|PROFIT\_PERCENT|UNIT\_MEASU|QTY\_ON\_HAND|REORDER\_LV1|SELL\_PRICE|COST\_PRICE

+---------+------------+-------------+----------+-----------+-----------+----------+---------+

P003 Graphics card 15 per 10 12 32000 26000

P004 Monitor 10 per 7 16 20000 15000

P005 X-box 20 per 9 20 50000 39999

P006 Laptop 15 per 9 20 62000 52452

**PROGRAM NO: 5**

### AIM: PLSQL (TRIGGERS & PROCEDURES)

SECTION: 1 – **TRIGGERS**

1) To write a database trigger before delete for each row, not allowing deletion and giving an appropriate message, for the employee table.

ALGORITHM

Step 1: Start trigger

Step 2: Define trigger with name emply\_trigg and act before deletion on employee table

Step 3: For any row is going to affect delete operation then Raise application error with an error message 'Deletion not allowed on employee'

Step 4: Stop trigger

**SOURCE CODE:**

create or replace trigger emply\_trigg before delete on employee for each row Begin

raise\_application\_error(-20101,'Deletion not allowed on employee');

End;

/

**OUTPUT**

Trigger created.

-- delete table --

SQL> DELETE from employee where empid=20768;

DELETE from employee where empid=20768

\*

ERROR at line 1:

ORA-20101: Deletion not allowed on employee

ORA-06512: at "SYSTEM.EMPLY\_TRIGG", line 2

ORA-04088: error during execution of trigger 'SYSTEM.EMPLY\_TRIGG'

2) Create a trigger that may insert a tuple into EMPLOYEE(empno,ename) when a tuple is updated into MANAGER(managerid,mname) the trigger has to check whether the new tuple has a first component 50 or less, and if so insert the tuple into EMPLOYEE

ALGORITHM

Step 1: Start trigger

Step 2: Define trigger with name tr2 and act after updating on manager table and refer new row as new

Step 3: For any row’s manager id gets updated to less than or equal to 50 then

Insert updated record on employee table

Step 4: Stop trigger

SOURCE CODE:

SQL> create or replace trigger tr2

after update on manager

referencing new as newrow

for each row when (newrow.managerid<=50)

begin

insert into employee values

(:newrow.managerid,:newrow.mname);

end tr2;

/

**OUTPUT**

Trigger created.

SQL> select \* from employee;

EMPID EMPNAME

---------- ------------

20765 Sam

20766 Pooja

20767 Abhirami

20768 Richu

20769 Ajo

20764 Ram

6 rows selected.

SQL> select \* from manager;

MANAGERID MNAME

---------- ------------

10 suhana

11 don

12 deepak

SQL> set serveroutput on

SQL> update manager set managerid = 34 where mname='don';

1 row updated.

SQL> update manager set managerid = 54 where mname='deepak';

1 row updated.

SQL> select \* from manager;

MANAGERID MNAME

---------- ------------

10 suhana

34 don

54 deepak

SQL> select \* from employee;

EMPID EMPNAME

---------- ------------

20765 Sam

20766 Pooja

20767 Abhirami

20768 Richu

20769 Ajo

20764 Ram

34 don

7 rows selected.

SECTION: 2 – **PROCEDURES**

Create a procedure to evaluate student grade by accepting student name.

**ALGORITHM**

Step 1: Start procedure.

Step 2: Define procedure sgrade taking parameter name and program variable mk(mark).

Step 3: Select mark of student into an integer(mk) whose name had been passed.

Step 4: If mk greater than 90 then Print “Grade:=X”.

Else if mk greater than 80 then Print “Grade:=A”.

Else if mk greater than 60 then Print “Grade:=B”.

Else if mk greater than 50 then Print “Grade:=C”.

Else Print “Grade:=F”.

Step 5: Stop procedure.

