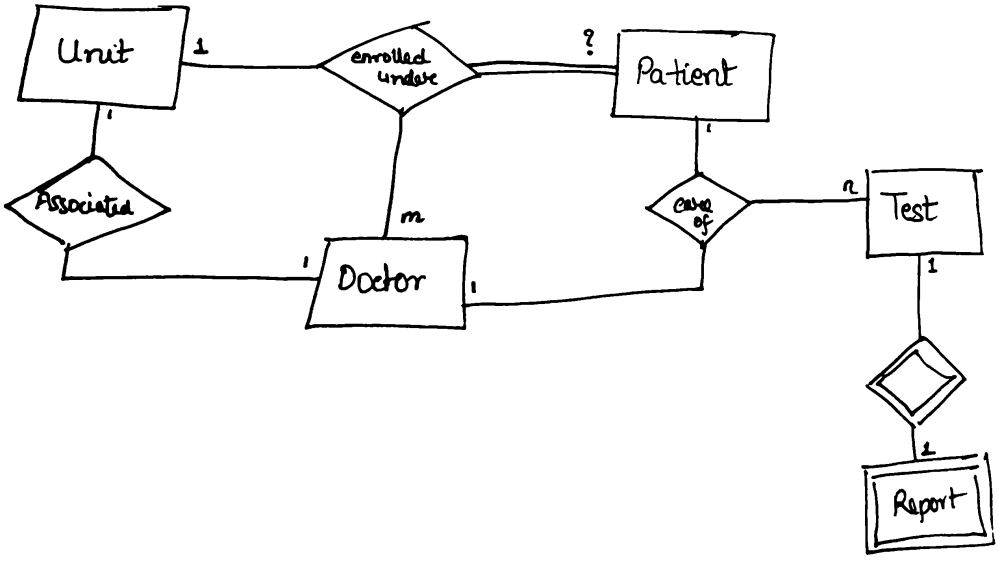

Unit
1
enrolled under
?
Patient
1
Associated
1
m
Doctor
1
1
cared of
n
Test
1
1
Report

$\{ it.iname \mid Item(it) \text{ AND } NOT\ (\exists or)(Order(or) \text{ AND } or.icode = it.icode) \}$

$T1 \leftarrow \Pi_{icode}(Order)$ // All icodes appearing in atleast 1 order

$T2 \leftarrow \Pi_{icode}(Item)$ // All icodes

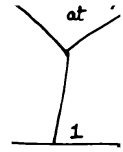
$T3 \leftarrow T2 - T1$ // All icodes not appearing in any order

$T4 \leftarrow \Pi_{iname}(T3 * Item)$

$\{ or.order\_no \mid Order(or) \text{ AND } (\exists it)(Item(it) \text{ AND } it.name = 'ABC' \text{ AND } it.icode = or.icode) \}$

$T1 \leftarrow \Pi_{icode}\left(\sigma_{iname='ABC'}(Item)\right)$

$T2 \leftarrow \Pi_{order\_no}(T1 * Order)$



Location

In timestamp based, there is no waiting for a tx^n to complete. Hence, deadlock does not occur

To fix a non-recoverable schedule we may wait for an earlier tx^n (which started earlier) to complete. Not

Never waits for tx^n which starts later

T_j read Q

```
if WTS(Q) > TS(Tj)
    rollback (Tj)
endif
Ti is the tx^n performing last write
if (bi) then
    allow Tj
else
    wait
endif
```

Unordered

rel^n state = set of n-tuples

each n-tuple is a ~~finite set of~~ mapping from A → D

$D = dom(A_1) \cup dom(A_2) \ldots$

c)

(i) $_{dcode}\mathcal{F}_{avg\ (basic)}\ (EMP)$

(ii) $\{ emp.basic \mid EMP(emp) \text{ AND } (\forall tu)$

$(\text{NOT}(EMP) \text{ OR } (tu.basic <= emp.basic)) \}$

(iii)

$\{ de.dcode \mid DEPT(de) \text{ AND NOT } (\exists em)$

$(EMP(em) \text{ AND } em.dcode = de.dcode) \}$

c) for deferred (no-steal), since ~~redo~~ undo is never required, we only need redo

↓

updation is deferred until tx^n commits

log-records are of the form

for immediate, ~~since~~ dB can be updated before ~~tx~~ tx^n commits, so we need both undo and redo

