

## SQL ass 1:

### SQL Queries:

- Design and Develop SQL DDL statements which demonstrate the use of SQL objects such as (Table, View, Index, Sequence, Synonym, different constraints etc.)
- Write at least 10 SQL queries on the suitable database application using SQL DML statements. (Insert, Select, Update, Delete with operators, Functions, and set operator)

Sr. No.	Key	DDL	DML
1	Stands for	DDL stands for Data Definition Language.	DML stands for Data Manipulation Language.
2	Usage	DDL statements are used to create database, schema, constraints, users, tables etc.	DML statement is used to insert, update or delete the records.
3	Classification	DDL has no further classification.	DML is further classified into procedural DML and non-procedural DML.
4	Commands	CREATE, DROP, RENAME and ALTER.	INSERT, UPDATE and DELETE.

Effect	The DDL command affects the entire database or table.	The DML commands will affect the single or multiple records based on the specified condition.
--------	---	---

(NOTE: cannot create synonym for table but for database in Mysql)



```
\sql  
\connect root@localhost:3306  
password  
SHOW DATABASES;  
CREATE DATABASE ass1;  
USE ass1;
```

(Note: MUL means foreign key that references to em\_id of employee)

```
CREATE TABLE employee(  
    em_id INT,  
    em_name CHAR(20),  
    em_join_date DATE,  
    PRIMARY key(em_id)  
);  
DESC employee;
```

```
CREATE TABLE emp_finance(  
    pay_id INT auto_increment,  
    em_id INT,  
    em_salary INT,  
    em_contract DATE,  
    FOREIGN key(em_id) REFERENCES employee(em_id),  
    PRIMARY key(pay_id)  
);  
DESC emp_finance;
```

```
INSERT INTO employee VALUES(1, 'sham', '2020-09-22');  
INSERT INTO employee VALUES(2, 'ram', '2020-09-12');  
INSERT INTO employee VALUES(3, 'ram', '2020-09-15');  
SELECT * FROM employee;
```

```
INSERT INTO emp_finance VALUES(1,1,2000,'2021-01-21');  
INSERT INTO emp_finance VALUES(null,3,4000,'2021-04-21');  
SELECT * FROM emp_finance;
```

```
CREATE VIEW EmpDetails AS  
SELECT e.em_id, e.em_name, f.em_salary FROM employee AS e  
INNER JOIN emp_finance AS f  
WHERE e.em_id = f.em_id;
```

```
SELECT * FROM EmpDetails;
```

```
ALTER TABLE emp_finance ADD INDEX salary (em_salary);
```

```
CALL sys.create_synonym_db('ass1', 'assignment1');  
show databases;  
use assignment1;  
show tables;
```

```
CREATE TABLE employee(  
    em_id INT,  
    em_name CHAR(20),  
    em_join_date DATE,  
    PRIMARY key(em_id)  
);  
DESC employee;  
  
CREATE TABLE emp_finance(  
    pay_id INT auto_increment,  
    em_id INT,  
    em_salary INT,  
    em_contract DATE,  
    FOREIGN key(em_id) REFERENCES employee(em_id),  
    PRIMARY key(pay_id)  
);  
DESC emp_finance;  
  
INSERT INTO employee VALUES(1, 'sham', '2020-09-22');  
INSERT INTO employee VALUES(2, 'ram', '2020-09-12');  
INSERT INTO employee VALUES(3, 'ram', '2020-09-15');  
SELECT * FROM employee;  
  
INSERT INTO emp_finance VALUES(1,1,2000,'2021-01-21');  
INSERT INTO emp_finance VALUES(null,3,4000,'2021-04-21');  
SELECT * FROM emp_finance;  
  
CREATE VIEW EmpDetails AS  
SELECT e.em_id, e.em_name, f.em_salary FROM employee AS e  
INNER JOIN emp_finance AS f  
WHERE e.em_id = f.em_id;  
  
SELECT * FROM EmpDetails;  
  
ALTER TABLE emp_finance ADD INDEX salary (em_salary);  
  
CALL sys.create_synonym_db('ass1', 'assignment1');  
show databases;  
use assignment1;  
show tables;
```

