Name:- Karan mane
Roll No. :- 30
Sub:- ADMT
D.O.S :-20/11/2020

EXPERIMENT NO : 1

AIM: To implement Advanced SQL Queries.

Theory:

The company database consists of Employees, Department and Projects.The company is organized into departments. Each department has unique name,number and particular employee who manages department. A department may have several locations.A department controls number of projects each of which has a unique name,number and single location.

We store each employee's name, SSN, Address, salary and DOB

An employee is assigned to one department but may work on each project.

We also keep track of direct superior of each project.

We want to keep track of dependence of each employee for purpose. We need first

name, DOB and relation to the employee.

Relational Model

Relational model is a tabular form of database that database is called schema denoted by's'.

Relation is an entity set schema is asset of relation S= {R1, R2, R3…}

Steps for converting ER model to relational model

1. Mapping of strong entity type.
2. Mapping of weak entity type.
3. Mapping of binary 1:1 relationship
4. Mapping of binary 1:N relationship
5. Mapping of binary M:N relationship
6. Mapping of multivalued attributes
7. Mapping of n array relationship

Mapping of strong entity type

A. Employee
B. Department
C. Project

Create table for each entity

Entity's attribute should fields of tables with their respective key

Mapping of weak entity type

Create table for weak entity set

Add all its attribute to table or field
Add the primary key of identifying entity set

Mapping of binary 1:1 relationship
Create table for relationship
Add foreign key for one of the relation

Mapping of binary 1: N relationship

Mapping of binary M: N relationship

If relationship is M: N then have to create separate relation of relationship. The attribute of relation are the primary key of referring table.