**SOURCE CODE:**

create or replace procedure sgrade(n\_name IN varchar2) IS mk number(3); Begin

select mark into mk from student where name=n\_name; if mk>90 then

dbms\_output.put\_line('Grade:=X'); else if mk>80 then dbms\_output.put\_line('Grade:=A'); else if mk>60 then dbms\_output.put\_line('Grade:=B'); else if mk>50 then dbms\_output.put\_line('Grade:=C'); else

dbms\_output.put\_line('Grade:=F'); end if; end if; end if; end if;

End sgrade;

/

OUTPUT

Procedure created.

SQL> execute sgrade('rajesh');

Grade:=B

**PROGRAM NO: 6**

## AIM: PROCEDURES & FUNCTIONS

SECTION: 1 – **PROCEDURES**

Write a procedure in PL/SQL to check if employee is eligible for promotion depending on his period of service.

Table Used

ENO ENAME DNAME DOJ DESIG

--------- --------------- --------------- -------------- ---------------

100 Rajesh MCA 25-MAY-22 Lecturer

102 Joy ME 20-APR-22 Lecturer

104 Ranjith EC 12-JAN-22 Lecturer

105 Mable CSE 12-JAN-22 Lecturer

ALGORITHM

Step1: Start.

Step2: Define procedure des\_ch taking parameter employee number (eid)

Step3: Declare variables dt (experience), jd(joining date), sd(system date).

Step4: Select doj field value from relation emp\_proc into program variable jd for an employee (whose employee id is eid).

Step5: Select system date into program variable sd.

Step6: Compute the number of months between jd and sd dates. Step7: Check if dt greater than or equal to 3, if true then update design=ass.prof

Print “Your designation changed”.

Else

Print “you have to wait”.

Step8: Stop.

**SOURCE CODE:**

create or replace procedure des\_ch(eid IN varchar2)

IS dt number; jd date; sd date; Begin

select doj into jd from emp\_proc where eno=eid; select sysdate into sd from dual; dt:=months\_between(sd,jd) / 12; if dt>=3 then

update emp\_proc set desig='ass.prof' where eno=eid; dbms\_output.put\_line('Your designation changed'); else

dbms\_output.put\_line('you have to wait'); end if;

End des\_ch;

**OUTPUT**

SQL> execute des\_ch(100); you have to wait

PL/SQL procedure successfully completed.

SECTION: 2 – **FUNCTION**

Write a function in PL/SQL to find student grade by accepting student name as argument.

Table used:

NAME MARK

---------- ----------

jerin 85

shyam 73

linto 69

rakhesh 84

ALGORITHM

Step 1: Start

Step 2: Define function Sff taking parameter as name.

Step 3: Declare variables mk(mark) and g(grade).

Step 4: Select mark of student into an mk whose name had been passed. Step 5: If mk greater than 90 then g=x

Print “Grade:=X”.

Else if mk greater than 80 then g=A

Print “Grade:=A”.

Else if mk greater than 60 then g=B

Print “Grade:=B”.

Else if mk greater than 50 then g=C

Print “Grade:=C”.

Else g=C.

Print “Grade:=F”.

Step 6: Return value to grade g.

Step 7: Stop

SOURCE CODE:

create or replace function fgrade(n\_name IN varchar2) return varchar2 IS g varchar2(1); mk number(3);

Begin

select mark into mk from student where name=n\_name; if mk>90 then g:='x';

dbms\_output.put\_line('Grade: ' || g ); else if mk>80 then g:='A';

dbms\_output.put\_line('Grade: ' || g ); else if mk>60 then g:='B';

dbms\_output.put\_line('Grade: ' || g ); else if mk>50 then g:='C';

dbms\_output.put\_line('Grade: ' || g ); else g:='F';

dbms\_output.put\_line('Grade: ' || g ); end if; end if; end if; end if; return g;

End fgrade;

OUTPUT

SQL> select fgrade('rakhesh') from dual; fgrade('NTINI')

--------------------------------------

A

Grade: A

**PROGRAM NO: 7**

## AIM: CURSOR & CAR DATABASE APPLICATION

SECTION: 1 – **CURSOR**

Write a PL/SQL block to display total salary of all employees using cursor.

