22-12-2021

ER-model :-

→ entities, attributes, relationships.

→ given by P.Chan in 1970's

Entities:-

→ Having physical (or) conceptual existence Ex:- student, course

→ The characteristics of entities describe its properties.

Properties for a student :- 1) Name
2) Roll No
3) Course etc.,

→ The othername of properties is 'attributes'

→ collection of entities are called 'entity sets' (or) 'entity types'

Ex:- (Sasi, --/12/2000, 420230, BTech) } Entity set.
(Arjun, --/12/2000, 420130, BTech)

Attributes / Properties:-

→ characteristics of attributes.

→ Types :-
- → Single valued
- → Multi-valued
- → Derived
- → Simple
- → Composite

Simple attribute:-

→ Value of this attribute is single / atomic / indivisible.

Ex:- Roll No of student, Course enrolled in

Composite attribute:-

→ There will be more than one component.

Ex:- Name → first name, middle name, last name.

Single valued attribute:-

→ It has only one value

Ex:- Aadhar Number.

Multi-valued attribute:-

→ It has multiple values.

Ex:- e-mail Id

Derived attribute:-

→ Value of this attribute is derivable from another attribute.

Ex:- Age.

Diagramatic notation:-

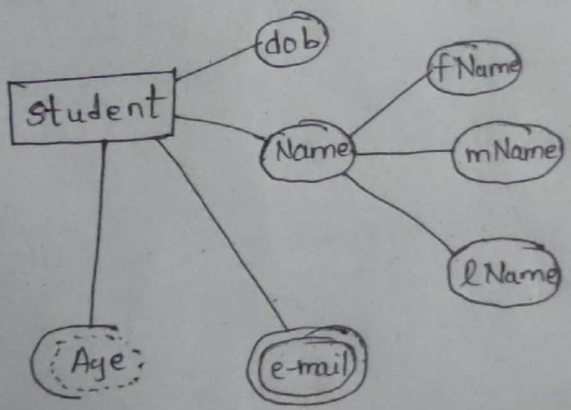
1) entity → rectangle.

2) attribute → ellipse connected to rectangle

3) Composite attribute → ellipse connected to ellipse.

4) Multivalued attribute → double ellipse.

5) Derived attribute → dashed ellipse.



Domain of Attributes:-

→ Domain is a set of values that an attribute can take

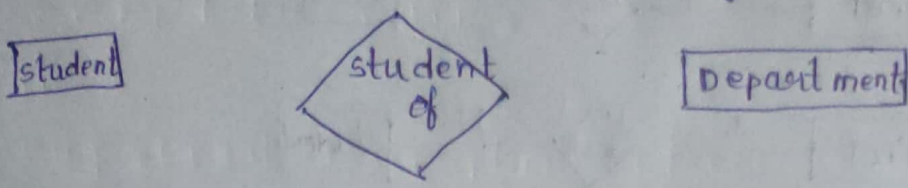
→ For composite attributes → domain is cross product of domain of components.

→ For multivalued attributes → set of subsets of values from basic domain

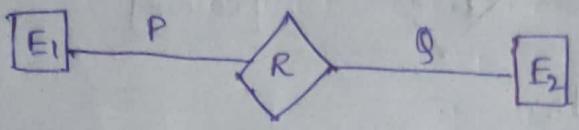
Relationship:-

→ An association between entities is called relationship.

→ Relationship is represented using diamond symbol.



Cardinality ratio:-



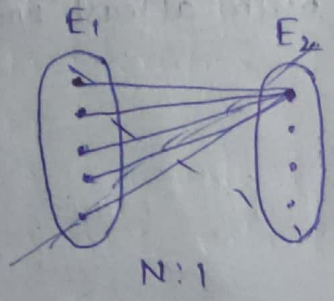
P = 1 (or) N

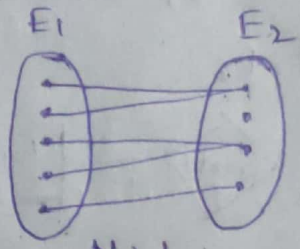
Q = 1 (or) M

Cardinality ratio:- P : Q

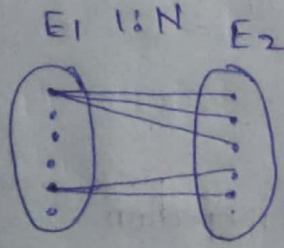
It can be 1.) 1 : 1 2.) 1 : M 3.) N : 1 4.) N : M

→ Maximum number of entities from E_2 than an entity from E_1 can possibly associate through R.

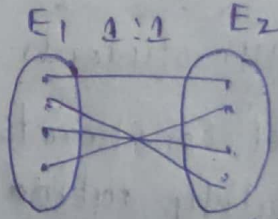




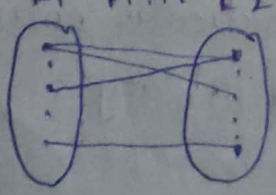
every entity in E_1 with atmost 1 entity E_2



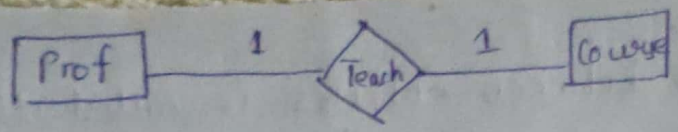
every entity in E_1 with atmost N entities in E_2



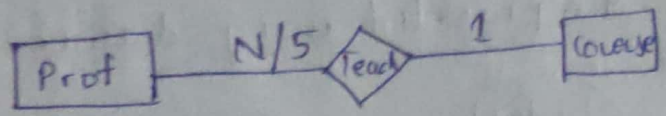
atmost 1 entity in E_1 with atmost 1 entity in E_2.



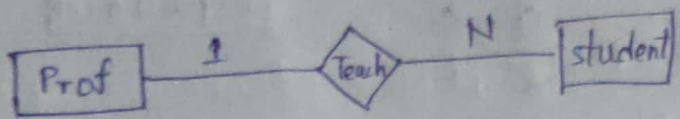
1:1



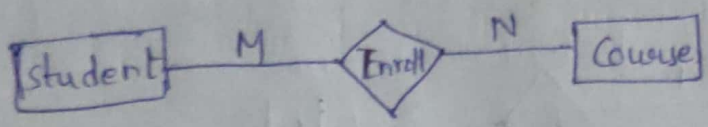
N:1



1:N



M:N

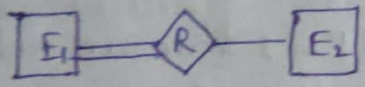


Participation Constraints:-

→ Total participation (There shouldn't be any unassociated entity)

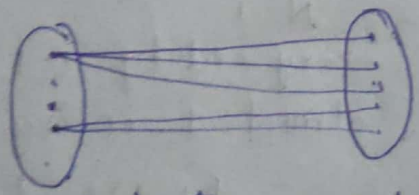
→ Partial participation

→ Total participation is represented by double line.



↓

All entities in E_1 should relate with atleast one entity of E_2



All students are related.

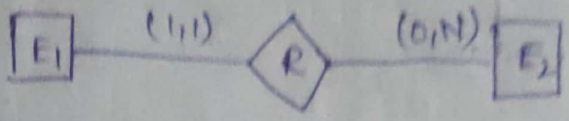
Structural Constraints:-

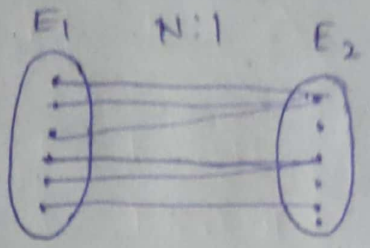
→ Cardinality ratio along with participation constraints is structural constraints.