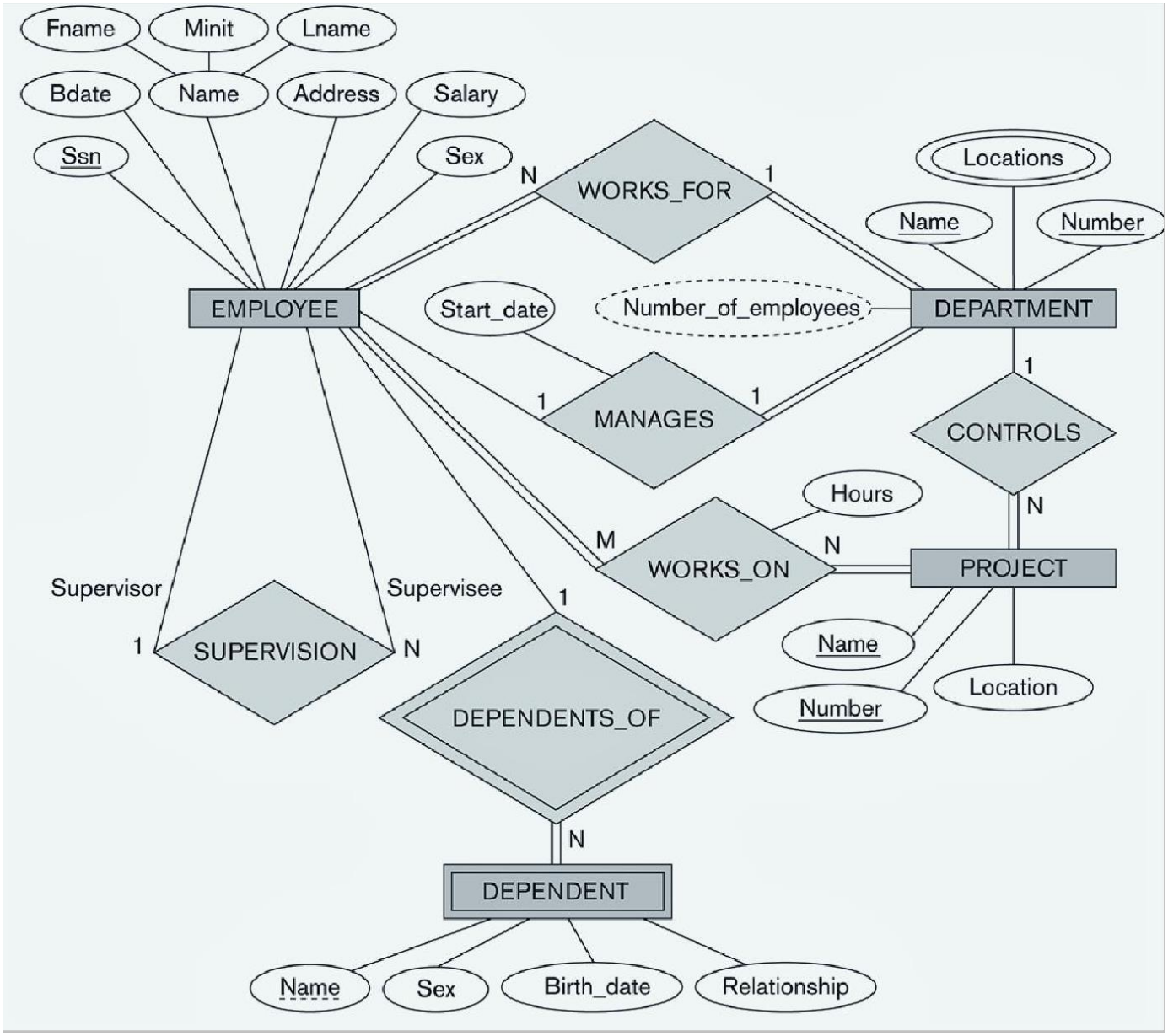
Mapping of multivalued attributes

Mapping of n array relationship

N array relationship

If n array relationship we have to scale separate location. Attribute is primary key of all the referring table.

Draw ER model for company database.



Create all tables using DDL command

Creating a Database CREATE DATABASE Company;

Viewing all databases SHOW DATABASES;

Viewing all Tables in a Database, SHOW tables;

```
CREATE TABLE Employee(
SSN INT(9) NOT NULL PRIMARY KEY,
Fname VARCHAR(20) NOT NULL,
Minit VARCHAR(15) NOT NULL,
Lname VARCHAR(20) NOT NULL,DOB VARCHAR(15),
Address VARCHAR(50),
Salary INT(8),
Dno INT(9));

CREATE TABLE Department(
Dname VARCHAR(10),
Dnumber INT(9),
Mgr_ssn INT(9));

CREATE TABLE Dept_Locations(
Dnumber INT(9),
Dlocation VARCHAR(10));

CREATE TABLE Project(
Pname VARCHAR(20),
Pnumber INT(9),
Plocation VARCHAR(10),
Dnum INT(9));

CREATE TABLE Works_On(
Essn INT(9),
Pno INT(9),
Hours INT(4));

CREATE TABLE Dependent(
Essn INT(9),
Dependent_name VARCHAR(20),
BOD INT(7));
```

Insert values in the table

INSERT INTO Employee (SSN,Fname,Minit,Lname,DOB,Address,Salary,Dno)
VALUES ('1','karan','k','Mane','27/08/00,'Malad','100000','5');

INSERT INTO Employee (SSN,Fname,Minit,Lname,DOB,Address,Salary,Dno)
VALUES ('2','Shubham','S','Mane','15/08/01','Malad','4000','5');

INSERT INTO Department (Dname,Dnumber,Mgr_ssn,Mgr_start_date)
VALUES ('Technical','3','2','5/9/17');

INSERT INTO Department (Dname,Dnumber,Mgr_ssn,Mgr_start_date)
VALUES ('Accounting','5','1','7/2/18');

Write SQL queries for-

1)Retrieve the name of the employee whose salary is greater than the salary of all employees work in department 5

Ans:

SQL Query

SELECT Fname,Minit,Lname FROM Employee WHERE Dno='5' AND Salary=(SELECT MAX(Salary) FROM Employee WHERE Dno='5') ;

2)Retrieve name of employees who have no dependent.

Ans:

SQL Query

SELECT Fname,Minit,Lname FROM Employee WHERE NOT EXISTS (SELECT * FROM Dependent WHERE SSN=Essn) ;

3)For each department,retrieve the department name, the no. of employee in the department, and their average salary.

Ans:

SQL Query

SELECT Dno,Count(*),Avg(Salary) FROM Employee GROUP BY Dno;

EXPERIMENT NO : 2

Aim: To implement active database using trigger.

Theory:

ECA model

Triggers are active rules, which can be fired when some conditions are satisfied. For eg. It may be useful to specify condition that, if violation. The model that has been used to satisfy active database in referred to as the event condition action model has 3 components.

1. Event

Event that triggers the rule, these are usually database update operation that are explicitly applied to the database. However in general model, they could also be temporary events or other kind of external events.