Table Used

EMPID EMPNAME SALARY COMM

---------- ------------ ---------- ----------

20765 Sam 5000 1500

20766 Pooja 4500 750

20767 Abhirami 4000 1050

20768 Richu 5600 1100

ALGORITHM:

Step1: Start

Step2: declare the variables n1(empno),sl(sal),na(empname),com(comm) of %type Step3: Initialize the cursor c1 by selecting empno,empname,sal,comm from emp.

Step4: open the cursor c1

Step5: fetch the values of n1,na,sl,com into cursor c1.

Step6: calculate ts = sl + com

Step7: print the values of n1,na,ts

Step8: close the cursor c1

Step9: Stop

**SOURCE CODE:**

SQL> declare

1. n1 employee.empid%type;
2. sl employee.salary%type;
3. na employee.empname%type;
4. com employee.comm%type;
5. cursor c1 is select empid, empname, salary, comm from employee;
6. ts int;
7. begin
8. open c1;
9. dbms\_output.put\_line('…………………');
10. loop
11. fetch c1 into n1,na,sl,com;
12. exit when (c1%notfound);
13. ts:=sl+nvl(com,0);
14. dbms\_output.put\_line('empno' || to\_char(n1));
15. dbms\_output.put\_line('name' || na);
16. dbms\_output.put\_line('total salary' || to\_char(ts));
17. dbms\_output.put\_line('…………………');
18. end loop;
19. close c1;
20. end;
21. /

**OUTPUT**

.......

empno20765 nameSam

total salary6500

.......

empno20766 namePooja total salary5250

.......

empno20767 nameAbhirami total salary5050

.......

empno20768 nameRichu

total salary6700

.......

PL/SQL procedure successfully completed.

## SECTION: 2 – CAR DATABASE APPLICATION

Create the following database for an application and insert values to it.

car(serialno,model,maufacturer,price) options(serialno,optionname,price) sales(salespersonid,serialno,date,salesprice) salesperson(salespersonid,name,phone) populate the database with data 2. Do the following using PL/SQL

1. Create a trigger that prints the change in price everytime the price of the car is changed.
2. For the salesperson named joe, list the following information for all the cars he has sold:

serial number, manufacturer, salesprice

1. List the serial number, model no. of cars that have no options.

ALGORITHM:

Step 1: Start

Step 2: Create a table Car, Options, Sales, Salesperson.

Step 3: Populate the tables with data.

Step 4: Create a trigger named sellprice before update operation on car table for each row.

1. Declare a variable f of number type.
2. Check the if condition for Non-equality of old and new values of the field price.
3. Store the difference of the field price in f.

d.Print the value of f.

Step 5: Initliaze a cursor named scursor for storing the value of the field serialno, manufacturer, salesprice from car table, sales table, salesperson table based on a condition.

1. The condition should satisfy that the salespersonid from the sales table and salesperson should be alike,salesperson name should be ‘joe’ and serialno from car and sales table should also be same.
2. print the value of serialno, manufacturer and salesprice.

Step 6: Create a cursor named scursor for storing the fields serialno, model from the table car where serialno is not in options.

a. print the values of serialno and model form the sval obtained from scursor.

Step 7: Stop

SOURCE CODE:

SQL> create or replace trigger sellprice

1. before update on car
2. FOR EACH ROW
3. declare
4. f number(10);
5. begin
6. if :old.price <> :new.price then
7. f:=:new.price - :old.price;
8. dbms\_output.put\_line('Change in price = '||f);
9. end if;
10. end;
11. /

Trigger created.

SQL> set serveroutput on

SQL> update car set price=350050 where serialno=100001;

Change in price = 25

1. row updated.

SQL> declare

1. cursor scursor is select s.serialno,manufacturer,salesprice from car c,sales s, salesperson sp where
2. s.salespersonid=sp.salespersonid and sp.name='Joe'and s.serialno=c.serialno; 4 begin
3. for sval in scursor
4. loop
5. dbms\_output.put\_line(sval.serialno||' '||sval.manufacturer||' '||sval.salesprice);
6. end loop;
7. end;
8. /

100002 Maruthi 451493

PL/SQL procedure successfully completed.