$\langle T_i, Q, V_{old}, V_{new} \rangle$

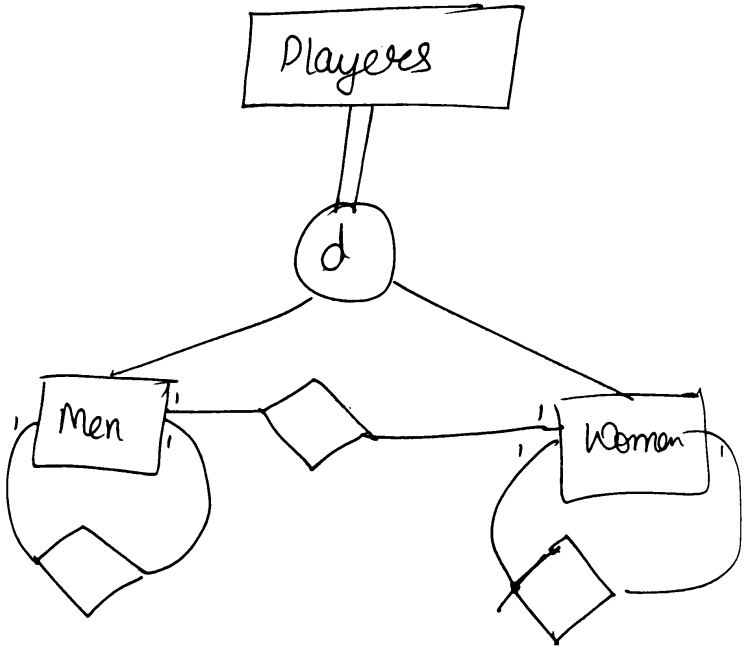
d) ~~Poor instr.~~

Given a ~~no~~ concurrent schedule S, I_i and I_j are 2 consecutive instr. belonging to diff tx^n, I_i and I_j are said to conflict if atleast 1 of them is a write operation

A non serial schedule S, if through a series of swaps of non-conflicting instr. can be converted to a serial schedule S', then S is called conflict serializable

2010

Q2C)



(a) → A rel^n schema R is made of a rel^n name (R) and a set of attribs ($A_1, A_2, \ldots A_m$)

a) → ~~Attributes are ordered~~

• ~~A relation~~ Attrib. denotes the role played by a value from the corr. domain

a) Attribs are ordered

A rel^n (or rel^n state) r of the rel^n schema R (denoted by $r(R)$) is a set of n-tuples $r = \{t_1, t_2 \ldots t_n\}$ where each n-tuple t_i is an ordered list of values $t_i = \langle v_1, v_2, \ldots v_m \rangle$ where each value v_i ($1 \leq i \leq m$) $\in$ DOM(A_i) or is a null value

(Ordering of attribs is done at the time of schema specification)

b) Attribs are unordered

A rel^n state $r(R)$ is a finite set of mappings $r = \{t_1, t_2 \ldots t_n\}$ where each tuple t_i is a mapping from R to D, $D = \text{DOM}(A_1) \cup \text{DOM}(A_2) \ldots \cup \text{DOM}(A_m)$

b) Relational Database

It is a collection of relations

Schema of rel^n DB consists of the set of schemas of the rel^n $\{R_1, R_2 \ldots R_n\}$ along with the set of their integrity constraints

State of a rel^n DB is the set of states of individual rel^{ns} $\{r_1, r_2 \ldots r_n\}$

where $r_i = r_i(R_i)$

c) Course (courseid, coursename)
Student (roll, name)
Enrollment (roll, courseid)

R. Algebra

$T1 \leftarrow \Pi_{roll}\left(\sigma_{name='JOHN'} Student\right)$

$T2 \leftarrow \Pi_{courseid}\left(T1 * Enrollment\right)$

$T3 \leftarrow \Pi_{coursename}\left(T2 * Course\right)$ ...(Ans)

R. Calc

$\{ co.coursename \mid Course(co) \text{ AND } (\exists su)(\exists en)$

$(Student(SU) \text{ AND } Enrollment$

$AND\ Su.name = 'JOHN'\ AND$

$Su.roll = en.roll\ AND\ en.courseid = co.courseid)\}$

d) Use trigger

```
create trigger item_quant_check
before insert or update
on ITEM
for each row
begin
  if :new.quantity < min_level then
      raise_application_error(-20222, 'Invalid quantity');
  endif;
end;
/
```

Q2) In context of ERD:

entity type: defines a collection of similar type entities
↓
share same attributes