2. Condition

That determines whether the rule action should be executed. Once the triggering event has occurring and optional condition. May be evaluated if no condition is specified the action will be executed.

3. Action

The action should be taken, the action is usually a sequence of SQL statements but it could also be database transaction or an external program that will be automatically executed.

```
SQL&gt; CREATE TABLE semp
2 (
3 emp_id NUMBER(3) PRIMARY KEY,
4 Name VARCHAR(10) NOT NULL,
5 Salary NUMBER(7),
6 DNumber NUMBER(3)
7 );



2 (
3 DNo NUMBER(3) PRIMARY KEY,

4 Name VARCHAR(10) NOT NULL,
5 Total_Salary NUMBER(7)
6 );
Table created.
SQL&gt; ALTER TABLE semp
```

```
2 ADD FOREIGN KEY (DNumber)
3 REFERENCES sdept(DNo);
Table altered.
SQL&gt; CREATE TRIGGER insert_trigger
2 AFTER INSERT ON semp 3 FOR EACH
ROW
4 BEGIN
5 IF(:NEW.DNumber!=0)
6 THEN UPDATE sdept
7 SET Total_Salary = Total_Salary + :NEW.Salary
8 WHERE :new.DNumber=Dno;
9 END IF;
10 END;
11/
Trigger created.
SQL&gt; CREATE TRIGGER delete_trigger
2 AFTER DELETE ON semp 3 FOR EACH
ROW
4 BEGIN
5 IF(:OLD.DNumber!=0)
6 THEN UPDATE sdept
7 SET Total_Salary = Total_Salary - :OLD.Salary
8 WHERE :OLD.DNumber=Dno;
9 END IF;
10 END;
11/
Trigger created.
SQL&gt; CREATE TRIGGER update_trigger
2 AFTER UPDATE ON semp 3 FOR EACH
ROW
4 BEGIN
5 IF(:NEW.DNumber!=0)
6 THEN UPDATE sdept
7 SET Total_Salary = Total_Salary + (:NEW.Salary - :OLD.Salary)
8 WHERE :new.DNumber=Dno;
9 END IF;
10 END;
11/

Trigger created.
```

SQL> INSERT INTO sdept(DNo,Name,Total_Salary)
values(1,'Sales',0); 1 row created.
SQL> INSERT INTO semp (emp_id,Name,Salary,DNumber)
values(1,'Sanchit',55000,1);
1 row created.
SQL> SELECT * FROM semp;
EMP_ID NAME SALARY DNUMBER
---------- ---------- ---------- ----------
1 Sanchit 55000 1
SQL> SELECT * FROM sdept;
DNO NAME TOTAL_SALARY
---------- ---------- ------------
1 Sales 55000
SQL> UPDATE semp SET Salary = 75000 WHERE emp_id=1;

EXPERIMENT NO : 3

AIM : Simulation of serializability for concurrency control.

Theory :- 1) Serializability is used to keep the data in the data item in a consistent state. Serializability is a property of a transaction schedule

2) Serializability of a schedule means equivalence (in the outcome, the database state, data values) to a serial schedule (i.e., sequential with no transaction overlap in time) with the same transactions.

3) It is the major criterion for the correctness of concurrent transactions' schedule, and thus supported in all general purpose database systems

4) In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions.

5) We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.

6) Concurrency control protocols can be broadly divided into two categories –

a) Lock based protocols
b) Time stamp based protocols

CODE:

```
non_serial_operation = []
t1_list = []
t2_list = []
t3_list = []
while True:
operations = input('Enter your Non Serial schedule one by one (write done
after complete): ').lower()
if operations == 'done' :
print('Okay Completed')
break
non_serial_operation.append(operations)
# print(non_serial_operation)
for i in non_serial_operation: if
i[1] == '1':
t1_list.append(i)
elif i[1] == '2':
t2_list.append(i)
elif i[1] == '3':
t3_list.append(i)
# print(t1_list,t2_list, t3_list)
conflict_pair_list = []
for i in non_serial_operation:
if i[0] == 'r':
if i[1] == '1': for j in
t2_list: pair_list =
[]
pair_list.append(i)
```