SQL> declare

1. cursor scursor
2. is select serialno,model from car
3. where serialno not in (select serialno from options);
4. begin
5. for sval in scursor
6. loop
7. dbms\_output.put\_line(sval.serialno||' '||sval.model);
8. end loop;
9. end; 11 /

100003 Swift

100001 Alto

PL/SQL procedure successfully completed.

**PROGRAM NO: 8**

## AIM: STUDENT DATABASE APPLICATION

1. Create the following database for an application and insert values to it. student(ssn,name,major,bdate) course(coursenumber,coursename,department) enroll(ssn,coursenumber,quarter,grade) book(coursenumber,quarter,book-isbn) text(book-isbn,book-title,publisher,author)
2. Do the following using PL/SQL
   1. List the number of courses taken by all students named joe in quarter-1
   2. Produce a list of textbooks(coursenumber,book-isbn,book-title etc)for courses offered by cs department that have used more than two books.
3. List any department that has all its books published by "pearson".
4. Create a trigger that prints the name of the book when the isbn number of the book is changed.

**ALGORITHM:**

Step 1: Start

Step 2: Create tables Student, course, enroll,book,text

Step 3: Declare variable numberOfCourses

1. Select the distinct courseno from student table and enroll table.
2. Selection is based on the condition that the value of ssn of the student table and enrol table should be same, quarter = 1 and name of the student should be ‘JOE’

Step 4: Intialize a cursor <cursor name> for fields coursenumber, book\_isbn, book\_title from tables, book and text.

1. The above selction is based on the condition that the values of field coursenumber from book and course table should be same and book\_isbn value from book and text table should be same and department name should be ‘computer science’.
2. Group by coursenumber where the field book\_isbn should be distinct and the count of book\_isbn should be greater than two.

Step 5: Create a cursor named cursor1 to store the field department and count of distinct book\_isbn field from course and book table based on the condition that the value of the coursenumber should be same in the tables.

a. Create another cursor named cursor2 to store the field department, count of distinct book\_isbn with the condition that the value of coursenumber from the course and book table should be same ,similarly the \_isbn number from book and text table should be same and also the publisher should be ‘Pearson’. b. open cursor2

1. Create a for loop for row1 in cursor1
2. Fetch the values in cursor2 into row2
3. if row1.department is equal to row2.department then check whether row1.num equals row2.num then print the value of row1.department.

Step 6: Create a trigger named print\_book\_title before updating the text for each row based on the if condition, that the old isbn value of the book (before updation) and the new value(after updation) does not match..

a. Print the value of the object old of the field book\_isbn.

Step 7: Stop

**SOURCE CODE:**

SQL> create table student(ssn varchar(6) primary key,name varchar(20),major varchar(30),bdate date);

Table created.

SQL> create table course(coursenumber varchar(6) primary key,coursename varchar(25),department varchar(25));

Table created.

SQL> create table enroll(ssn varchar(6) references student(ssn),coursenumber varchar(6) references course(coursenumber),quarter number(1),grade number(1));

Table created.

SQL> create table text(book\_isbn varchar(20) primary key,book\_title varchar(30),publisher varchar2(30),author varchar(30));

Table created.

SQL> create table book(coursenumber varchar2(20) references course(coursenumber),quarter number(2),book\_isbn varchar(20) references text(book\_isbn));

Table created.

SQL> insert into student values('s1','Akshay','major','10-FEB-1990');

1 row created.

SQL> insert into student values('s2','Joe','major','01-APR-1991');

1 row created.

SQL> insert into student values('s3','Jibin','major','20-DEC-1989');

1 row created.

SQL> insert into student values('s5','Anuja','major','19-SEP-1990');

1 row created.

SQL> insert into course values('c101','btech','cs');

1 row created.

SQL> insert into course values('c102','bhm','HK');

1 row created.

SQL> insert into course values('c103','PG','MCA');

1 row created.

SQL> insert into enroll values('s1','c102',2,3);

1 row created.