Insert, select, update, delete

Using

Operators -> in, not in, limit, is null, like, set, between

Functions ->

Strings : upper, lower, length, concat, substring

Numeric: abs, mod, sqrt, power

Date: curdate, curtime, year, month, day

Aggregate: avg, sum, min, max, count

Create database ass1b;

Use ass1b;

```
CREATE TABLE emp(emp_id int primary key auto_increment,  
emp_fname char(20) not null, emp_lname char(20) not null,  
emp_salary int, join_date date);
```

Desc emp;

```
INSERT INTO emp values(1, 'Ram', 'Kapoor', 40000, '2019-02-23');
```

```
INSERT INTO emp values(null, 'Ramesh', 'Kapoor', 20000, '2019-02-24');
```

```
INSERT INTO emp values(null, 'Ramu', 'Kapoor', 4000, '2019-02-25');
```

```
Select * FROM emp;
```

```
UPDATE emp set emp_salary=10000 where emp_id=3;
```

```
DELETE FROM emp where emp_id=3;
```

```
SELECT AVG(emp_salary) FROM emp;
```

```
SELECT concat(emp_fname, emp_lame) FROM emp where emp_salary in (SELECT  
max(emp_salary) FROM emp);
```

```
SELECT upper(concat(emp_fname, emp_lame)) FROM emp where emp_salary in (SELECT  
max(emp_salary) FROM emp);
```

```
SELECT * FROM emp WHERE YEAR(join_date)='2015';
```

```
SELECT * FROM emp WHERE emp_name like 'Ram%';
```

```
SELECT * FROM emp WHERE emp_id in (1,2,3);
```

```
SELECT * FROM emp WHERE emp_id not in (1,2,3);
```

```
SELECT * FROM emp WHERE YEAR(join_date) between '2015' and '2018';
```

```

CREATE database ass1b;
Use ass1b;
CREATE TABLE emp(
    emp_id INT PRIMARY key auto_increment,
    emp_fname CHAR(20) NOT NULL,
    emp_lname CHAR(20) NOT NULL,
    emp_salary INT,
    join_date DATE
);
DESC emp;

INSERT INTO emp VALUES(1, 'Ram', 'Kapoor', 40000, '2019-02-23');
INSERT INTO emp VALUES(null, 'Ramesh', 'Kapoor', 20000, '2019-02-24');
INSERT INTO emp VALUES(null, 'Ram', 'Kapoor', 4000, '2019-02-25');
SELECT * FROM emp;

UPDATE emp SET emp_salary = 10000 WHERE emp_id = 3;

DELETE FROM emp WHERE emp_id = 3;

SELECT AVG(emp_salary) FROM emp;

SELECT concat(emp_fname, emp_lame) FROM emp
WHERE emp_salary IN (SELECT MAX(emp_salary) FROM emp);

SELECT upper(concat(emp_fname, emp_lame)) FROM emp WHERE emp_salary IN (SELECT
max(emp_salary) FROM emp);
SELECT * FROM emp WHERE YEAR(join_date)='2015';
SELECT * FROM emp WHERE emp_name like 'Ram%';
SELECT * FROM emp WHERE emp_id IN (1,2,3);
SELECT * FROM emp WHERE emp_id NOT IN (1,2,3);
SELECT * FROM emp WHERE YEAR(join_date) BETWEEN '2015' AND '2018';

```

## SQL Assignment 2:

SQL Queries – all types of Join, Sub-Query and View:

Write at least 10 SQL queries for suitable database application using SQL DML statements. Note: Instructor will design the queries which demonstrate the use of concepts like all types of Join, Sub-Query and View

An **SQL Join statement** is used to combine data or rows from two or more tables based on a common field between them.

A **subquery** is a query that is nested inside a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement, or inside another subquery.

Joins and subqueries are both used to combine data from different tables into a single result.

## What is a Subquery?

A *subquery* is a nested query (inner query) that's used to filter the results of the outer query. Subqueries can be used as an alternative to joins. A subquery is typically nested inside the `WHERE` clause.