Min-Max Notation:- (m,n)

→ m: Minimum number of times a particular entity must appear in the relationship tuples at any particular point of time.

→ n: Maximum number of times a particular entity must appear with the relationship tuples at any point of time.





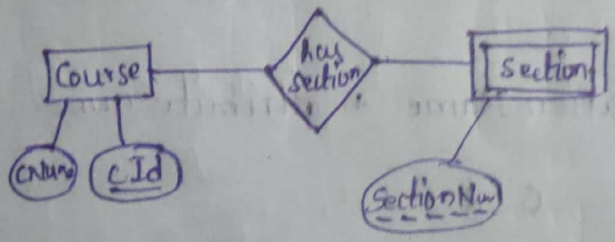
28-12-2021

Weak Entity Sets:-

→ These sets, depend for their existance on strong entity sets.

→ Weak entity is represented in double rectangle.

Here section is a weak entity.



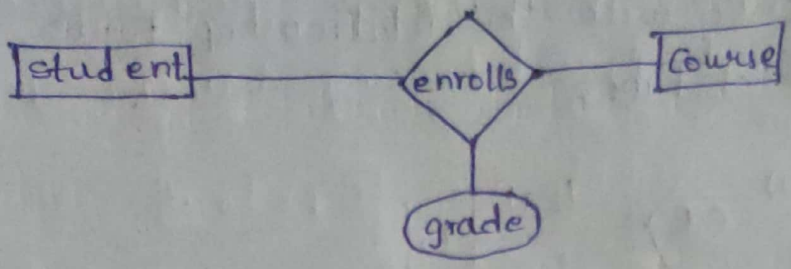
→ Weak entities have partial key i.e, ~~a~~ it can't serve as a complete key which helps in identifying a particular entity.

→ It should be used in combination with another key to get a particular identity.

Key - ——

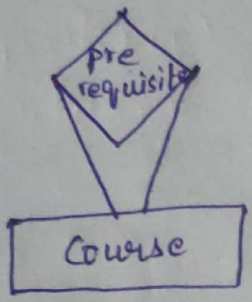
Partial key - - - - -

→ Relationships can also have attributes.



Recursive Relation:-

→ Relationships defined on the same entity.



Key:-

→ These are used to identify a particular entity.

ER model is a conceptual model of database.

Relational model is a logical model of database.

Relational model

Relation schema = Relation Name + attribute name

→

S.No	S.Name	CId
1	S_1	C_1
2	S_2	C_2
3	S_3	C_3

} Relation instance

→ One of the attribute (or) collection of attributes may act as a key.

→ A relation may have more than one key.

Consider the relation Book

Isbn No: Author Book Name Edition

Isbn No:- key

Author, Bookname, edition → key

Author, Bookname → key → Candidate key ↓ super key

→ One of the candidate keys is the primary key.

Minimal super key:-

→ The key whose subsets of it are not keys.

foreign key:-

→ A key is foreign key if it can take values of a key from another table in which it is a primary key.

Note that a table can have only one primary key.

(Referential integrity constraint)-

(29-12-2021) Relational Instance:- tuples of the relation

Database schema:-

→ collection of finite set of relations, schema along with integrity constraints.

Integrity constraints:-

1) Domain constraints

2) Key constraints:-

3) Referential integrity constraints:-

① values that an attribute can take is only from the declared domain.

② Its values can't be repeated for any other tuple since it should act as a key.

③ Foreign key shouldn't be violated.

Database Instance:-

→ collection of finite relational instances along with integrity constraints.

Relational Algebra:-

→ Operations to which operands are passed as relations and certain operations are performed finally another relation is obtained as output.

→ set of operators

→ arguments of operators are relations

→ Result of Relational algebra operations is also a relation.

Operators (σ , Π , $\times$, $\cup$, $\cap$, $-$, ∞)

↓ select, ↓ project, ↓ cross product, ↓ union, ↓ intersection, ↓ set diff, ↓ join

Select operator (σ):-

let the relational instance of relation R be

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

$\sigma(R)$

$A = 'a_1' \vee B = 'b_2'$

Result:-

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\sigma(R)$

$A = 'a_1' \wedge B = 'b_1'$

A	B	C
a_1	b_1	c_1

$R(C, D, E) \leftarrow \sigma(R)$

$A = 'a_1' \wedge B = 'b_1'$

C	D	E
a_1	b_1	c_1

Project operation (Π):-

$R_1(C, D) \leftarrow \Pi_{A,B}(R)$

Result:- R_1

C	D
a_1	b_1
a_2	b_2
a_3	b_3

Cross-Product operator ($\times$):-

R_1

a	b	c
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

R_2

d	e	f
d_1	e_1	f_1
d_2	e_2	f_2
d_3	e_3	f_3

$R_1 \times R_2$

a	b	c	d	e	f
a_1	b_1	c_1	d_1	e_1	f_1
a_2	b_2	c_2	d_1	e_1	f_1
a_3	b_3	c_3	d_1	e_1	f_1
a_1	b_1	c_1	d_2	e_2	f_2
a_2	b_2	c_2	d_2	e_2	f_2
a_3	b_3	c_3	d_2	e_2	f_2
a_1	b_1	c_1	d_3	e_3	f_3
a_2	b_2	c_2	d_3	e_3	f_3
a_3	b_3	c_3	d_3	e_3	f_3

Union:-

→ No: of attributes should be same.

→ domains of attributes are correspondingly equal.

$R_1 \cup R_2$

a	b	c
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
d_1	e_1	f_1
d_2	e_2	f_2
d_3	e_3	f_3

Intersection:-

→ The tuples common to both relational instances.

R_1

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2

A	B	C
a_2	b_2	c_2
a_3	b_3	c_3

$R_1 \cap R_2$

A	B	C
a_2	b_2	c_2

Set difference:-

→ Tuples that are in first operands but not in second relational instance.

$R_1 - R_2$

A	B	C
a_1	b_1	c_1

Join :- ⋈

R_1

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R_2

C	D	E
c_1	d_1	e_1
c_2	d_2	e_2

$R_1 \times R_2$

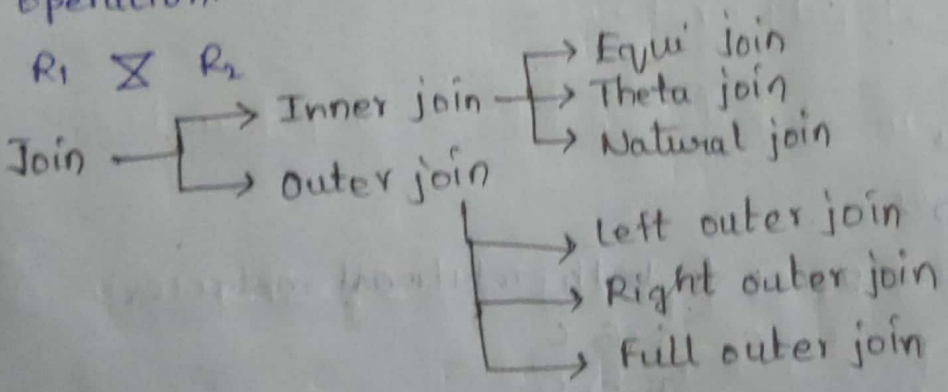
A	B	R_1.C	R_2.C	D	E
a_1	b_1	c_1	c_1	d_1	e_1
a_1	b_1	c_1	c_2	d_2	e_2
a_2	b_2	c_2	c_1	d_1	e_1
a_2	b_2	c_2	c_2	d_2	e_2

$\sigma (R_1 \times R_2)$

$R_1.C = R_2.C$

A	B	R_1.C	R_2.C	D	E
a_1	b_1	c_1	c_1	d_1	e_1

Cross-product followed by selection is called join operation.