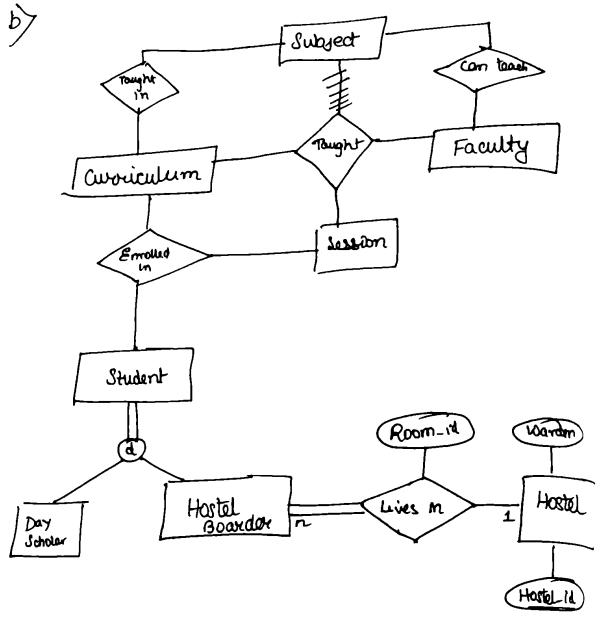
Rel^n type: Rel^n type R ~~defines a set of~~ among n entity types ~~$R_1, R_2, \ldots$~~ $E_1, E_2, \ldots E_n$ defines a set of associations among entities from these entity types

Disjoint constraint: Used in extended ERD

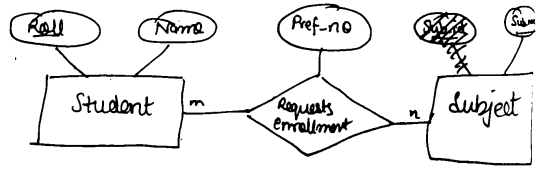
Specifies that the subclasses of the specialization (of the superclass) must be disjoint. Every entity can be a member of almost one of the subclasses

whether the general entity type can belong to ~~an~~ more than one specialized entity type or not

b)



c)

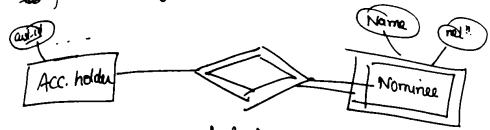


Student (Roll, Name)
Subject (Sub_name)
Enroll_request (Roll, Sub_name, pref_no);

Q3)

a) A weak entity type is an entity type that does not have its own primary key.

Given an entity of a related strong e.type, the identifying rel^n is traversed to obtain the corr. subset of weak. etypes within that subset, a discriminator is used for identification



related

Copy. P.key of strong entity type (as foreign key) into the schema of weak e.type

P.K → PK of strong etype ∪ discriminator

Nominee (Cust_id , Name , . . .);

b) DML precompiler

Converts DML statements embedded in host language into equivalent host language statements

DB manager

SW module forming the core part of the DBMS

- Interacts with file mgr. module of the OS
- Concurrency control
- Recovery against failure
- Integrity enforcement

c) A decomposition of a rel^n schema R into $R_1, R_2 \ldots R_n$ to said to be lossless when

$$\Pi_{R_1}(r) * \Pi_{R_2}(r) \ldots * \Pi_{R_n}(r) = r$$

Let F be a set of FDs on R and let R_1, R_2 be a decomp. of R. then decomp. is said to be lossless when atleast one of the foll. FDs is in F^+:

(i) $R_1 \cap R_2 \longrightarrow R_1$

(ii) $R_1 \cap R_2 \longrightarrow R_2$

i.e., if $R_1 \cap R_2$ forms a superkey of either R_1 or R_2, the decomp is lossless

d) Consider a secondary index on a data file

Consider this 2nd ary index as an ordered data file (ordered on indexing field) with fixed length records. Create a p.index for this file. This forms the 2nd level.

Construct another p. index for this p. index in 2nd level (this forms the 3rd level)

This process is continued upto k-levels until the index at the k^{th} level occupies only 1 block on disk

Accessing each level requires a block access

Accessing the actual data file takes $k+1$ block accesses

Q4) P. dependency :

i) Participation of an entity E in a rel^n set R is said to be total if every ~~entity~~ entity in E participates in atleast one rel^n in R
If only some entities in E participate in rel^n in R, then rel^n is said to be partial

b) a) represent_name
B represent_address
C represent_phone

A → BC
D → EF
AD → G

D doctor_name
E doc_specialization
F doc_ph
G date_visit
} multivalued

i) Problems

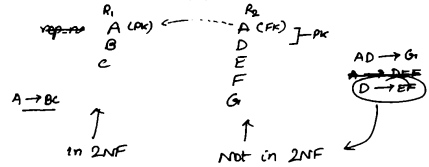
Anomalies : insertion : → Redundancy : doctor details repeated
→ Null values : Not possible to insert new doctor w/o null in r_name & vice versa

deletion : → Loss of info if rep is deleted, and rep is the only rep for a doctor, doc. is also lost