Syntax:

```
SELECT <columns>
```

```
FROM <table>
```

```
WHERE <column> <operator>
```

```
(SELECT <columns>
```

```
FROM <table>
```

```
)
```

(NOTE: Mysql doesn't support full joins)

```

Create database ass2;
Use ass2;
Create table student(roll_no int primary key, name char(20), address char(40), phone int(10),
age int(2));
Desc student;
INSERT INTO student values(1,'Harsh', 'delhi', '11', '18');
INSERT INTO student values(2,'Harshal', 'mumbai', '12', '16');
INSERT INTO student values(3,'prathamesh ', 'pune', '13', '20');
INSERT INTO student values(4,'Shreyash ', 'chennai', '14', '21');
CREATE TABLE studentcourse (cid int(1), roll_no int, foreign key(roll_no) references
student(roll_no));
Desc studentcourse;
INSERT INTO studentcourse values(1,2);
INSERT INTO studentcourse values(2,1);
INSERT INTO studentcourse values(3,3);
INSERT INTO studentcourse values(4,4);
SELECT * FROM student
Inner join studentcourse
ON student.roll_no = studentcourse.roll_no;
SELECT * FROM student
left join studentcourse
ON student.roll_no = studentcourse.roll_no;
SELECT * FROM student
Right join studentcourse
ON student.roll_no = student.course.roll_no;
SELECT * FROM student
join studentcourse;
SELECT * FROM student;
CREATE VIEW newView AS
SELECT student.name, student.roll_no, studentcourse.cid FROM student
LEFT JOIN studentcourse
ON student.roll_no = studentcourse.roll_no;
SELECT * FROM newView;
SELECT * FROM student WHERE roll_no in
(SELECT roll_no FROM studentcourse);
SELECT * FROM student
WHERE roll_no in
(SELECT roll_no FROM studentcourse
WHERE student.age < 20);

```



```

CREATE database ass2;
Use ass2;

CREATE TABLE student(
    roll_no INT PRIMARY key,
    name CHAR(20),
    address CHAR(40),
    phone INT(10),
    age INT(2)
);
DESC student;

INSERT INTO student values(1,'Harsh', 'delhi', '11', '18');
INSERT INTO student values(2,'Harshal', 'mumbai', '12', '16');
INSERT INTO student values(3,'prathamesh ', 'pune', '13', '20');
INSERT INTO student values(4,'Shreyash ', 'chennai', '14', '21');

CREATE TABLE studentcourse (
    cid INT(1),
    roll_no INT,
    FOREIGN key(roll_no) REFERENCES student(roll_no)
);
DESC studentcourse;

INSERT INTO studentcourse values(1,2);
INSERT INTO studentcourse values(2,1);
INSERT INTO studentcourse values(3,3);
INSERT INTO studentcourse values(4,4);

SELECT * FROM student
Inner join studentcourse
ON student.roll_no = studentcourse.roll_no;

SELECT * FROM student
left join studentcourse
ON student.roll_no = studentcourse.roll_no;

SELECT * FROM student
Right join studentcourse
ON student.roll_no = studentcourse.roll_no;

SELECT * FROM student
join studentcourse;

SELECT * FROM student;

CREATE VIEW newView AS
SELECT student.name, student.roll_no, studentcourse.cid FROM student
LEFT JOIN studentcourse
ON student.roll_no = studentcourse.roll_no;

SELECT * FROM newView;

SELECT * FROM student WHERE roll_no in
(SELECT roll_no FROM studentcourse);

SELECT * FROM student
WHERE roll_no in
(SELECT roll_no FROM studentcourse
WHERE student.age < 20);

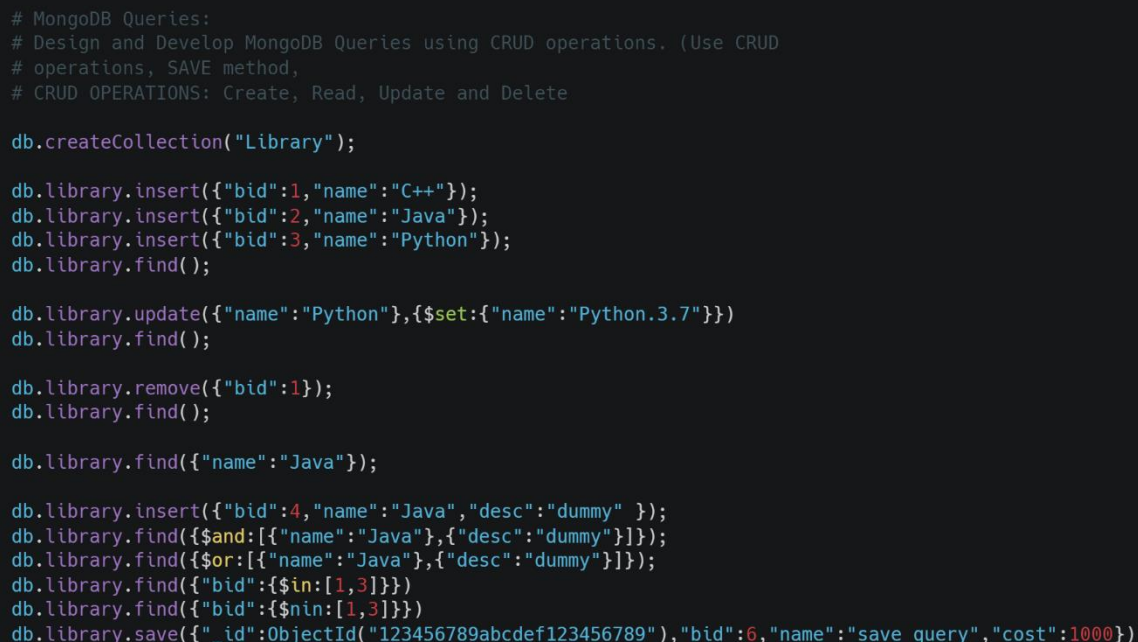
```