30-12-2021

R_1

A	B	C
a_1	b_1	5
a_2	b_2	6

R_2

D	E	F
8	e_1	f_1
9	e_2	f_2

$R_1 \bowtie R_2$

$C < D$

A	B	C	D	F	F
a_1	b_1	5	8	e_1	f_1
a_2	b_2	6	9	e_2	f_2

a_1	b_1	5	9	e_1	f_1
a_2	b_2	6	8	e_2	f_2

→ This join is called theta join.

→ Because conditions that are provided are called as theta.

Equi Join:-

→ In equi join, the only comparision operator should be used is 'equals to'.

Ex:- $R_1 \bowtie R_2$
$C = D$

Natural Join:-

→ If the two participating relations have a common attributes, then output will be those instances where attribute values are equal.

$R_1 * R_2$

R_1

A	B	C
a_1	b_1	5
a_2	b_2	9

R_2

C	B	F
5	b_1	f_1
9	b_2	f_2

$R_1 * R_2$

A	R_1.B	R_1.C	R_2.C	R_2.B	F
a_2	b_2	9	9	b_2	f_2

Left outer join:-

$R_1 \bowtie R_2$
$R_1.C = R_2.D$

R_1

A	B	C
a_1	b_1	5
a_2	b_2	8

R_2

D	E	F
5	e_1	f_1
8	e_2	f_2

A	B	C	D	E	F
a_1	b_1	5	5	e_1	f_1
a_2	b_2	8	Null	Null	Null

→ Every tuple in the left hand side relation should be considered.

Right outer join:-

$R_1 \bowtie R_2$

$R_1.C = R_2.D$

R_1

A	B	C
a_1	b_1	5
a_2	b_2	9

R_2

D	E	F
5	e_1	f_1
8	e_2	f_2

A	B	C	D	E	F
a_1	b_1	5	5	e_1	f_1
-	-	-	8	e_2	f_2

Full outer join:-

$R_1 \bowtie R_2$

$R_1.C = R_2.D$

R_1

A	B	C
a_1	b_1	5
a_2	b_2	9

R_2

D	E	F
5	e_1	f_1
8	e_2	f_2

$R_1 \bowtie R_2$

A	B	C	D	E	F
a_1	b_1	5	5	e_1	f_1
a_2	b_2	9	-	-	-
-	-	-	8	e_2	f_2

Renaming :-

Professor

Name	eid

My Sql
Oracle

Rename 'name' as 'pname'.

$\rho_{pname}(\Pi_{name}(Professor))$

04-01-2022

Division :-

S(A,B); R(A,B,C,D) $\quad x \leftarrow r - s \quad$ X(C,D)

S

A	B
a_1	b_1
a_2	b_2

R

A	B	C	D
a_1	b_1	c_1	d_1
~~a_1~~	~~b_2~~	~~c_2~~	~~d_2~~
a_1	b_1	c_2	d_2
a_2	b_2	c_1	d_1
a_2	b_2	c_3	d_3
a_1	b_1	c_2	d_2

① $\Pi_{C,D}(r)$

C	D
c_1	d_1
c_2	d_2
c_3	d_3

c_2, d_2 is not in combination with all tuples of S in R.

$x \leftarrow r \div s$

C	D
c_1	d_1
c_3	d_3

Student | rollno | name | Enrollment | rollno | courseid |

Find those students who enrolled in all of the courses offered by CSE.

1) $\Pi_{rollno, courseid}$ (student * enrollment) → studEnrol

courses | courseid | dno | Department | dno | dname |

2) csCourse ← $\sigma_{dname = CSE}$ (course ⋈ department course dno = dept dno

3) studEnrol ÷ csCourse.

Aggregate functions:-

→ Avg
→ SUM
→ MIN
→ MAX
→ COUNT
→ These functions are denoted by $\mathcal{F}$ (which is subscript f)

EMPLOYEE | eno | ename | salary | dno |

$\mathcal{F}_{avg(salary)}$ EMPLOYEE → avg salary of employees.

$\mathcal{F}$
↙ grouping attributes
↘ aggregate functions.

$\mathcal{F}_{dno}$ $_{avg(salary)}$ (EMPLOYEE)

| dno | avg(salary) |

EMPLOYEE

dno, sex, avg(salary)

dno	sex	avg(salary

1) Convert specification into ER diagram

2) Convert ER diagram into specification.

05-01-2022

Student

S.No	S.Name	d.No	Mark
s_1	aaa	d_1	50
s_2	bbb	d_1	20
s_3	ccc	d_2	20

SQL

→ attributes/columns

select * from Student; → prints whole table

Keywords

select S.No from Student; → selects only S.No column.

select S.No as id from Student; → aliases

→ The SQL language allows duplicates

If the output shouldn't have duplicates, then use distinct keyword.

select distinct d.No as dept from Student;

dept
d_1
d_2

select count(*) from Student; → gives the count of no. of rows in the table.

select sum(Marks) from Student; → gives the sum of Marks.

select * from Student where Mark > 30;

select * from Student where Mark > 30 and d.No = d_2;

select * from Student where Mark > 10 and Mark < 30

select * from student where sName like '%va%'

↓

S.Name containing va

select * from Student where S.Name like 'va%'

↓

start with va

select * from Student where S.Name like '%va'

If there are two tables, student, Dept:-

the query, select * from student, dept gives the cross product of two tables.

select * from student, Dept where dNO = Dept.dnum

S.No	S.Name	d.NO	Mark	d.num	dname	phone
S_1	siva	d_1	10	d_1	dep 1	222
S_2	sivakumar	d_1	20	d_1	dep 1	222
S_3	Vanitha	d_2	30	d_2	dep 2	333
S_4	Rajesh	d_3	50	d_3	dep 3	444

select * from student, Dept where Mark = 10

S.No	S.Name	d.No	Mark	d.Num	d.Name	phone
S_1	Siva	d_1	10	d_1	dep1	222
S_1	siva	d_1	10	d_2	dep2	333
S_1	siva	d_1	10	d_3	dep3	444

select Student * from Student, Dept where Mark=10

only student attributes.