SQL> insert into enroll values('s2','c101',1,2);

1 row created.

SQL> insert into enroll values('s5','c103',4,1);

1 row created.

SQL> insert into text values('b1','Database','Hudson plb','Jain');

1 row created.

SQL> insert into text values('b2','Java 2','Pearson','Herbict');

1 row created.

SQL> insert into text values('b4','OOPS','Hardy','Dwayne');

1 row created.

SQL> insert into text values('b7','OOP with c++','Pearson','Balaguru');

1 row created.

SQL> insert into text values('b9','Operating system','Pearson','Allan morge');

1 row created.

SQL> insert into book values('c101',1,'b2');

1 row created.

SQL> insert into book values('c101',1,'b7');

1 row created.

SQL> insert into book values('c101',2,'b9');

1 row created.

SQL> insert into book values('c103',2,'b1');

1 row created.

SQL> insert into book values('c103',4,'b2');

1. row created.

SQL> set serveroutput on

SQL> declare

1. numberOfCourses number(5);
2. begin
3. select count(distinct coursenumber) into numberOfCOurses from student, enroll
4. where student.ssn = enroll.ssn and quarter = 1 and name like 'JOE%'; 6 dbms\_output.put\_line('Number of courses taken by all students named Joe in quarter 1 is' || numberOfCourses);
5. end;
6. /

Number of courses taken by all students named Joe in quarter 1 is0

PL/SQL procedure successfully completed.

SQL> declare

2 cursor cursor1 is select DISTINCT coursenumber, a.book\_isbn, book\_title 3 from book a, text b

1. where a.book\_isbn = b.book\_isbn and
2. coursenumber in (
3. select a.coursenumber
4. from course a, text b, book c where
5. a.coursenumber = c.coursenumber and
6. b.book\_isbn = c.book\_isbn and
7. a.department like 'Computer Science'
8. group by a.coursenumber having count(distinct b.book\_isbn)>2);
9. row1 cursor1%ROWTYPE;
10. begin
11. dbms\_output.put\_line('COURSENUMBER BOOK\_ISBN BOOK\_TITLE');
12. for row1 in cursor1
13. loop
14. dbms\_output.put\_line(row1.coursenumber || ' ' || row1.book\_isbn || '' || row1.book\_title); 18 end loop;
15. end;
16. /

COURSENUMBER BOOK\_ISBN BOOK\_TITLE c101 b9 Operating system c101 b2 Java 2 c101 b7 OOP with C++

PL/SQL procedure successfully completed.

SQL> declare

1. cursor cursor1 is select a.department, count(distinct b.book\_isbn) as num from course a,
2. book b
3. where a.coursenumber = b.coursenumber group by department;
4. cursor cursor2 is select department, count(distinct b.book\_isbn) as num from course a, book
5. b, text c
6. where a.coursenumber = b.coursenumber and
7. c.book\_isbn = b.book\_isbn and
8. publisher like 'Pearson' group by department;
9. row1 cursor1%ROWTYPE;
10. row2 cursor2%ROWTYPE;
11. begin
12. open cursor2;
13. for row1 in cursor1
14. loop
15. fetch cursor2 into row2;
16. if(row1.department = row2.department) then
17. if(row1.num = row2.num) then
18. dbms\_output.put\_line(row1.department);
19. end if;
20. end if;
21. end loop;
22. end;
23. /

cs

PL/SQL procedure successfully completed.

SQL> create or replace trigger print\_book\_title

1. before update on text
2. for each row
3. begin
4. if(:old.book\_isbn <> :new.book\_isbn) then
5. dbms\_output.put\_line('Book Title');
6. dbms\_output.put\_line('...................');
7. dbms\_output.put\_line(:old.book\_title);
8. end if;
9. end;
10. /

Trigger created.

SQL> update text set book\_isbn = 'b8' where book\_isbn='b4'; Book Title

...................

OOPS 1 row updated.