## MongoDB Queries:

Design and Develop MongoDB Queries using CRUD operations. (Use CRUD operations, SAVE method,

CRUD OPERATIONS: Create, Read, Update and Delete

```
db.createCollection("Library");
db.library.insert({"bid":1,"name":"C++"});
db.library.insert({"bid":2,"name":"Java"});
db.library.insert({"bid":3,"name":"Python"});
db.library.find();
db.library.update({"name":"Python"},{$set:{"name":"Python.3.7"}})
db.library.find();
db.library.remove({"bid":1});
db.library.find();
db.library.find({"name":"Java"});
db.library.insert({"bid":4,"name":"Java","desc":"dummy" });
db.library.find({$and:[{"name":"Java"},{"desc":"dummy"}]});
db.library.find({$or:[{"name":"Java"},{"desc":"dummy"}]});
db.library.find({"bid":{"$in":[1,3]}})
db.library.find({"bid":{"$nin":[1,3]}})
db.library.save({"_id":ObjectId("123456789abcdef123456789"),"bid":6,"name":"save query","cost":1000})
```



```
# MongoDB Queries:
# Design and Develop MongoDB Queries using CRUD operations. (Use CRUD
# operations, SAVE method,
# CRUD OPERATIONS: Create, Read, Update and Delete

db.createCollection("Library");

db.library.insert({"bid":1,"name":"C++"});
db.library.insert({"bid":2,"name":"Java"});
db.library.insert({"bid":3,"name":"Python"});
db.library.find();

db.library.update({"name":"Python"},{$set:{"name":"Python.3.7"}})
db.library.find();

db.library.remove({"bid":1});
db.library.find();

db.library.find({"name":"Java"});

db.library.insert({"bid":4,"name":"Java","desc":"dummy" });
db.library.find({$and:[{"name":"Java"},{"desc":"dummy"}]});
db.library.find({$or:[{"name":"Java"},{"desc":"dummy"}]});
db.library.find({"bid":{"$in":[1,3]}})
db.library.find({"bid":{"$nin":[1,3]}})
db.library.save({"_id":ObjectId("123456789abcdef123456789"),"bid":6,"name":"save query","cost":1000})
```

Unnamed PL/SQLcode block: Use of Control structure and Exception handling is mandatory.

