Name: Gurdev singh

Id:100376466

LAB-8

Q-1 (a) Give an example of a schedule showing the lost update anomaly.

Ans: T4 has the value from T1 and T2. Let's assume that T4 takes the value from the two but in the mean time T 1 changes or updates T2 but T4 still uses the same value of T2.This represents the loss update anomaly as the values are updated for T2 but T4 still using the original value of T2.

(b) Give an example schedule to show that the lost update anomaly is possible with the read committed isolation level.

Ans:

So, we used read Committed isolation:

BEGIN

Transaction t1

Transaction T2

Transaction T4

COMMIT

Let's assume the lock is created on T4 to avoid the loss update anomaly. But due to this lock whole system will get locked and no values will be sent to each other transaction. Like T2 value depends on T1 But due to deadlock T2 will not get any transaction from T1 as a result T4 also will not get any value from T2. This whole created a deadlock problem.

Q-2 Consider a disk with block size B = 512 bytes. • A block pointer is P = 6 bytes long, and a record pointer is RP = 7 bytes long. • A file has r = 30,000 EMPLOYEE records of fixed length. • Each record has the following fields: Name (30 bytes), SSN (9 bytes), Department code (9 bytes), Address (40 bytes), Phone (10 bytes), Birthdate (8 bytes), Sex (1 byte), Job code (4 bytes), and Salary (4 bytes, real number).

A) **Calculate the record size R in bytes. (A record is composed of fields. A field holds information about an entity. For example, Name and SSN are two fields of an employee's record).**
   Ans:
   Total Record size R is the sum of all fields R=30+9+9+40+10+8+1+4+4
   = 115

B) **Calculate the blocking factor bfr and the number of file blocks b, assuming an un-spanned organization.**
   Ans:
   Blocking factor bfr = floor(B/R)
   = 512/115
   = 4 record/block

Number of blocks needed for file = ceiling(r/bfr) = ceiling(30000/4) = 7500

C) **Suppose that the file is ordered by the key field SSN and we want to construct a primary index on SSN. Calculate:**

I.The index blocking factor bfri (which is also the index fan-out fo);

Ans:

The total record size of SSN= (VSSN+RP)

=9+7

=16 Bytes

Index blocking factor bfri = floor (B/R)

=512/16

=32 records/block

Number of block required=ceil(r/bfr)

=30,000/32

=938 blocks

(ii) the number of first-level index entries and the number of first-level index blocks.

Ans:

Number of first-level index entries = number of file blocks b =30000/4

=7500

Number of first-level index blocks b1 = ceiling (r1 / bfr i)

= ceiling (7500/32)

= 234 blocks

(iii) the number of levels needed if we make it into a multilevel index:

ANS:

Number of second-level index entries r2 = number of first-level blocks b 1= 234 entries

Number of second-level index blocks b2= ceiling (r2 /bfr i)

= ceiling (234/32)

= 7 blocks

Number of third-level index entries r3 = number of second-level index blocks b2 = 7

Entries

Number of third-level index blocks b3 = ceiling (r3 /bfr i)

= ceiling (7/32)

=1

The third level has only one block, it is the top index level. So , the index 3 levels

IV) The total number of blocks required by the multilevel index

ANS:

Total number of blocks for the index bi = b 1 + b 2 + b 3 = 234 + 7 + 1 = 242 blocks

Q-3) Emp(eid: integer, ename: string, age: integer, salary: real) Works(eid: integer, did: integer, pct time: integer) Dept(did: integer, budget: real, managerid: integer)

ANS:

1. **Employees must make a minimum salary of $1000.**
   Ans:

```
CREATE TABLE Emp ( eid    INTEGER PRIMARY KEY,
ename  VARCHAR(255),
age    INTEGER,
salary REAL   CHECK  ( salary > 1000))
```

2. **Every manager must be also be an employee.**
   Ans:

```
CREATE TABLE Dept
(
  did INTEGER PRIMARY KEY,
 budget REAL,
 managerid INTEGER CHECK
          (  managerid=check_Manager(managerid)
   )
);
CREATE OR REPLACE FUNCTION check_Manager(mangerId INTEGER)
RETURNS  integer
AS $$
BEGIN
RETURN eid;
SELECT eid
FROM
emp
WHERE
emp.eid=mangerId ;
END; $$
LANGUAGE 'plpgsql';
```

3. **The total percentage of all appointments for an employee must be under 100%.**
   Ans:

```
CREATE TABLE Works
(eid     INTEGER REFERENCES emp(eid),
 did     INTEGER REFERENCES dept(did),
 pct_time INTEGER CHECK(appointments(eid)<100),
 PRIMARY KEY (eid, did)
 );
```

```
CREATE OR REPLACE FUNCTION appointments(empId INTEGER)
RETURNS  integer
AS $$
BEGIN
RETURN eid;
SELECT sum(pct_time)
FROM
Works
WHERE eid=empId
GROUP BY eid;
END; $$
LANGUAGE 'plpgsql';
```

4. **A manager must always have a higher salary than any employee that he or she manages.**
   ANS:

```
CREATE TABLE dept
(
  did INTEGER PRIMARY KEY,
 budget REAL CHECK
          (  budget>check_Employee_Salary(did)
   ),
           managerid INTEGER
);
--//THIS FUNCTION GIVES THE MAXIMUM SALARY OF AN EMPLOYEE WHO WORKS IN GIVEN
//DEPARRTMENT
CREATE OR REPLACE FUNCTION check_Employee_Salary(did INTEGER)
RETURNS  integer
AS $$
BEGIN
RETURN salary;
SELECT MAX(salary)
FROM
emp JOIN works
ON
emp.eid = works.eid
WHERE
works.did=did;
END; $$
LANGUAGE 'plpgsql';
```

5. **Whenever an employee is given a raise, the manager's salary must be increased by the same percentage.**
   ANS:

```
CREATE OR REPLACE FUNCTION
Manager_Salary_changes()
RETURNS trigger AS
$BODY$
```

```
BEGIN
IF NEW.salary <> OLD.salary THEN
        UPDATE Emp M
        SET    M.Salary = new.salary
        WHERE  M.salary <> new.salary
        AND    M.eid IN (SELECT D.mangerid
                FROM Emp E, Works W, Dept D
                WHERE E.eid = new.eid
                AND E.eid = W.eid
                AND W.did = D.did);
END IF;
RETURN NEW;
END;
$BODY$
--//CALLING THE TRIGGER FUNCTION:
CREATE TRIGGER salary_changes
AFTER UPDATE
ON emp
FOR EACH ROW
EXECUTE PROCEDURE
Manager_Salary_changes();
```