**PROGRAM NO: 9**

## AIM: HOSPITAL DATABASE APPLICATION

1. Create the tables below and insert data

1. Patient(patient-no,name,dob,age,place) with primary key patient-no
2. Doctor(doct-id,specialization,salary) with primary key doct-id
3. Treatment (patient-no, doct-id,from-date,to-date) patient-no foreign key refrences patient.

i)Develop SQL Queries for the following

1.Find the details of patients treated by the doctor with maximum salary

2.Find the details of doctors who treated patients within the period 04/06/2022 to 03/10/2022

3.Find the details of patients discharged on august ii). Create triggers for the following

1. Deletion is possible from patient table if dob is less than 01/01/1910
2. Updation is possible on doctor table if the specialization is pediatric

1.select doctor\_id, specialization from doctor where doctor\_id in

(select doctor\_id from treatment where from\_date >='04-jun-20022 and date\_to <= '03-oct-2022');

2.select doctor\_id,(count(patient\_no)) from treatment group by doctor\_id order by count(patient\_no) desc;

3.select \* from patient where patient\_no in (select patient\_no from treatment where substr(date\_to, 4,3) like 'aug');

ii. a. ALGORITHM

Step 1: Start trigger

Step 2: Define trigger with name delpatient and act before deletion on patient table

Step 3: For any row check if old d\_o\_b value is greater ‘1-Jan-1910’ then

Raise application error with an error message 'not possible' Step 4: Stop trigger ii.b. ALGORITHM

Step 1: Start trigger

Step 2: Define trigger with name updoctor and act before updating on doctor table

Step 3: For any row check if old specialization value is not equal ‘pediatric’ then

Raise application error with an error message 'updation not possible'

Step 4: Stop trigger

**SOURCE CODE:**

11.HOSPITAL DATABASE APPLICATION

1) a.create table Patient(patient\_no int PRIMARY KEY,name varchar(11),dob date,age int,place varchar(14));

b.create table Doctor(doctor\_id int PRIMARY KEY,specialization varchar(20),salary int);

c.create table Treatment(patient\_no int references Patient,doctor\_id int references Doctor,fromdate date,todate date);

1. 1.select \* from patient where patient\_no in(select patient\_no from treatment where doctor\_id in(select doctor\_id from doctor where salary in(select max(salary) from doctor)));

2.select doctor\_id,specialization from doctor where doctor\_id in(select doctor\_id from treatment where fromdate>='03-jun-2022'and todate<='03-oct-2022');

3.select \* from patient where patient\_no in(select patient\_no from treatment where todate>='01-aug-2022'and todate<='31-aug-2022');

1. 1.create or replace trigger delpatient before delete on patient for each row begin

if :old.dob >'01-jan-1910' then raise\_application\_error(-20101, 'not possible'); end if;

end;

/

2.create or replace trigger updoctor before update on doctor for each row begin

if :old.specialization !='pediatric' then raise\_application\_error(-20101, 'updation not possible'); end if;

end; /

**PROBLEM NO: 10**

## AIM: MONGODB

Create a database named “exam” and create a collection named “student” into it.

⦁ Insert the following document into the collection student(Using python code) \_id=1

Name=Anjali

Place=Kollam

Phone=8582639562

Vaccination\_status=”Both vaccinated”

RTPCR=negative

Lab\_mark=Internal:30,External:45

Department=MCA

⦁ Insert the following documents into the collection student(Using python code) \_id=2

Name=Anuradha

Place=Varkala

Phone=9992639562

Vaccination\_status=”Both vaccinated”

RTPCR=negative

Lab\_mark=Internal:40,External:48

Department=Civil

\_id=3

1

Name=Bismiya

Place=Kollam

Phone=9446639562

Vaccination\_status=”not vaccinated”

RTPCR=positive

Lab\_mark=Internal:50,External:39

Department=MCA

\_id=4

Name=Vimal

Place=Ernakulam

Phone=8582639568

Vaccination\_status=”First dose only”

RTPCR=positive

Lab\_mark=Internal:40,External:42

Department=Civil

\_id=5

Name=Vivek

Place=Kollam

Phone=8582639777

Vaccination\_status=”Both vaccinated”

RTPCR=negative

Lab\_mark=Internal:50,External:50

Department=MCA

⦁ Write a NoSQL Query to display the name and phone number of students who are both 2 not vaccinated

⦁ Write a NoSQL Query to display the name and phone number of top two students in MCA department(based on lab external mark)

⦁ Write a NoSQL Query to display the id, name and department of students whose name starts with ‘A’

⦁ Write a NoSQL Query to update the vaccination status as ‘both vaccinated’ of the student whose id=4

⦁ Write a NoSQL Query to display the name of students in descending order based on the lab external mark.