Suggested Problem statement:

Consider Tables:

1. Borrower (Roll\_no, Name, Date\_of\_Issue, Name\_of\_Book, Status)

2. Fine (Roll\_no, Date, Amt)

- Accept Roll\_no and Name\_of\_Book from user.
- Check the number of days (from Date\_of\_Issue).
- If days are between 15 to 30 then fine amount will be Rs 5per day.
- If no. of days>30, per day fine will be Rs 50 per day and for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.
- If condition of fine is true, then details will be stored into fine table.
- Also handles the exception by named exception handler or user define exception handler

SQL -> Declarative (Tells what to do)

PL-SQL (Procedural SQL) -> (What to do + How to do)

PL -SQL Block/Code

Declaration

a int

b int

c int

Executable code

begin

a:=10;

b:=10;

c:= a + b;

end

Exception Handling

Error

S.No	Parameter Mode & Description
1	<p><b>IN</b></p> <p>An IN parameter lets you pass a value to the subprogram. <b>It is a read-only parameter.</b> Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. <b>It is the default mode of parameter passing. Parameters are passed by reference.</b></p>
2	<p><b>OUT</b></p> <p>An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. <b>The actual parameter must be variable and it is passed by value.</b></p>

- **Functions** – These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** – These subprograms do not return a value directly; mainly used to perform an action.

# Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts –

S.No	Parts & Description
1	<b>Declarative Part</b> It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
2	<b>Executable Part</b> This is a mandatory part and contains statements that perform the designated action.
3	<b>Exception-handling</b> This is again an optional part. It contains the code that handles run-time errors.

## DECLARE

```
<< declaration section >>
```

## BEGIN

```
<<Block of executable code>>
```

## EXCEPTION

```
<< exception handling >>
```

```
WHEN excp1 THEN
```

```
<< excp1 handling block >>
```

```
WHEN excp2 THEN
```

```
<< excp2 handling block >>
```

```
.....
```

```
WHEN others THEN
```

```
<< excp2 handling block>>
```

## END;

```

create database ass4;
use ass4;
Create table borrower(rollin int(11) not null primary key, name char(20) not null, dateofIssue date,
bname char(20), status char(1));
Create table fine(rollno int(11), fdate date, amt int(11), foreign key(rollno) references
borrower(rollin));
desc borrower;
desc fine;
INSERT INTO borrower values(1,'a', '2022-11-01', 'java', 'T');
INSERT INTO borrower values(2,'b', '2022-10-01', 'networking', 'T');
INSERT INTO borrower values(3,'c', '2022-09-01', 'DBMS', 'T');
INSERT INTO borrower values(4,'d', '2022-08-01', 'CN', 'T');
SELECT * FROM fine;

```

(we use delimiter to push our code in block and not one by one)

delimiter \$

```

CREATE procedure fine_calculation (IN rno int(3), bname char(20))
begin
Declare i_date date;
Declare diff int;
Declare fine_amt int;
Declare exit handler for sqlexception select 'Table not found';
SELECT dateofIssue into i_date from borrower where rollin=rno and bname=bname;
Select datediff(curdate(), i_date) into diff;
If(diff>15 and diff<=30) then
Set fine_amt = diff*5;
Insert into fine values(rno, curdate(), fine_amt);
Elseif(diff>30) then
Set fine_amt=15*5 + (diff-30)*50;
Insert into fine values(rno, curdate(),fine_amt);
End if;
Update borrower set status='R' where rollin=rno and bname=bname;
End
$
Delimiter ;
Call fine_calculation(3, 'DBMS');
SELECT * FROM fine;
SELECT * FROM borrower;

```

(NOTE: here rno in rollin = rno and bname in bname = name are formal parameters of the procedure)

(At last the delimiter is changed back to ; as it was \$ first)

```

create database ass4;
use ass4;
Create table borrower(rollin int(11) not null primary key, name char(20) not null, dateofIssue date,
bname char(20), status char(1));
Create table fine(rollno int(11), fdate date, amt int(11), foreign key(rollno) references
borrower(rollin));
desc borrower;
desc fine;
INSERT INTO borrower values(1,'a', '2022-11-01', 'java', 'T');
INSERT INTO borrower values(2,'b', '2022-10-01', 'networking', 'T');
INSERT INTO borrower values(3,'c', '2022-09-01', 'DBMS', 'T');
INSERT INTO borrower values(4,'d', '2022-08-01', 'CN', 'T');
SELECT * FROM fine;