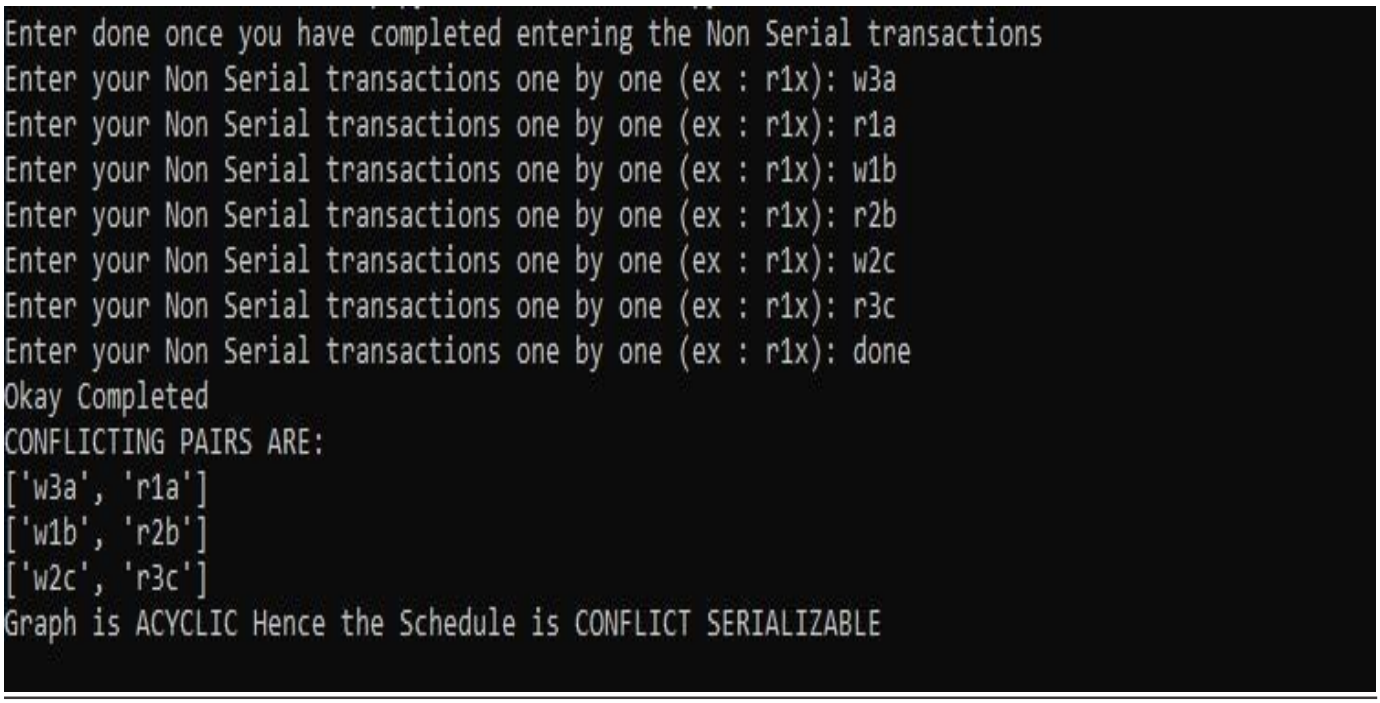
```
if j[0] == 'w' and j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

for j in t3_list:
pair_list = []
pair_list.append(i)
if j[0] == 'w' and j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

t1_list.remove(i)
elif i[1] == '2':
for j in t1_list:
pair_list = []
pair_list.append(i)
if j[0] == 'w' and j[-1] == i[-1]:
```

```
pair_list.append(j)

conflict_pair_list.append(pair_list)

for j in t3_list:
pair_list = []

pair_list.append(i)
if j[0] == 'w' and j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

t2_list.remove(i)
elif i[1] == '3':
for j in t2_list:
pair_list = []
pair_list.append(i)
if j[0] == 'w' and j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

for j in t1_list:
pair_list = []
pair_list.append(i)
if j[0] == 'w' and j[-1] == i[-1]:
pair_list.append(j)
conflict_pair_list.append(pair_list)
t3_list.remove(i)
elif i[0] == 'w':
if i[1] == '1':
for j in t2_list:
pair_list = []
pair_list.append(i)
if j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

for j in t3_list:
pair_list = []
pair_list.append(i)
if j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

t1_list.remove(i)
elif i[1] == '2':
for j in t1_list:
pair_list = []
pair_list.append(i)
```

```
if j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

for j in t3_list:
pair_list = []
pair_list.append(i)
if j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

t2_list.remove(i)
elif i[1] == '3':
for j in t2_list:
pair_list = []
pair_list.append(i)
if j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

for j in t1_list:
pair_list = []
pair_list.append(i)
if j[-1] == i[-1]:
pair_list.append(j)

conflict_pair_list.append(pair_list)

t3_list.remove(i)
print('CONFLICTING PAIRS ARE: ')
for i in conflict_pair_list:
print(i)

adj = {}
for i in conflict_pair_list:
key = i[0][1]
adj.setdefault(key,set())
adj[key].add(i[1][1])
color = {}
for u in adj.keys():
color[u] = 'W'
def dfs(u , color ):
color[u] = "G"
# print(u)
for v in adj[u]:
try:
if color[v] == 'W':
dfs(v , color)
if cycle_present == True:
return True
```

```
elif color[v] == 'G':
# print('CYCLE FOUND in GRAPH at
NODE',u,',',v) return True
except :
return False
color[u] = "B"
return False

cycle_present = False
for u in adj.keys():
if color[u] == "W":
cycle_present =dfs(u, color)
if cycle_present == True:
print('As the graph is cyclic we can conclude that Schedule is NOT
CONFLICT SERIALIZABLE')
break
elif cycle_present == False:
print('Graph is ACYCLIC Hence the Schedule is CONFLICT SERIALIZABLE')
```