06-01-2021

Student | S.No | S.Name | Adv | Enrollment | S.No | C.No | Grade |

select SName from Student as S
who S.No in (select S.No from Enrollment as e
where student S.No = Enrollment.S.No)

Professor | Emp Id | Sex | Start Year |

select S.No, S.Name from Student as S
where S.Adv in (select EmpId from Professor where
sex = 'F')

In:- connects inner query to outer query

select S.No, S.Name from student as S
where S.Adv that in (select EmpId from Professor where
sex = 'F')

Get the yebou emp id, name of the senior most professor.

select EmpId, from Professor where
Start Year ≤ All (select start Year from Professor)

Student

S.No	S.Name	Adv	D.No
S_1	aaa	e_1	d_1 -
S_2	bbb	e_2	d_2 -
S_3	ccc	e_3	d_3 x

Professor

EmpId	Sex	Start Year
e_1	M	1982
e_2	M	1983

select S.No from Student where exists (select * from Professor where EmpId = adv)

Student

S.No	S.Name	Adv	dNo
s_1	aaa	e_1	d_1
s_2	bbb	e_2	d_2
s_3	ccc	e_3	d_1
s_4	ddd	e_1	d_2

select S.No from Student where Sadv = 'e_1'
UNION
select S.No from Student where Sadv = 'e_2'

→ for union, attribute name and data type of attr should be same.

select S.No from Student where dNO = 'd_1'
INTERSECTION
select S.No from Student where adv = 'e_1'

Output:- s_1

→ Attr and data type should be same.

select SNO from Student where dNO = 'd_1'
EXCEPT
select S.No from Student where adv = 'e_3'

18-01-2022

Joins:-

Professor (pname, deptno)

Department (dname, deptId)

select pname from Professor p where p.deptno = deptId from Department d and d.dname = "CSE"

select * from (Professor as P join Department as d on p.deptno = d.deptId) where d.dname = "CSE"

join = Inner join

Left outer join:-

→ All tuples from left side of join.

select * from (r_1 left outer join r_2 on $r_1.col1$ = $r_2.col1$) $r_1.col2 = 10$

Right outer join:-

select * from (r_1 right outer join r_2 on $r_1.col1$ = $r_2.col1$) where $r_1.col2 = 10$

Full outer join:-

select * from (r_1 full outer join r_2 on $r_1.col1$ = $r_2.col1$) where $r_1.col2 = 10$

Natural join:-

Aggregate functions:-

gateMarks (reg no, name, sex, branch, city, state, marks)

Find total number of students who took gate in CS and their avg marks.

```
select count(*) from gateMarks
where branch = "CS"
Select avg(marks) from gateMarks
where branch = "CS"

Select count(*) as No:of students,
       avg(Marks) as Average
from gateMarks where branch = "CS"
```

Get max, min, Avg marks of students whose city is Hyderabad.

```
select max(marks) as 'Max',
       min(marks) as Min,
       avg(marks) as Avg
from gateMarks where city = "Hyderabad'
```

Get the names of students who obtained maximum marks in branch of EC.

```
Select name from gateMarks where branch = "EC"
```

gateMarks (reg no, name, sex, branch, city, state, marks)

(r1, n1, m, b1, c1, s1, 90)

(r2, n2, m, b2, c1, s1, 60)

(r3, n3, f, b2, c2, s2, 70)

(r4, n4, m, b2, c1, s2, 60)

select branch, max(marks) from gateMarks group by branch.

output: (b1, 90), (b2, 70)

select max(marks) from gateMarks groupby state, sex

24-01-2022

select @@sql_

SET sql_mode=(select CONCAT(@sql_mode, 'ONLY_FULL_GROUP_BY')

1) Query: report the total enrollment in each course in the even semester of 2014, include only courses with a minimum enrollment of 10.

Enrollment (rollno, courseId, sem, year)

select count(*) from Enrollment

select * from Enrollment where sem = 2 and year = 2014.

select count(*) from Enrollment where sem = 2 and year = 2014.

select courseId, count(rollno) from Enrollment where sem=2 and year= 2014 group by courseId

Select courseId, count(rollNo) from Enrollment where sem = 2 and year = 2014 group by courseId having count(rollNo) >= 10

'having' can be applied on aggregate fun and group by.

Views-

select sname, dname from student s, department d where s.dno = d.dno

→ Views are virtual relations not physically stored and are executed on demand.

create view stud-dpt as (select sname, dname from student s, department d where s.dno = d.dno)

→ We can treat stud-dpt as some table.

Restrictions on insertions of views:-

→ Views involving more than one table can't be inserted.

→ Views not involving primary key of the table

→ Views involving group by clause in the select query.

create view V1 as (select sname from student)

delete from student where rollno = 'r2'

update student

set sname = 'aaaa', dno = 3

where snp = 'r1'

order by sname desc

25-01-2022

Functional Dependencies:-

student (studName, rollNo, sex, deptNo, dname, hod)

Correctness of the schema:-

$(s_1, r_1, m, d_1, dn_1, h_1), (s_2, r_2, m, d_1, dn_1, h_1)$

→ For students having same dept No; dept Name and hod will be same which is repetition of data.

(null, null, null, d_2, dn_2, h_2)

→ This is like d_2 has no students enrolled in it. → Insertion anamoly

→ Suppose hod of one of dept has changed. Then all values with that particular dept have to be updated with new hod which is error prone. → update anamoly, deletion anamoly

→ So, update the table with only the following.

Student (studName, rollno, sex, dno)

Department (dno, dname, hod)

Consider the relation R(X, Y, Z); Y

$X \rightarrow Y$: This is a functional dependency

t_1, t_2 are any two tuples of the relation

If $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$

→ If this holds for any t_1, t_2, then we say $X \rightarrow Y$

the relation, $R(X, Y_1, Y_2, Y_3)$

f X is a key then $X \to Y_1$, $X \to Y_2$, $X \to Y_3$

rivial Functional dependencies:-

$X \to Y$

X = set of attributes, Y = set of attributes.

If $Y \subseteq X$, then $X \to Y$ is called as Trivial functional dependency.

If $X \cap Y = \phi$, then $X \to Y$ is completely non-trivial Functional dependency.

→ Using the concept of functional dependencies, Normal forms 1NF, 2NF, 3NF, BCNF, 4NF, 5NF

1-01-2022

et R(X,Y,Z) be a relation

r be a relation instance such that $X \to Y$

$r : X \to Y$

(x_1, y_1, z_1) t_1

(x_2, y_2, z_2) t_2

(x_1, y_1, z_3) t_3

$: X \to Y$ if $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$

onsider t_1, t_3

$t_1[X] = x_1 = t_3[X]$

eck for $t_1[Y]$, $t_3[Y]$ $t_1[Y] = y_1 = t_3[Y]$

r satisfies the functional dependency $X \to Y$

If r contains the following tuples:-

$t_1 (x_1, y_1, z_1)$

$t_2 (x_2, y_2, z_2)$

$t_3 (x_1, y_1, z_3)$

$t_4 (x_1, y_2, z_4)$

Here $t_1[x] = t_3[x] = t_4[x] = x_1$

but $t_1[y] = t_3[y] \neq t_4[y]$

So, r doesn't satisfy functional dependency.

If relation $R[X, Y, Z, A]$ with X as a key, then $X \to Y$, $X \to Z$, $X \to A$ holds

Deriving new Functional Dependencies:-

→ Given that a set of FD's F holds on R we can infer that a certain new FD must also hold on R.

→ Ex:- given that $X \to Y$, $Y \to Z$ hold on R then $X \to Z$ must also hold.

→ Unless all FD's are known, a Relation schema is not fully specified.

Entailment Relation:-

→ FD's $F \models \{X \to Y\}$ [F entails $X \to Y$]

if in every instance r of R on which FD's F hold, FD $X \to Y$ also holds.