```

```

delimiter $
CREATE procedure fine_calculati(IN rno int(3), bname char(20))
begin
Declare i_date date;
Declare diff int;
Declare fine_amt int;
Declare exit handler for sqlexception select 'Table not found';
SELECT dateofIssue into i_date from borrower where rollin=rno and bname=bname;
Select datediff(curdate(), i_date) into diff;
If(diff>15 and diff<=30) then
Set fine_amt = diff*5;
Insert into fine values(rno, curdate(), fine_amt);
Elseif(diff>30) then
Set fine_amt=15*5 + (diff-30)*50;
Insert into fine values(rno, curdate(),fine_amt);
End if;
Update borrower set status='R' where rollin=rno and bname=bname;
End
$
Delimiter ;
Call fine_calculati(3, 'DBMS');
SELECT * FROM fine;
SELECT * FROM borrower;

```





```

CREATE database ass4;
use ass4;

CREATE TABLE borrower(
    rollin INT(11) NOT NULL PRIMARY key,
    name CHAR(20) NOT NULL,
    dateofIssue DATE,
    bname CHAR(20),
    status CHAR(1)
);
CREATE TABLE fine(
    rollno INT(11),
    fdate DATE,
    amt INT(11),
    FOREIGN key(rollno) REFERENCES borrower(rollin)
);

DESC borrower;
DESC fine;

INSERT INTO borrower values(1,'a', '2022-11-01', 'java', 'I');
INSERT INTO borrower values(2,'b', '2022-10-01', 'networking',
INSERT INTO borrower values(3,'c', '2022-09-01', 'DBMS', 'I');
INSERT INTO borrower values(4,'d', '2022-08-01', 'CN', 'I');
SELECT * FROM fine;

delimiter $

CREATE PROCEDURE fine_calculation (IN rno INT(3), bname CHAR(20))
BEGIN
    DECLARE i_date DATE;
    DECLARE diff INT;
    DECLARE fine_amt INT;
    DECLARE EXIT handler FOR SQLEXCEPTION SELECT 'Table not found';

    SELECT dateofIssue INTO i_date FROM borrower
    WHERE rollin=rno AND bname=bname;

    SELECT datediff(curdate(), i_date) INTO diff;
    IF(diff>15 AND diff<=30)
    THEN
        SET fine_amt = diff*5;
        INSERT INTO fine VALUES(rno, curdate(), fine_amt);

    ELSEIF(diff>30)
    THEN
        SET fine_amt=15*5 + (diff-30)*50;
        INSERT INTO fine VALUES(rno, curdate(),fine_amt);
    END IF;

    UPDATE borrower SET status='R'
    WHERE rollin=rno AND bname=bname;

END
$ delimiter ;

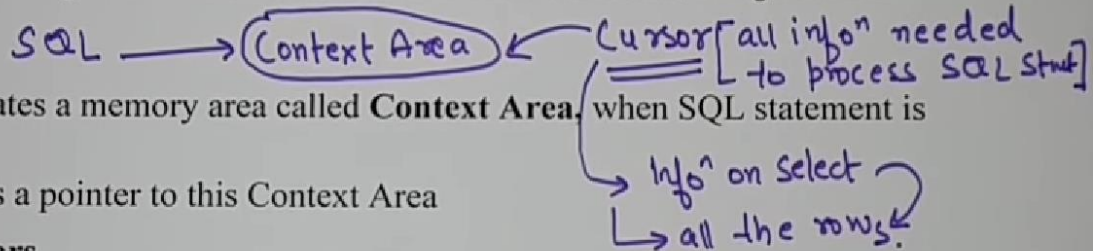
Call fine_calculation(3, 'DBMS');
SELECT * FROM fine;
SELECT * FROM borrower;

```

Cursors: (All types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor)

Write a PL/SQL block of code using parameterized Cursor that will merge the data available in the newly created table N\_Roll\_Call with the data available in the table O\_Roll\_Call. If the data in the first table already exists in the second table then that data should be skipped.