OUTPUT :



EXPERIMENT NO : 4

AIM : Simulation of ARIES recovery algorithm.

Theory:- ARIES stands for "Algorithm for Recovery and Isolation Exploiting Semantics." which is a recovery algorithm designed to work with a no-force, steal database approach.

ARIES uses a steal/no-force approach for writing, and it is based on three concepts:

1)Write-ahead logging
2)Repeating history during redo

ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state when the crash occurred. Transactions that were uncommitted at the time of the crash (active transactions) are undone.

Logging changes during undo It will prevent ARIES from repeating the completed undo operations if a failure occurs during recovery, which causes a restart of the recovery process.

Before describing this topic we need to elaborate some concepts:

Log sequence number

It refers to a pointer used to identify the log records.

Dirty page table

It refers to pages with their updated version placed in main memory and disk version of it is not updated.

A table is maintained which is useful in reducing unnecessary redo operation.

Fuzzy checkpoints.

A new type checkpoints i.e. fuzzy checkpoints has been derived that allowed to process new transactions alter the log has been updated without having to update the database.

The ARIES recovery procedure consists of three main steps:

Analysis

The analysis step identifies the dirty (updated) pages in the buffer (Note 6), and the set of transactions active at the time of the crash. The appropriate point in the log where the REDO operation should start is also determined

REDO

The REDO phase actually reapplies updates from the log to the database. Generally, the REDO operation is applied to only committed transactions. However, in ARIES, this is not the case. Certain information in the ARIES log will provide the start point for REDO, from which REDO operations are applied until the end of the log is reached. In addition, information stored by ARIES and in the data pages will allow ARIES to determine whether the operation to be redone has actually been applied to the database and hence need not be reapplied. Thus only the necessary REDO operations are applied during recovery.

UNDO

During the UNDO phase, the log is scanned backwards and the operations of transactions that were active at the time of the crash are undone in reverse order. The information needed for ARIES to accomplish its recovery procedure includes the log, the Transaction Table, and the Dirty Page Table. In addition, check pointing is used. These two tables are maintained by the transaction manager and written to the log during check pointing.