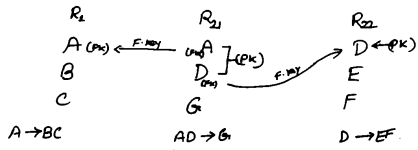
modification → edit doc. details requires changing on a no. of tuples

Applying 1NF

All attribs should be atomic and single valued



Applying 2NF



~~3NF~~

3NF is satisfied

c) BCNF states that for every non triv FD
$X \rightarrow Y$, X must be a C. key

Consider: Elections(Roll, elect-scode, faculty)

Roll, elect-scode → faculty

faculty → elect-scode * valid in 3NF Not in BCNF

if we make a new table

Elective (roll , faculty)

fac_dec (faculty , scode)

then roll, scode → faculty dependency is lost

BCNF is not dependency preserving

d) MVD

given a rel^n schema R , $X \subset R$, $Y \subset R$ and $R - [X \cup Y] = Z$, $X \twoheadrightarrow Y$ or X multidetermines Y

if, $\forall$ for any tuples t_1, t_2 , if $t_1[X] = t_2[X]$

then there exists t_3 and t_4 such that

(i) $t_1[X] = t_2[X] = t_3[X] = t_4[X]$

(ii) $t_1[Y] = t_3[Y]$ & $t_2[Y] = t_4[Y]$

(iii) $t_1[Z] = t_4[Z]$ and $t_2[Z] = t_3[Z]$

abc
deade
abe
adc

Also, if $X \twoheadrightarrow Y$ then $X \twoheadrightarrow Z$ hence,

$X \twoheadrightarrow Y|Z$

Q5) Student (roll, name)

Assignment (assign_id , due_dt)

Submission (roll, assign_id, subdate , score)

(a) select roll, count(*) from submission group by roll ;

(b) delete from Student where roll not in (select distinct roll from Submission);

c) ~~For each student~~

```
select sub.roll, sub.assign_id from submission sub
~~val~~ student st where ~~st.roll~~ sub.roll != st.roll
```

d)
```
select assign_id from submission group by
    assign_id having avg(score) >= ALL
        (select avg(score) from submission
            group by assign_id) ;
```

e)
```
select name. from student where roll in
    (select roll from submission group
        by roll having sum(score) >= ALL
            (select sum(score) from submission
                group by roll) );
```

Q6)

a) Transactions

ACID property

Atomicity: Either the effect of the tx^n will be there as a whole, or ~~there~~ the tx^n will ~~be~~ have no effect at all.
(Responsibility of recovery module)

Consistency preserving: Tx^n must be logically correct. After execution from a consistent state, DB will move to another consistent state
(Responsibility lies with the programmer)

Isolation Even if a tx^n is running in a concurrent environment, it must appear as being executed in isolation (not affected by other tx^{ns})
(Responsibility of concurrency control module)

Durability If a tx^n completes successfully, its effect must be durable. Must be reflected in the DB
(Responsibility of recovery module)

States :

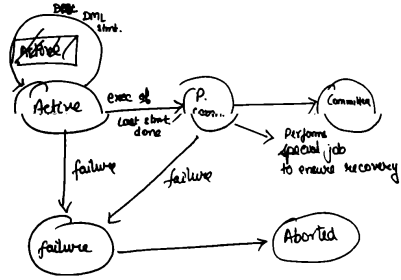
(i) Active tx^n stays in this st. while exec.

(ii) Partially committed : after fin. stmt. is executed

(iii) Committed : after successful completion

(iv) Failed : after discovery that exec. can no longer proceed

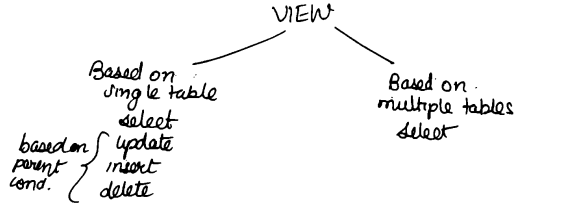
v) Aborted : after tx^n is rolled back & DB is in previous stable state (prior to start of tx^n)



b) Result (roll, q_no, p_no, score)
Summary (roll, total_score)

```
declare
    cursor c1 is select roll, q_no, sum(score) as ccsum from Result
                 group by roll, q_no order by desc sum(score);
    cursor c2 is select distinct roll from Result;
    score_ques number;
    count_ques number;
begin
    cursor c2 is select distinct roll from Result;
begin
<<outer>>
    for i in c2 loop
        score_ques := 0
        count_ques := 0
<<inner>> for j in c1 loop
            if j.roll = i.roll then
            score_ques := score_ques + j.ccsum;
            count_ques := count_ques + 1;
```

c) View: View is a logical table, not a physical table
Provides data abstraction, a user can only see the data
Can be created from 1 or more tables
If schema of any of parent tables change view must be recreated
Adding into view adds to parent table



```
        endif;
        if count_ques = 5 then
          exit inner;
        endif;
      end loop inner;
      insert into summary values (i.roll, score_ques);
    end loop outer;
  end;
  /
```