### Cursor



- Oracle creates a memory area called **Context Area** when SQL statement is processed
- A Cursor is a pointer to this Context Area

### Types of Cursors

PL SQL Implicit Cursors	PL SQL Explicit Cursor
<p>automatically → SQL Stmt.</p> <p>i) %FOUND → True (DML affects one or more row) Select (one " ")</p> <p>ii) %NOTFOUND → always false for implicit</p> <p>iii) %ISOPEN → always false for implicit</p> <p>iv) %ROWCOUNT → no. of rows affected.</p>	<p>→ declared explicitly by developer.</p> <p>→ Can be manipulated.</p> <pre> graph LR     Declare[Declare] --&gt; Open[open]     Open --&gt; Fetch[Fetch]     Fetch --&gt; Empty{Empty}     Empty --&gt; Close[close]                     </pre>

### PL SQL Explicit Cursor Syntax:

Declare Cursor	Open Cursor	Fetch Cursor	Close Cursor
<p>CURSOR cursor-name IS SELECT_stmt; cursor c1 is Select * from emp;</p>	<p>OPEN cursor-name; open c1;</p>	<p>Fetch cursor-name into variable list / record Fetch c1 into v-i;</p>	<p>CLOSE cursor-name CLOSE c1;</p>

- Used to Gain More Control over Context Area
- Defined in the Declaration Section

It is created on a **SELECT** statement which returns more than one row.

Select sal into v\_sal from emp;

↳ Cursor.

```

Create table o_rollcall(rno int(11) not null primary key, name varchar(20), addr varchar(20));
Create table n_rollcall(rno int(11), name varchar(20), addr varchar(20));
Desc o_rollcall;
Desc n_rollcall;
Delimiter //
Create procedure n1(IN rno1 int)
Begin
Declare rno2 int;
Declare exit_cond Boolean;
Declare c1 cursor for select rno from o_rollcall where rno>rno1;
Declare continue handler for not found set exit_cond=TRUE;
Open c1;
L1: loop
Fetch c1 into rno2;
If not exists(select * from n_rollcall where rno=rno2) then
Insert into n_rollcall select * from o_rollcall where rno=rno2;
End if;
If exit_cond then
Close c1;
Leave l1;
End if;
End loop l1;
End;
//
Delimiter ;
Call n1(3);
Select * from n_rollcall;

```

```

CREATE TABLE o_rollcall ( rno INT(11) NOT NULL PRIMARY KEY, name VARCHAR(20), addr VARCHAR(20));
CREATE TABLE n_rollcall ( rno INT(11), name VARCHAR(20), addr VARCHAR(20) );

DESC o_rollcall;
DESC n_rollcall;

INSERT INTO o_rollcall VALUES(1,'sunny','pune');
INSERT INTO o_rollcall VALUES(2,'bunny','chennai');
INSERT INTO o_rollcall VALUES(3,'lonny','mumbai');
INSERT INTO o_rollcall VALUES(4,'donny','delhi');
INSERT INTO o_rollcall VALUES(5,'ramesh','kerela');
SELECT * FROM o_rollcall;

INSERT INTO n_rollcall VALUES(1,'sunny','pune');
INSERT INTO n_rollcall VALUES(2,'bunny','chennai');
INSERT INTO n_rollcall VALUES(3,'lonny','mumbai');
Select * FROM n_rollcall;

Delimiter //
CREATE PROCEDURE n1(IN rno1 INT)
BEGIN
    DECLARE rno2 INT;
    DECLARE exit_cond BOOLEAN;
    DECLARE c1 CURSOR FOR

    SELECT rno FROM o_rollcall
    WHERE rno > rno1;

    DECLARE CONTINUE handler FOR NOT found SET exit_cond = TRUE;

    OPEN c1;
    l1: LOOP
        FETCH c1 INTO rno2;

        IF NOT EXISTS
        (
            SELECT * FROM n_rollcall
            WHERE rno = rno2
        )
        THEN
            INSERT INTO n_rollcall
            SELECT * FROM o_rollcall
            WHERE rno = rno2;

        END IF;

    IF exit_cond
    THEN
        CLOSE c1;
        LEAVE l1;
    END IF;

    END
    LOOP l1;
END;
//
delimiter;

CALL n1(3);

SELECT * FROM n_rollcall;

```