Define $F^+ = \{X \to Y \mid F \models X \to Y\}$

F^+ is called closure of F

Armstrong's Inference Rules

1.) Reflexive rule

$F \models \{X \to Y \mid Y \subseteq X\} \; \forall \; X$

→ This is called trivial F.D

2.) Augmentation rule

$\{X \to Y\} \models \{XZ \to YZ\}, Z \subseteq R \quad YZ = X \cup Z$

3.) Transitive rule

$\{X \to Y, Y \to Z\} \models \{X \to Z\}$

4) Decomposition or Projective rule

$\{X \to YZ\} \models \{X \to Y\}, \{X \to YZ\} \models \{X \to Z\}$

5.) Union (or) Additive rule

$\{X \to Y, X \to Z\} \models \{X \to YZ\}$

6.) Pseudo transitive rule

$\{X \to Y, WY \to Z\} \models \{WX \to Z\}$

Derive Rule 5 from 1, 2, 3

$X \to Y, \quad X \to Z$

$X \to Y \Rightarrow X \to YX$ (Augmentation) ↳ ①

$X \to Z \Rightarrow XY \to YZ$ (Augmentation ↳ ②

① & ② $\Rightarrow X \to YZ$ (Transitive)

Rule 4:-

$\{X \to YZ\} \models \{X \to Y, X \to Z\}$

$X \to YZ \Rightarrow XY \to YZ \Rightarrow X \to Z$ (Augmentation)

$X \to YZ \Rightarrow XZ \to YZ \Rightarrow X \to Y$ (Augmentation)

Rule -5:-

$\{X \to Y, WY \to Z\} \models \{WX \to Z\}$

$X \to Y \Rightarrow WX \to WY$ (Augmentation)

WY ↳ (1)

$WY \to Z \to (2)$

(1) & (2) $\Rightarrow WX \to Z$ (transitive)

→ Rules 1, 2, 3 are sufficient.

Sound and complete Inference rules:-

Sound:-

→ Every new FD $X \to Y$ derived from a given set of FD's F using Armstrong's Axioms is such that $F \models \{X \to Y\}$

Completeness:-

→ Any FD $X \to Y$ logically implied by F can be derived by F using Armstrong's Axioms.

Proving soundness (01-02-2022)

Suppose $X \to Y$ is derived from F using AA in some n steps. If each step is correct then overall deduction would be correct

2-02-2022

Computing X^+

R = (rollNo, name, advisorId, advisorName, courseId, grade)

F = {rollNo → name, rollNo → advisorId, advisorId → advisorName; rollNo, courseId → grade}

$\{rollNo\}^+ = \{name, advisorId, advisorName, rollNo\}$

$\{advisorName\}^+ = \{advisorName\}$

$\{rollNo, courseId\}^+ = \{rollNo, name, advisorId, courseId, advisorName, grade\}$

$\{rollNo, courseId, advisorId\}^+ = \{rollNo, name, advisorId, courseId, advisorName, grade\}$

→ rollNo, courseId determines every attribute in the scherma.

→ {rollNo, courseId} is the key of R.

Normal Form

2NF:-

A FD X → A for which there is no proper subset Y of X such that Y → A is called full functional dependency. A is said to be fully functional dependent on X.

1NF:- attributes can take only atomic values. (like only one value).

2NF:-

A relation schema R is in 2NF if every non-prime attribute is fully dependent

Ex:-

Book (authorName, title, authoAffiliation, ISBN, publisher, pubYear)

Keys:- (authorName, title), ISBN

We also have authorName → authorAffiliation

(AuthorAffiliation is not fully functional dependent on the first key)

Student (rollNo, name, dept, sex, hostelName, roomNo, admitYear)

Keys:- rollNo, hostelName → roomNo)

Also have hostelName → sex

→ So, this is not in 2NF

3NF:- 07-02-2022

→ Relation schema R is in 3NF if it it is in 2NF and no non prime attribute of R is transitively dependent on any key of R.

Student (rollNo, name, dept,

Another definition of 3NF:-

→ Relation schema R is in 3NF if for any non-trivial FD $X \rightarrow A$ either (i) X is superkey or (ii) A is prime.

Suppose some R violates the above definition.

BCNF:-

gradeInfo (rollNo, studName, course, grade)

Suppose:-

1) rollNo, course → grade

2) studName, course → grade

3) rollNo → studName

4) studName → rollNo

keys:- (rollNo, course), (studName, course)

For 1,2 LHS is key. for 3,4 RHS is prime attribute.

So, gradeInfo is in 3NF.

But studName is stored redundantly along with every course being done by the student.

Boyce-Codd Normal form:-

→ R is in BCNF if for every non-trivial FD $X \rightarrow A$, X is a superkey of R.

In gradeInfo, F.D's 3, 4 are nontrivial LHS is not a superkey. So, gradeInfo is not in BCNF.

Decompose:- gradeInfo (rollNo, course, grade)
studInfo (rollNo, studName)

→ Redundancy allowed by 3NF is disallowed by BCNF.

→ BCNF is stricter than 3NF.

→ 3NF is stricter than 2NF.

Transactions:

→ sequence of queries performed on DB.

→ They a property: ACID → Durability
→ Isolation
→ Atomicity
→ Consistency

Atomicity:-

→ Either all instructions in the transaction should execute or not.

→ There are no intermediate states.

Consistency:-

→ The changes should be consistent.

Durability:-

→ changes made should last forever.

→ The interlinking of two or more transactions is called concurrent execution.

→ A transaction can be in any of the following states:-

1) active → during execution
2) partially committed → like computations completed, only write pending
3) failed → incomplete
4) Abort → terminated by the system
5) commit
→ finished

Active → partially committed → committed

→ failed → Abort

Schedule:

→ A plan for concurrent execution.

let

T_1	T_2
read(A)	
A = A - 50	
write(A)	
read(A)	
B = B + 50	
write(B)	
commit	
	read(A)
	temp = A * 0.1 → A = A - temp
	write(A)
	read(B)
	B = B + temp
	write(B)
	commit

→ The above is serial schedule (op's not interlinked)

→ This is like plan / schedule.

read(A)	
A = A - 50	
write(A)	
	read(A)
	temp = A * 0.1
	A = A - temp
	write(A)

read(B)

B = B+50

write(B)

commit

read(B)

B = B+temp

write(B)

commit

→ This is non-serial schedule

T_1	T_2
read(A)	
A = A-50	
	read(A)
	temp = A * 0.1
	A = A - temp
	write(A)
	read(B)
write(A)	
read(B)	
B = B+50	
write(B)	
commit	
	B = B+temp
	write(B)
	commit

→ Inconsistent schedule.

08-03-2022

1) I_i = read(Q) I_j = write(Q)

2) I_i = write(Q) I_j = read(Q)

3) ~~for~~ I_i = write(Q), I_j = write(Q)

} I_i is in conflict

4) I_i = read(Q) I_j = read(Q) => not in conflict.

S_1

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

S_2

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

S_2 is serial
→ so, S_1 is conflict serializable schedule

S_1 is equivalent to S_2

r(A) r(A)
r(B) w(A)
w(A) r(B)
w(B) w(B)

R(A) R(A)
W(A)
R(B)

serial schedule
No conflict

→ Concurrency control module generates schedules such that they don't produce any anomalies.