Q7)

a) Ordered file: records in a file are ordered/stored following an ordering ~~with some~~ based on one or more fields or attribs

Unordered file: Not ordered wrt any field. Order of insertion is maintained

Insert data

- Easier in unordered & faster
- for ordered, first locate position, then make space and insert record. Comparatively more complex

Search

→ if search based on ordering field, then
⇒ for ordered file, binary search
⇒ for unordered, lin. search (slower)

→ if search not based on ~~no~~ ordering field, then linear search.

b) P.index : indexing field same as ordering key.

Ordered file with fixed length records of the form <index field value, block ptr>

Datafile

- first record of each block is called block anchor
- corr. to each anchor, there is entry into p.index
- if 1 block holds many records, then no of p.index entries = no of blocks << no of records

Hence, index is sparse

c) Tuples ordered on Dcode :

↓

not a candidate key

Create a cluster index (indexing field = ordering field)

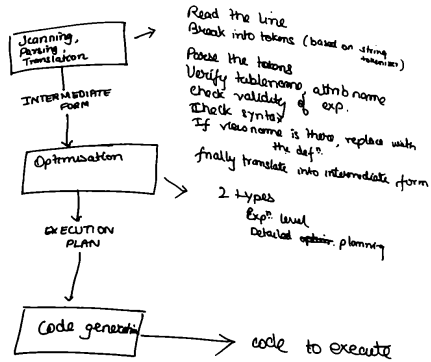
Using Cluster index, we can easily get all tuples corresponding to a particular Dcode

For ~~student~~ ecode search:

Create sec. index with ecode as indexing field

(Multilevel indexing can also be used)

d) Query processing



&

e) Hash Join

/* Reln R1 and R2 stored on disk
Joining attrib is A
h is the hash fn ← k buckets in all
i = h(A), i = bucket no where tuple is stored */

for (i=0 ; i<k ; i++)
    $h_{1i} = \{\}$
    $h_{2i} = \{\}$

```
for each tuple t1 in R1:
    i = h(t1[A])
    h1i = h1i ∪ {ptr to t1 in R1}
for each tuple t2 in R2:
    i = h(t2[A])
    h2i = h2i ∪ {ptr to t2 in R2}
for (i=0; i<k; i++)
{
    S1 = {}
    S2 = {}
    for each ptr p in h1i:
        S1 = S1 ∪ {*p}
    for each ptr p in h2i:
        S2 = S2 ∪ *p;
for each tuple t1 in S1:
    for " " t2 in S2:
        check (t1, t2) and output
```

Q8) steal / No force

a) No-steal : requires a lot of memory buffers
Force : Greater I/O time required

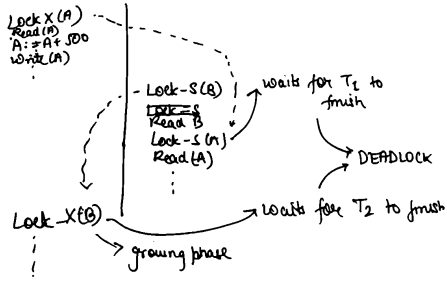
b) 2 phase locking

2-p A tx^n has 2 phases
growing → init., a tx^n is in growing phase
Locks can be requested in g. phase

shrinking → Once a lock is released tx^n is in shrinking phase. No more locks can be requested in s. phase

~~Always~~ Ensures con. serializability

Dead lock can still occur



In timestamp based, there is no waiting for a tx^n to complete. Hence, deadlock does not occur

To fix a non-recoverable schedule we may wait for an earlier tx^n to complete. Not ~~or~~ which started earlier

Never wait for tx^n which starts later

~~To~~ T_j read Q

```
if WR TS(Q) > TS(Tj)
    rollback (Tj)
endif
Ti is the txn performing last write
if (bi) then
    allow Tj
else
    wait
endif
```

c) ~~In a non-concurrent env~~, checkpoints indicate that
- → all log records sent to log file
- → All modified ~~data~~ blocks written to ~~DB~~ disk

~~Any~~ ~~Not~~ No operation before checkpt needs to be redone

In non-concurrent, only 1 tx^n is active during checkpt.