⦁ Write a NoSQL Query to remove students whose internal marks are less than or equal to 30.

**SOURCE CODE:**

import pymongo

myclient=pymongo.MongoClient("mongodb://localhost:27017/")

mydb=myclient["exam"]

mycol=mydb["student"]

mylist = [

{"\_id":1,"Name":"Anjali","Place":"Kollam","Phone":2582639562,"Vaccination\_status":"Both vaccinated","RTPCR":"negative","Lab\_mark":{"Internal":30,"External":45},"Depart ment":"MCA"},

{"\_id":2,"Name":"Anuradha","Place":"Varkala","Phone":9992639562,"Vaccination\_status":"Both vaccinated","RTPCR":"negative","Lab\_mark":{"Internal":40,"External":48},"Department":"Civil"}, {"\_id":3,"Name":"Bismiya","Place":"Kollam","Phone":9446639562,"Vaccination\_status":"not vaccinated","RTPCR":"positive","Lab\_mark":{"Internal":50,"External":39},"Depart ment":"MCA"},

{"\_id":4,"Name":"Vimal","Place":"Ernakulam","Phone":8582639568,"Vaccination\_status":"First dose only", "RTPCR": "positive", "Lab\_mark": {"Internal":40,"External":42},"Department":"Civil"},

{"\_id":5,"Name":"Vivek","Place":"Kollam","Phone":8582639777,"Vaccination\_status":"Both vaccinated","RTPCR":"negative","Lab\_mark":{"Internal":50,"External":50},"Depart ment":"MCA"}

]

x=mycol.insert\_many(mylist) print(x.inserted\_ids)

q3=mycol.find({"Vaccination\_status":"not vaccinated"}) for i in q3:

print(i["Name"]+" "+str(i["Phone"])+" "+(i["Vaccination\_status"])) print()

q4=mycol.find().sort("Lab\_mark.External",-1).limit(2) for s in q4:

print(s["Name"]+" "+str(s["Phone"])) print()

q5=mycol.find({"Name":{"$regex":"^A"}}) for q in q5:

print(str(q["\_id"])+" "+q["Name"]+" "+q["Department"]) print()

q6=mycol.update\_many({"\_id":4},{"$set" :{"Vaccination\_status":"Both vaccinated"}}) print(q6.modified\_count," documents updated ") print()

q7=mycol.find().sort("Lab\_mark.External",-1) for j in q7:

print(j["Name"]+" "+str(j["Phone"])) print()

q8=mycol.delete\_many({"Lab\_mark.Internal":{"$lte":30}})

print(q8.deleted\_count," documents deleted ") print()

**OUTPUT:**

[1, 2, 3, 4, 5]

================= RESTART: C:\Users\user\Desktop\Mongo\nosql.py

================

Bismiya 9446639562 not vaccinated

================= RESTART: C:\Users\user\Desktop\Mongo\nosql.py

================

Vivek 8582639777

Anuradha 9992639562

================= RESTART: C:\Users\user\Desktop\Mongo\nosql.py

================

1. Anjali MCA
2. Anuradha Civil

================= RESTART: C:\Users\user\Desktop\Mongo\nosql.py

================

1 documents updated

================= RESTART: C:\Users\user\Desktop\Mongo\nosql.py

================

Vivek 8582639777

Anuradha 9992639562

Anjali 2582639562 Vimal 8582639568

Bismiya 9446639562

================= RESTART: C:\Users\user\Desktop\Mongo\nosql.py

================

1 documents deleted