Check whether the following schedule is conflict serializable:-

S_1: $r_2(A), w_2(A), r_1(A), w_1(A), r_2(B), w_2(B)$

T_1	T_2
	$r_2(A)$
	$w_2(A)$
$r_1(A)$	
$w_1(A)$	
	$r_2(B)$
	$w_2(B)$

$\Rightarrow$

	$r_2(B)$
$r_1(A)$	
	$w_2(B)$
$w_1(A)$	

$\Rightarrow$

	$r_2(B)$
	$w_2(B)$
$r_1(A)$	
$w_1(A)$	

$\therefore$ S_2: $r_2(A), w_2(A), r_2(B), w_2(B), r_1(A), w_1(A)$

S_1 and S_2 are conflict equivalent and S_2 is serial schedule

$\therefore$ S_1 is conflict serializable.

$\Rightarrow$

$r_1(A)$	
	$r_2(B)$
	$w_2(B)$
$w_1(A)$	

$\Rightarrow$

$r_1(A)$	
	$r_2(B)$
	$w_2(B)$
	$w_1(A)$

→ There is a simple way to check whether a schedule is conflict serializable using a graph.

→ The graph we construct is precedence graph.

→ If the graph has a cycle then it is not conflict serializable.

→ Nodes in the graph are transactions.

→ Edges are determined by overlapping op^ns.

T_1 T_2

w(A)

r(A)

09/03/2022

Precedence graph:-

→ If an operation T_i is followed by an operation of T_j and one of the two operations is a write then add edge from T_i to T_j.

$G = (V, E)$

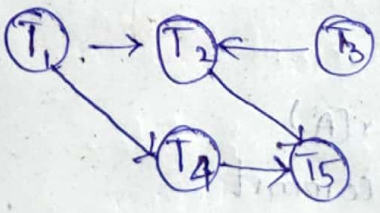
$V \leftarrow Tr^{n}s$

$E \rightarrow$

S: $w_1(A), r_2(A), w_1(B), w_3(C), r_2(C), r_4(B), w_2(D), w_4(E), r_5(D), w_5(E)$

$T_1 \rightarrow T_2$

S: T_1 T_4 T_3 T_2 T_5



→ As there is no cycle in precedence graph

→ The graph is conflict serializable.

→ To get the serial schedule, apply topographical sort for the directed graph.

→ Suppose the two transactions T_i, T_j

T_i: writes an item A : first

T_j: read item A : second

If commit of T_i proceeds before commit of T_j, then that schedule is called recoverable schedule.

Ex:-

T_1	T_2
read(A)	
write(A)	
	read(A)
	~~commit~~
Abort	
commit	
	commit

→ This is a recoverable schedule.

If the schedule is in this form;

T_1	T_2
r(A)	
w(A)	
	r(A)
	commit
commit	

→ This is non-recoverable schedule.

If there is a role back in T_1 after commiting T_2, T_2 can't be rolled back

T_1	T_2	T_3
r(A)		
r(B)		
w(A)		
	r(A)	
	w(A)	
		r(A)

→ If T_1 aborts after r(A), in T_3, then all the three transactions roll back because of the recovarable schedule condition.

→ This is called as cascade.

→ If r(A) of T_2 is after commit of T_1 and if r(A) of T_3 is after commit of T_2 then schedule is cascade less schedule.

T_1	T_2
r(A)	
A = A - 1000	
w(A)	
	r(A)
	A = A + 500
	w(A)
	commit
commit	

→ This is non-recoverable schedule.

T_1	T_2
r(A)	
A = A-1000	
w(A)	
	r(A)
	A = A+500
	w(A)
commit	
	commit

→ A recoverable schedule.

→ It is not cascadeless as read(A) of T_2 is happening before commit of T_1

T_1	T_2
r(A)	
A = A-1000	
w(A)	
commit	
	r(A)
	A = A+500
	w(A)
	commit

→ Recoverable and cascadeless schedule.

10-03-2022

View serializability:-

→ If in schedule S, T_i reads the initial value of Q then in S' also, T_i must read initial value of Q.

→ If in schedule S', T_i executes read(Q) and that value was produced by T_j (if any) then in schedule S' also T_i must read the value of Q that was produced by the same write(Q) operation of T_j.

→ The transaction (if any) that performs the final write(Q) operation in S must also perform the final write(Q) operation S'.

IS

S_1

T_1	T_2	T_3
r(Q)		
	w(Q)	
w(Q)		
		w(Q)

S_2

T_1	T_2	T_3
r(Q)		
w(Q)		
	w(Q)	
		w(Q)

→ In both schedules, the above conditions hold

→ So, S_1 and S_2' are view equivalent

→ As S_2' is serial schedule, S_1 & S_2 are view serializable.

→ View serializability is a restricted form of conflict serializability.

→ If a write to an operand is performed directly without a read, then that write is called blind write.

→ If a schedule is view serializable, is not conflict serializable, then it should have atleast one blind write.

Ex:-

S: $R_2(B), R_2(A), R_1(A), R_3(A), W_1(B), W_2(B), W_3(B)$

B: T_2

T_3

Choose the schedule with starting as T_2 and ending with T_3

Our required serial schedule is $T_2\ T_1\ T_3$

Now, check for A

A: T_2

Here also $r(A)$ is done by T_2

So, $T_2\ T_1\ T_3$ is serial schedule and satisfies all points.

So, given schedule S is view serializable.

Transaction control language.

- Delete from T_1 where $T_1.c = 'a'$
  Rollback
- Save point SP_1
  - Delete from customer where Id = 1
- Savepoint SP_2
  Delete f

savepoint sf1;
Delete from customers where id 3;
Rollback to sf1;

15-03-2022

Concurrency (Module) Control:

→ Concurrency is achieved by locking
→ Two types:
(i) shared lock (lock-S) → read
(ii) Exclusive lock (lock-X) → write

lock-S(A)	lock-X(A)
read(A)	write(A)
unlock(A)	unlock-X(A)

→ If lock held is shared lock, then other transaction can read that element / can get that lock
→ If lock is exclusive lock, then other tr can't get that lock.

Lock compatibility Matrix:

	S	X
S	True	False
X	False	False

→ These locks don't ensure consistency. They provide isolation.

Ex:-

T_1	=>	T_2
		lock-X(B)
read(B)		read(B)
B = B-50		B = B-50
write(B)		write(B)
r(A)		unlock(A)

A = A+50
w(A)

A = A+50
write(A)
unlock(A)

→ Converting a shared lock to exclusive lock is lock upgrading.

→ Converting exclusive lock to shared lock is lock downgrading.

T_2
r(A)
r(B)
display(A+B)

⇒

T_2
lock-S(A)
r(A)
unlock(A)
lock-S(B)
r(B)
unlock(B)
display(A+B)

→ Let the transactions interleave this way;

T_1
lock-X(B)
read(B)
B = B-50
write(B)
unlock(B)

T_2
lock-S(A)
read(A)
unlock(A)
lock-S(A)
read(B)
unlock(B)
display(A+B)

lock-X (A)
read (A)
A = A+50
write (A)
unlock (A)

→ Here, when A = 100, B = 200, the exp res is 300, but, the actual res is 250

→ So, lock solemnly doesn't ensure consistency.

Two-phase locking protocol (2PL)

→ It says that all the locks should be attained and then only unlocking should start.