for concurrent, multiple $tx^{n's}$ active

<checkpoint L>, list of all active $tx^{n's}$

2013

Q1) Attendance (roll, scode)
Result (roll, scode, score) ??
Backlog (roll, scode)

b) Given a rel^n R1 (referenced rel^n) and R2 (referencing rel^n),
PK = P. key of R1
FK = subset of attributes in R2 such that
(i) Dom (FK) in R2 = Dom(PK) in R1
(ii) for any tuple t_2 in R2, either
$t_2[FK]$ is null
or, $\exists t_1 \in R_1$ such that $t_2[FK] = t_1[PK]$

then, FK in R2 is said to be the foreign key referencing PK of R1

Q2)
a) Dom: set of atomic values which an attribute can assume

Rel^n: rel^n of rel^n state r of schema R is given by the set of n-tuples $\{t_1, t_2 \dots t_n\}$
where $t_i = \langle v_i, v_2 \dots v_m \rangle$

Schema: structure of the database
specifies name & a set of attribs

b) Ordered
↳ rel^n state = set of n-tuples
tuple = ordered list of values
Ordering done @ schema specification

Unordered
rel^n state = set of n-tuples
each n-tuple is a ~~finite set of~~ mapping from
$D = dom(A_1) \cup dom(A_2) \ldots$

a)
(i) $\mathcal{F}_{avg\ (basic)}$ (EMP)

(ii) $\{ emp.basic \mid EMP(emp) \text{ AND } (\forall tu)$
$(NOT(EMP) \text{ OR } (tu.basic <= emp.basic))\}$

(iii)
$\{ de.dcode \mid DEPT(de) \text{ AND } NOT(\exists em)$
$(EMP(em) \text{ AND } em.dcode = de.dcode)\}$

Q3)

a) entity type: defines a collection of similar types of entities
   - schema or intension

entity set: A collection of entities of similar types (of a part. entity type)
   - extension

b) Constraints on ERD

- Participation:
  2 types
  - total: participation of an entity type E in rel^n R is said to be total if every entity in E must participate in atleast one relationship in R
  - partial: every entity in E need not participate in a rel^n in R
- Cardinality
  no of entities to which another entity can be associated via a rel^n set
  - One one
  - One many
  - Many 1
  - Many many

d) for one-one, choose PK of one ~~entity~~ rel^n and store it as FK in the other

for many-one, or one-many choose the PK of the 'one' side and store as FK in the 'many' rel^n side

Q4)

(a)

A Enr_id (PK)
B Name
C DOB
D Categ.
E prof_no
F inst_code
G inst_name
H course_code
I course_name
J ~~pay_info~~
K Draft_no
L Draft_dt
M Bank_name
N branch
O amnt

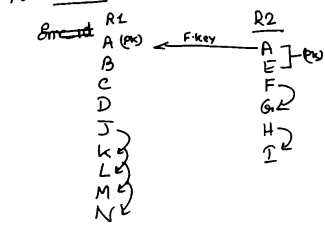
E, F, G, H, I — M. valued

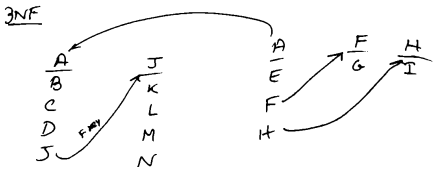
A → BCDJ
AE → FH
F → G
H → I
J → KLMN

Prob with insert:

Redundancy
Null values
Multivalued attrbs

~~FDs~~ INF





b) Given a rel^n schema R, and a FD set F which holds on R

Let $R_1, R_2 \ldots R_n$ be a decomp. of R

Let F_i be the set of all dep. that can be verified using R_i,

~~F_i is the set of all~~

Restriction of F to R_i (called F_i) is the set of all FDs in F^+ $X \longrightarrow Y$ where $X \subseteq R_i$ and $Y \subseteq R_i$

Let $F_c = F_1 \cup F_2 \ldots \cup F_n$

then, the dec. is dep. pres. when

$F_c^+ = F^+$

Adv: Allows ~~updation~~ validation during updation of database (so that ~~no~~ no illegal rel^n is formed) without having to compute the join

[redacted]

Q5)

a) ~~Vendor~~
~~select vendor-id, trade from vendor, item~~
~~where~~

Q6) Variable length records: difficult to handle

Requires separators
- record sep
- field sep
- field name sep
- value sep

} must not appear as part of data

~~To enter~~

Reasons

- In same file, records of diff type
- records of same type, but
  - Var. length fields
  - Optional fields → requires field name, value
  - Multivalued ~~dep~~ fields (requires value sep)
  - (Sep. must not be part of data)

Convert:

- Var. length : fixed length
- Optional : Mandatory, with option for NULL
- Multivalued : Provision to store max values or store in new table altogether

Causes wastage of space

b) Ordered on ECODE

↓

p.key

Create p. index on ordering key

Create sec. index on DCODE (contains 1 level of indirection, since for each D_code value, multiple record pointers are required)

c) B/B+ tree over BST in indexing:

- can have upto (n-1) children
- balanced
- No of nodes reduced
- each node stored in a disk block

7)

a) Simple iter., with one rel^n in memory

b) Active: tr^n stays in active state as long as stmts are exec.