CODE:

```
num = 0
records = []
ttCheck = []
TT = {}
DPT = {}
pageLSN =
{} class LSN:
def __init__(self, lsn, trans, page,
action): self.lsn = lsn
self.trans = trans
self.page = page
self.action = action
if(action==""):
self.desc ="LSN "+str(lsn)+": T"+str(trans)+" updates P"+str(page)
elif(action=="commit"):
self.desc ="LSN "+str(lsn)+": T"+str(trans)+" commit"
elif(action=="end"):
self.desc ="LSN "+str(lsn)+": T"+str(trans)+" end"
elif(action=="disk"):
self.desc ="LSN "+str(lsn)+": disk P"+str(page)
class ttObj:
def __init__(self,lsn,p_lsn): self.lsn =
lsn self.p_lsn = p_lsn
def printTT():
global TT
print("------Transaction Table------")
print("T_ID\tLSN\tPrev LSN")
```

```
for i in TT:
print("T"+str(i)+"\t"+str(TT[i].lsn)+"\t"+str(TT[i].p_lsn)) def
printDPT():
global DPT
print("------Dirty Page Table------")
print("Page No.\tRecLSN")
for i in DPT:
print(str(i)+"\t\t"+str(DPT[i]))
def createRecords():
global num,records,pageLSN
num = 1
while(1):
print("************************************************************")
```

```
print('1.Update\t2.Commit\t3.Disk\t4.End Transaction\t5.System Crash')
opt=int(input())
if(opt==1):
trans = int(input("Enter transaction number\n"))
page = int(input("Enter page number\n"))
x = LSN(num,trans,page,"")
records.append(x) print(x.desc)
num+=1
if(page not in pageLSN.keys()):
pageLSN[page] = 0
elif(opt==2):
action = "commit"
trans = int(input("Enter transaction number\n"))
x = LSN(num,trans,0,action)
records.append(x)
print(x.desc)
num+=1
elif(opt==2):
action = "disk"
page = int(input("Enter page number\n"))
x = LSN(num,0,page,action)
records.append(x)
print(x.desc)
num+=1
elif(opt==4):
action = "end"
trans = int(input("Enter transaction number\n"))
x = LSN(num,trans,0,action)
records.append(x)
print(x.desc)
num+=1
elif(opt==5):
for i in range(0,len(records)):
print(records[i].desc)
print("\n") return
else:
print("Invalid option")
def phaseOne():
print("\n******PHASE 1 - ANALYSIS******")
global num,records,TT,ttCheck,DPT,pageLSN
for i in range(0,len(records)):
if(records[i].action==""): if(records[i].trans not
in ttCheck): ttCheck.append(records[i].trans) x
= ttObj(records[i].lsn,0)

TT[records[i].trans] = (x)
else:
```

```
TT[records[i].trans].p_lsn = TT[records[i].trans].lsn
TT[records[i].trans].lsn = records[i].lsn
if(records[i].page not in DPT.keys()):
DPT[records[i].page] = records[i].lsn
if(records[i].action=="commit"): if(records[i].trans in
ttCheck): del TT[records[i].trans]

if(records[i].action=="disk"):
pageLSN[records[i].page] = records[i].lsn
printTT()
printDPT()
def phaseTwo(): print("\n******PHASE 2 - REDO******")
global num,records,TT,ttCheck,DPT
for i in range(0,len(records)):
if(records[i].action==""):
if(records[i].page in DPT.keys()):
if(DPT[records[i].page] <= records[i].lsn):
if(pageLSN[records[i].page] < records[i].lsn):
print("REDO LSN "+str(records[i].lsn))
def phaseThree():
print("\n******PHASE 3 - UNDO******")
global num,records,TT,ttCheck,DPT
x = len(records)-1
for i in range(x,0,-1):
if(records[i].trans in TT.keys()):
temp = LSN(num,0,0,"undo")
temp.desc = "LSN "+str(num)+": CLR undo LSN"+str(records[i].lsn)
records.append(temp)
num+=1
for i in range(0,len(records)):
print(records[i].desc)
createRecords()
phaseOne()
phaseTwo()
phaseThree()
```

OUTPUT :

```
LSN 4: T3 updates P1
************************************************************
1.Update        2.Commit        3.Disk  4.End Transaction       5.System Crash
1
Enter transaction number
2
Enter page number
3
LSN 5: T2 updates P3
************************************************************
1.Update        2.Commit        3.Disk  4.End Transaction       5.System Crash
1
Enter transaction number
2
Enter page number
4
LSN 6: T2 updates P4
************************************************************
1.Update        2.Commit        3.Disk  4.End Transaction       5.System Crash
5
LSN 1: T1 updates P4
LSN 2: T1 commit
LSN 3: T3 updates P2
LSN 4: T3 updates P1
LSN 5: T2 updates P3
LSN 6: T2 updates P4



******PHASE 1 - ANALYSIS******
------Transaction Table------
T_ID    LSN     Prev LSN
T3      4       3
T2      6       5
------Dirty Page Table------
Page No.        RecLSN
4               1
2               3
1               4
3               5

******PHASE 2 - REDO******
```

```
.Update        2.Commit        3.Disk  4.End Transaction       5.System Crash

nter transaction number

nter page number

SN 1: T1 updates P4
************************************************************
.Update        2.Commit        3.Disk  4.End Transaction       5.System Crash

nter transaction number

SN 2: T1 commit
************************************************************
.Update        2.Commit        3.Disk  4.End Transaction       5.System Crash

nter transaction number

nter page number

SN 3: T3 updates P2
************************************************************
.Update        2.Commit        3.Disk  4.End Transaction       5.System Crash

nter transaction number

nter page number

SN 4: T3 updates P1
```

```
******PHASE 2 - REDO******
REDO LSN 1
REDO LSN 3
REDO LSN 4
REDO LSN 5
REDO LSN 6

******PHASE 3 - UNDO******
LSN 1: T1 updates P4
LSN 2: T1 commit
LSN 3: T3 updates P2
LSN 4: T3 updates P1
LSN 5: T2 updates P3
LSN 6: T2 updates P4
LSN 7: CLR undo LSN6
LSN 8: CLR undo LSN5
LSN 9: CLR undo LSN4
LSN 10: CLR undo LSN3
```