T_1

lock-X (B)
read (B)
B = B-50
write (B)
lock-X (A)
read (A)
A = A+50
write (A)
unlock (B)
unlock (A)

Growing phase — No unlocking

shrinking phase — No locking even after unlocking

T_2

lock-S (A)
read (A)
lock-S (B)
read (B)
unlock (A)
unlock (B)
display (A+B)

17-03-2022

Read(D):

If T_i has a lock on D

then

read(D)

else

begin

if necessary, wait until no other transaction has a lock-X on D

grant T_i a lock-S on D

read(D)

end

Write(D):

If T_i has a lock-X on D

then

write(D)

else begin

if necessary wait until no other transaction has any lock on D

If T_i has a lock-S on D

then

upgrade lock on D to lock-X

else

grant a lock-X on D

write(D)

end

After commit(or) Abort unlock

→ The point of time at which last lock was obtained by the transaction is called lock-point of transaction.

Theorem:
Two phase locking produces only conflict serializable schedule.

Proof:-
Assume $T_0, T_1, T_2, \ldots, T_{n-1}$ be transactions in the schedule.

let us assume there is a cycle in precedence graph involving all trⁿs.

If there is an $T_i \rightarrow T_j$ in graph, then

$\alpha_i < \alpha_j$

So, if $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow \ldots \rightarrow T_{n-1} \rightarrow T_0$

$\Rightarrow \alpha_0 < \alpha_1 < \ldots < \alpha_{n-1} < \alpha_0$

$\Rightarrow \alpha_0 < \alpha_0$

which is contradiction.

So, there will not be any conflict.
Hence two phase locking gives only conflict serializable schedule.

Deadlock:-

T_1	T_2
lock-X(B)	
r(B)	
B = B+50	
w(B)	
	lock-S(A)
	r(A)
	lock-S(B)

lock-X(A)

→ T_1 waits for lock on A

T_2 waits for lock on B

→ So, a deadlock

→ We should one of the tr^n to deal with deadlock.

T_1	T_2	T_3
l-X(A)		
r(A)		
l-S(B)		
r(B)		
w(A)		
unlock(A)		
	lock-X(A)	
	r(A)	
	w(A)	
	u-l(A)	
		l-S(A)
		r(A)

→ If there is a deadlock in above schedule then if we decide to rollback T_1, then T_2 & T_3 should also be rolled back due to cascading.

→ A transaction, if it is waiting for the lock for long time, it is called starvation.

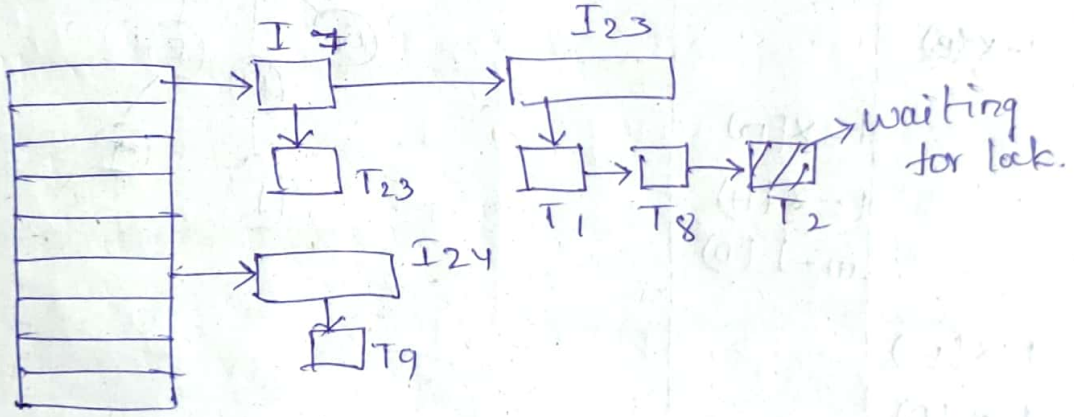
→ A transaction if it holds all its exclusive locks till it commits/aborts, then cascading rollback can be avoided. "strict 2 phase"

→ A transaction if it holds all the locks till it commits or aborts.

→ This is rigorous 2 phase locking

→ The d.s used for implementing locks is lock-table.

Lock Table:-



22-03-2022

Graph Based Protocol

1) only exclusive locks are allowed.

2) The first lock by T_i may be on any data item subsequently a data Q can be locked by T_i only if the parent of Q is currently locked by T_i

3.) Data items may be unlocked at any time.

4) A data item that has been locked and unlocked by T_i cannot be subsequently be locked by T_i

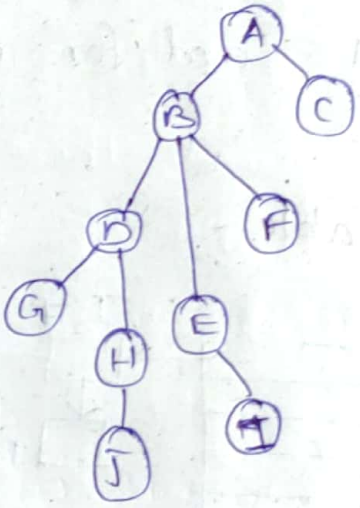
Ex:-

T_0: lock-x(B), lock-x(E), lock-x(D), unlock(B), unlock(E), lock-x(G), unlock(D), unlock(G)

T_1: l-x(D), l-x(H), unlock(D), unlock(H)

T_2: l-x(B), l-x(E), unlock(E), unlock(B)

T_3: l-x(D), l-x(H), unlock(D), unlock(H)



T_0	T_1	T_2	T_3
l-x(B)			
	l-x(D)		
	l-x(H)		
	un-l(D)		
l-x(E)			
l-x(D)			
un-l(B)			
un-l(E)			
		lock-x(B)	
		l-x(E)	
	unlock(H)		
l-x(G)			
un l(D)			
			l-x(D), l-x(H)
			unlock(D)
			unlock(H)
			unlock(
		u-l(E)	
		u-l(B)	
u-l(G)			

→ It ensures that the schedule is free from deadlock but doesn't gurantee recoverability and cascade freedom.

Commit Dependency:-

→ Q is a data item with uncommitted write.

→ Record which transaction performed last write to Q (T_j)

→ T_i reads Q

→ T_i is not allowed to commit until T_j commits.

Deadlock Handling:-

1.) Deadlock Prevention

2.) Deadlock Detection

Deadlock Prevention:-

→ acquire all locks before starting the transaction.

→ But concurrency suffers.

→ Impose partial ordering for all data items before it begins execution.

→ Require that a transaction can lock data items only in order specified by the partial order.

Timeout schemes:-

→ If any lock is not obtained, then wait for 100ms and retry the lock for acquiring.

23-03-2022

Deadlock Prevention:-

- wait - die scheme
- older transactions wait for younger transactions to release data item.

  Younger transactions never wait for older ones, they are rolled back.

  Old — wait

  Young - die.

- wound - wait scheme
- older tr^n^s forces roll back of younger tr^n^s instead of waiting for it.

  Younger transactions wait for older ones.

  wound - older tr^n^s (kills younger tr^n^s)

  wait - younger tr^n^s (wait for older to release)

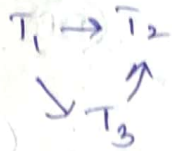
→ The above schemes may result in starvation.

→ To avoid starvation, make sure that oldest tr^n^ is not rolled back.