P. comm: enters this state after exec. of last stmt.

Comm: After succ. completion

Failed: after discovery that normal execution cannot proceed further

Aborted: After rollback

c) for deferred (no-steal), since undo is never required, we only need redo

updation is deferred until tx^n commits

log-records are of the form

for immediate, since db can be updated before tx^n commits, so we need both undo and redo

$\langle T_i, Q, V_{old}, V_{new} \rangle$

d) Poor instr.

Given a concurrent schedule S, I_i and I_j are 2 consecutive instr. belonging to diff tx^n, I_i and I_j are said to conflict if atleast 1 of them is a write operation

A non serial schedule S if through a series of swaps of non-conflicting instr. can be converted to a serial schedule S', then S is called conflict serializable

2012

Q1)

b) Key constraint

Superkey

Subset of attribs such that for every tuple in the relation have unique value

for any t_i, t_j

$t_i[SK] \neq t_j[SK]$

∵ No 2 tuples can be same, ~~we take~~ all attribs can be taken as a Superkey

C. key

Minimal s.key, of which, no proper subset is a superkey

There may be many C.keys
1 is selected as P.key

Ref. integrity

F.key defn.

And referential integrity constraint from ~~refer~~ R2 → R1 is said to hold

c) $T1 \leftarrow \Pi_{icode}(ORDER)$

$T2 \leftarrow \sigma_{iname}\left(ITEM * \left(\Pi_{icode}(ITEM) - T1\right)\right)$

$\{ it.iname \mid Item(it) \text{ AND } NOT(\exists or)(Order(or) \text{ AND } or.icode = it.icode) \}$

Q2)

a) ERD

↳ ~~used for~~ ~~es~~ data model used for conceptual design of DB or to represent data requirements in the system

↳ ~~represents the~~

↳ Shows entities, their description, relⁿ

b) Composite : → non atomic collection of other attribs

Multivalued → can ~~hold~~ have multiple values for a single entity

Q3)

a) F.D – constraints between 2 sets of attributes

ie say $R \Rightarrow$ schema $X \subseteq R$, $Y \subseteq R$

then $X \rightarrow Y$

means that for any $t_1, t_2 \in r(R)$

if $t_1[X] = t_2[X]$

$\Rightarrow t_1[Y] = t_2[Y]$

$X \rightarrow Y$ and $WY \rightarrow Z$

$WX \rightarrow Z$

$X \rightarrow Y$

$WX \rightarrow WY$

Q4c) ~~100 by~~ record length $r = 100$ bytes

Block size $= 2^2$ bytes $= b$

bfr = blocking factor $= \left(b/r \right) = 5$

thus, 5 records per block

20000 records $\Rightarrow$ 4000 blocks

~~If file is unsor~~

File is ordered based on key field

$\therefore$ No of block accesses $= \lceil \log_2 4000 \rceil$

$= 12$

Using P. index

Let P. index record entry size = 20 bytes

Now, no of P. index entries = no of blocks = 4000

$bfr = \lfloor b/r \rfloor = \lfloor 512/20 \rfloor = 25$

25 records per block

4000 records → 80 blocks

∴ No of block accesses to get the block no storing required data record

$= \lceil \log_2 80 \rceil = 7$

1 additional block access for actually accessing data file

7+1 = 8 block accesses

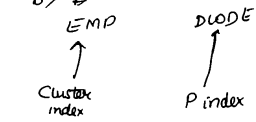
Q5) A (a1, a2)

B (a1, b1, b2)

Since B is weak entity type, we need PK of strong entity type related to it by the identifying rel^n ∪ discriminator as PKey of B

Create table A (a1 number primary key,
a2 number);

Create table B (b1 number, a1 number
references A(a1) b2 number,
also primary key(a1, b1));



$\sigma_{emp.dcode = 'd1'}(EMP * DEPT)$

take select as early as possible

$\sigma_{EMP.dcod = 'd1'}(EMP) * DEPT$

use cluster index to compute

c) Merge join

R1 & R2 are physically stored together sorted on joining attrib

p1 = ptr to first tuple of R1

p2 = " " " " " R2

```
while (p1 != NULL & p2 != NULL)
{
    t1 = *p1
    p1++;
    s1 = t1
```

```
S2 = {}
while (p2! = NULL && (*p2)[A] == t1[A])
    t2' = *p2
    S2 = S2 ∪ p2
    p2++

for each tuple t1 in S1
    for each tuple t2 in S2
        put (t1, t2) in O/P
```