EXPERIMENT NO : 5

Aim: To implement distributed database.

Theory:

Fragmentation

This technique that is used to break up the database into logical unit called fragment, which may be assigned for storage as the various sites.

Types of partitioning

1. Horizontal

It is a subset of types in that relations. A subset of tuples that belong to the horizontal fragment are specified by condition on one or more attribute of relation. Often only one single attribute is involved.

2. Vertical

It is a divided relation vertically by columns. A vertical fragment on a relation keeps only contain attribute of the relation. We may want to fragment employee relation into vertical fragmentation. The first fragment includes personal name, birth date, address and gender and the second includes work.

3. Mixed

It is combination of horizontal fragmentation in order range participating, hash partitioning, composite partitioning. Each partition method has different advantage and design considerations.

Range partitioning maps data to the partitions based on ranges of partition. Key values that you establish for each partition is most common type or partitioning is often used with dates.

Hash partitioning maps data to partitions based on hashing algorithm that Oracle applies to a partitioning that you identify the hashing algorithm evenly distribute those among partitions, giving partitions approved the same size hash partitioning is the ideal method for distributing data evenly across devices.

List partitioning

It enables you to explicitly control rows map to partitions you do this by specifying a list of discrete values for the partitioning column in the description where you have to control the route to part mapping.

1)Partition By List: create
table sales_list68
(
salesman_id number(2),
salesman_name varchar(20),
sales_state varchar(20),
sales_amount number(10),
sales_date Date
)
partition by list(sales_state)
(
partition sales_west values('maharashtra'),
partition sales_north values('kashmir'),
partition sales_south values('kanyakumari','kerala')

);
SELECT * FROM sales_list68 PARTITION (sales_west) sales_state;
SALESMAN_ID SALESMAN_NAME SALES_STATE SALES_AMOUNT SALES_DATE
---------- ---------- ---------- ---------- ---------
1Tarun maharashtra 90000 12-JAN-01
2 Kinjal maharashtra 90001 12-JAN-03
SELECT * FROM sales_list68 PARTITION (sales_south) sales_state;
SALESMAN_ID SALESMAN_NAME SALES_STATE SALES_AMOUNT SALES_DATE
---------- ---------- ---------- ---------- ---------
4 sumil kanyakumari 50000 12-JAN-05
5 shreyans kerala 50000 12-JAN-08
2) Partition By
Range: create
table
sales_range68
(
salesman_id
number(2),
salesman_name
varchar(20),
sales_state
varchar(20),
sales_amount
number(10),
sales_date Date
)
PARTITION BY RANGE (sales_date)
(PARTITION sales_q1 VALUES LESS THAN (TO_DATE('12-JAN-2006','dd-MON-yyyy')), PARTITION sales_q2 VALUES LESS THAN (TO_DATE('12-JAN-2012','dd-MON-yyyy')));

SELECT * FROM sales_range68 PARTITION (sales_q1) sales_date;
SALESMAN_ID SALESMAN_NAME SALES_STATE SALES_AMOUNT SALES_DATE
---------- ---------- ---------- ---------- ---------
1Tarun Maharashtra 90000 12-JAN-01
2 Kinjal Maharashtra 90001 12-JAN-03
3 Sushmita Kashmir 80000 12-JAN-04
4 Sumil Kanyakumari 50000 12-JAN-05

SELECT * FROM sales_range68 PARTITION (sales_q2) sales_date;
SALESMAN_ID SALESMAN_NAME SALES_STATE SALES_AMOUNT SALES_DATE
---------- ---------- ---------- ---------- ---------
5 Shreyans Kerala 50000 12-JAN-08