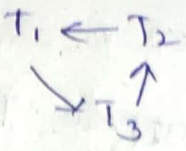
Deadlock Detection:-

→ If T_i waits for T_j to release a lock, then draw an edge from T_i to T_j

→ If there is a cycle in the graph, then deadlock has occured. $T_i \rightarrow T_j$



No deadlock
No cycle



Deadlock
(has cycles)

Deadlock Recovery:-

→ kill one of the transaction causing the deadlock

→ This is the only way to recover from deadlock.

→ Two types of rollback:-

1) Total Rollback:- Blindly rollback entire trⁿ

2) Partial Rollback:- Rollback till certain instrⁿ.

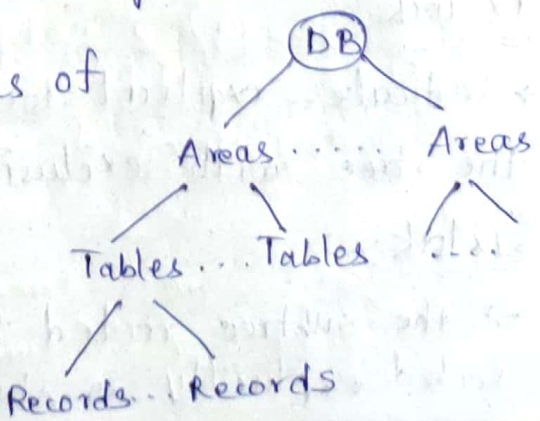
Multiple Granularity:-

→ If we have to update all records in the table to a single value, then instead of obtaining a lock for each record, acquire a single lock for the table.

→ If we consider only few records, obtain a lock for each record.

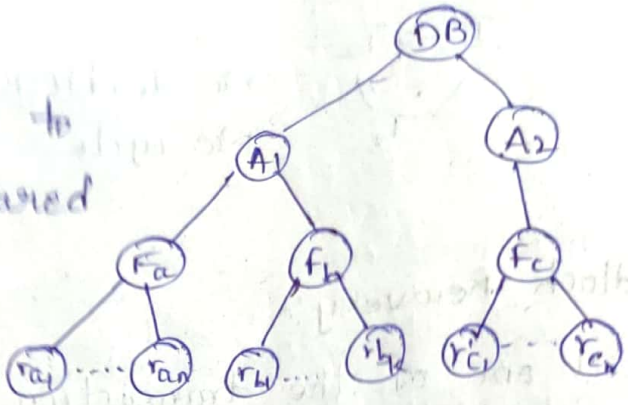
→ Obtaining locks at various levels of db is called multiple granularity.

→ These are various levels of the database.



Ex:-

Suppose T_1 wants to read ra_2 and shared lock is supplied.



At that time, T_2 wants to write to db, but ex-lock is not acquired due to shared lock on ra_2

Also, it should traverse across all nodes to check if x-lock can be granted

→ This problem is due to multiple granularity.

→ There are locks called intention locks

1.) IS lock
(Intention shared lock)

2.) IX lock
(Intention exclusive lock)

3.) SIX lock (Intention shared-exclusive lock)

IS lock:-

→ Indicates explicit locking at a lower level of the tree but with shared locks.

IX lock

→ Indicates explicit locking at a lower level of the tree with exclusive locks / shared locks.

SIX lock:-

→ The subtree rooted by the present node is locked explicitly in shared mode and explicit

locking is done at a lower level with X-locks.

24-03-2022

Intention lock compatibility matrix:-

	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

Rules for T_i to lock a node Q:

1) The lock compatibility matrix should be considered.

2) The root of tree must be locked first and may be locked in any mode.

3) A node Q can be locked by T_i in S (or) IS mode, the parent of Q is currently locked by T_i in either IX (or) IS mode.

4) A node Q can be locked by T_i in X, SIX, IX mode only if the parent of Q is currently locked by T_i in either IX (or) SIX mode.

5) No unlocking

6) All the locks on its children must be released before releasing locks on particular node.

Index locking

select count(*) from ins where dept = Physics
↳ req shared
insert into ins values
(a, b, physics, c) → req X

→ But there is no conflict blw them.
→ This is Phanton phenomenon.
→ This problem can be solved by acquiring a lock on data item dept_name.
→ Or use index locking.

Time stamp based protocol

→ Two timestamps for any item
1.) Read timestamp → largest T.S of T_i which made read
2.) write time stamp
↳ largest T.S among tr's that made write (Q)

30-03-2022

→ Time stamp is a counter.
→ Every transaction is assigned a time stamp when it arrives in the system.
→ If T_i arrives before T_j, then

$$TS(T_i) < TS(T_j)$$

→ For a data item, it is provided with two time stamps.
write - w - timestamp (Q) - W-TS(Q)
read - r - timestamp (Q) - r-TS(Q)

where r-TS(Q) = Time stamp of latest trⁿ which successfully read Q

w-TS(Q) = Time stamp of latest trⁿ which successfully wrote Q.

→ Timestamp order = serializability order.

Read(Q):-

→ T_i wants to read Q.

→ Time stamp of T_i is $TS(T_i)$

if $TS(T_i) \leq$ w-Timestamp(Q)

then T_i is rolled back

→ As the value of Q is overwritten by a trⁿ that has a ts greater than T_i.

if $TS(T_i) >$ w-Time stamp(Q)

then read(Q) is successfully done by T_i

R-TS(Q) = max(R-TS(Q), TS(T_i))

Write(Q):-

→ T_i wants to write Q

If $TS(T_i) <$ R-Timestamp(Q)
then T_i is rolled back } Why ?

If $TS(T_i) <$ w-Timestamp(Q)
then T_i is rolled back.

otherwise T_i can write Q

w-Timestamp(Q) = $TS(T_i)$

Ex:-

T_1	T_2	Initial cond:-
r(Q)		$TS(T_1) = 1$
	w(Q)	$TS(T_2) = 2$
w(Q)		w-TS(Q) = 0
		r-TS(Q) = 0

1.) Read(Q)

$TS(T_1) = 1$

w-TS(Q) = 0

$TS(T_1) > w\text{-}TS(Q)$

T_1 reads 0

R-TS(Q) = 1

2.) write(Q)

$TS(T_2) = 2$

R-TS(Q) = 1

w-TS(Q) = 0

$TS(T_2) > r\text{-}TS(Q)$

$TS(T_2) > w\text{-}TS(Q)$, T_2 writes Q, w-TS(Q) = 2

3.) write(Q)

$TS(T_1) = 1$

R-TS(Q) = 1

w-TS(Q) = 2

$TS(T_1) < w\text{-}TS(Q)$

T_1 roll back

→ If there is a obsolete write, then ignore that write and proceed with other instn's.

→ This rule is called Thomas write rule.

→ This produces view serializable schedules.

Ex:-

T_1	T_2
read(B)	
	read(B)
	B = B-50
	write(B)
read(A)	
	read(A)
display(A+B)	
	A = A+50
	write(A)
	display(A+B)

$T_1 = 1$, $T_2 = 2$

r-TS(A) = 0, w-TS(A) = 0

r-TS(B) = 0, w-TS(B) = 0

1) r(B)

TS(T_1) = 1 , w-TS(B) = 0

TS(T_1) > w-TS(B)

T_1 reads B. , r-TS(B) = 1

2) r(B)

TS(T_2) = 2 w-TS(B) < 0

TS(T_2) > w-TS(B)

T_2 reads B, r-TS(B) = 2

3) w(B), TS(T_2) = 2 W-TS(B)