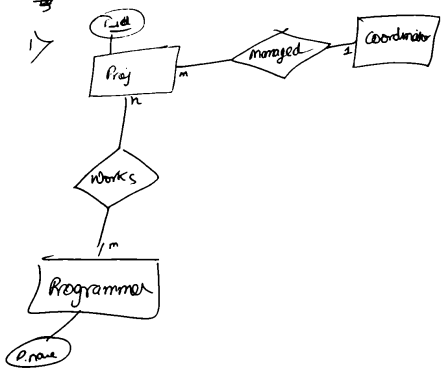
}

20/12

Q5

b) create or replace trigger bullshit
before insert or update
on delivery
for each row
declare
  pending_qty pending.qty_pend%TYPE;
begin
  select qty_pend into pending_qty
    from pending where order_no = :new.or
    AND item_code = new.ite
  if ~~qty=p~~ pending_qty

2011



Programmer (P_name, ...)

Proj (Proj_id, ..., co_name)

Co-ordinator (CO_name, ..)

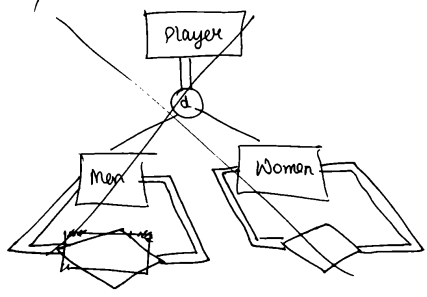
Works_on (Proj id, programmer P_name)

2013

Q5) select vendor_id, icode from vendor, item
where (vendor_id, icode) not in
(select sl.vendor_id, sicode from
supply sl, supply_details sd where
sl.supply_id = sd.supply_id)

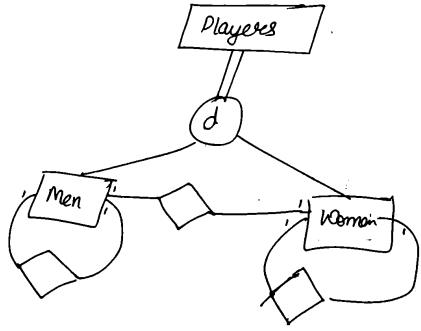
b) select s.vendor_id, sum(qty*rate) as amount
from supply s, supply_details sd where
s.supply_id = sd.supply_id AND
year from s.supplydt = 2012
group by vendor_id having
sum(qty * rate

c)



Q010

Q2C)



~~920~~
2010

9a) update table ~~student~~ submission
~~marks~~
set marks = 0
where exists
(select assign_no, ~~roll~~ from ~~submission~~ assignment
where ~~sub~~ subm_dt > due_dt)

(ii) ~~select distinct count (*)~~ ~~distinct roll~~
select assign_no, count (*) from
submission, assignment group by assign_no where
submission . ass_no = assignment . ass_no
AND submission . marks scored = assignment . full_marks
group by assign_no

(iii) select roll from submission ~~where~~
~~assign_no~~ group by roll
having count(*) = (select count (*)
from assignment) ;

A Consignment id (PK)
B receipt date
C Order no
D Order date
E supplier id
F supplier name
G supplier adr
H Item code
I Item name
J description
K qty

multivalued

$A \rightarrow BCE$
$E \rightarrow FG$
$H \rightarrow IJ$
$C \rightarrow D$
$AH \rightarrow K$

∴ Applying 1NF

R_1

A Consignment ID (PK)
B receipt date
C Order no
D Order date
E Supplier ID
F Supplier name
G Supplier adr

foreign key

R_2

A Consignment ID
H Item code
I Item name
J description
K qty

PK (based on FDs)

Applying 2NF

R1 satisfies 2NF (C. keys ~~P.key~~ are single attributes)

in R_2,

$H \rightarrow IJ \Rightarrow H \rightarrow I$
$H \rightarrow J$

H is part of C. key = AH

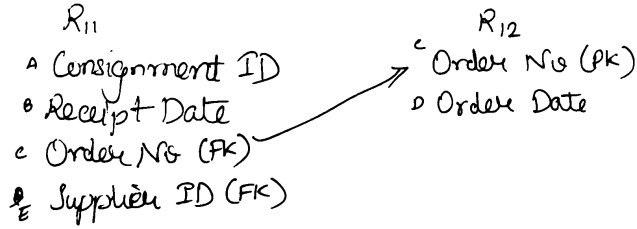
Applying 3NF

in R_1 :

$E \longrightarrow FG$

$C \longrightarrow D$

Decomposing R_1 to R_{11}, R_{12}, R_{13}



2013

Q1

Q2a