Fall-22

(1) a) ERD –

(1) b) 4 disk of 500 GB each.

So, RAID-5 ⇒ Storage Capacity = $(4-1) \times 500 \cdot GB$

$= (3 \times 500) GB$

$= 1500 GB = 1.5 TB$.

(2)a)(i) ~~AirlineNames~~ AirlineID $\mathcal{G}_{COUNT(FlightNumber)}$ ($\Pi_{AirlineName}$ (Airline ⋈ Flight))

(ii) $\Pi_{AircraftType, Capacity}$ ($\sigma_{AssignStatus = "NO"}$ (AssignTo ⋈ Aircraft))

(এটা করব when End-এ table থাকবে AssignTo relation-এ with Status-এ attribute)

(iii) $\Pi_{FlightNumber, DepartureCity, ArrivalCity}$ ($\sigma_{Email = "\%@example.com"}$ (Flight ⋈ Passenger))

(iv) $\Pi_{DepartureCity, ArrivalCity}$ ($\sigma_{Booking.FlightNumber = Flight.FlightNumber}$ (Flight ⋈ Booking)).

(2)b) Data Abstraction:

① Physical:- describes how a record is stored; ie = instructor

② Logical:- data stored in database and relationships among them.

```
type instructor = record
    ID: string;
    name: string;
    dept_name: string;
    salary: integer;
end;
```

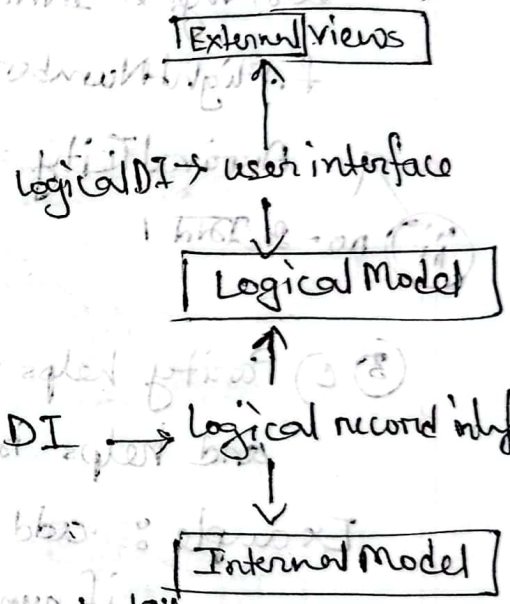
③ view level:- Application programs hide details of data types and can also hide information like employee's salary for security purpose

Three of them helps in Data Independence (DI) and security

→ physical DI : Can modify physical schema, without changing logical schema. In general, the interfaces btw various levels and components should be well defined so that changes in some parts do not influence others.

→ logical DI : modifies logical schema without changing view level.

(mark বেশি থাকলে ছবি এঁকে দেয়া যায়)



(3) a) i) Select FlightNumbers, DepartCity, ArrivalCity,
FROM Flight $ INNER JOIN Airline ON Flight.AirlinID=
Airline.AirlineID where AirlineName = 'Bangladesh
Airways';

ii) Select FirstName, LastName from Passenger, WHERE
passengerID in (select flightNumber from Flight
where DepartureCity = "Dhaka");

iii) SELECT Distinct(f.flightNumber), DepartureTime from Flight f INNER JOIN Booking b ON f.AirlineID = b.AirlinID group by DepartureTime HAVING Count(bookingID) = (Select capacity FROM Aircraft);

iv) SELECT p.firstName, p.LastName from Passenger PINNER JOIN Booking b ON p.passengerID = b.passengerID where b.passengerID in (SELECT b.passengerID FROM Booking b INNER JOIN Flight f ON b.FlightNumber = f.FlightNumber where DepartureCity = "Dhaka" AND ArrivalCity = "Chittagong");

(iii) no - 3 নাম্বার।

(3) c) Parity helps to determine the disk that is at faulty and helps in correction.

Example: add up all bits that are 1.

if number of 1 is even then parity bit 0

" " " 1 " odd " " " 1

Doing and XOR will give the parity bits

Fault tolerance: Once we get the parity bit.

Suppose the read data = 1001101

and disk-2 is at fault.

We can perform XOR with parity bit and will know which bit to flip for tolerance.

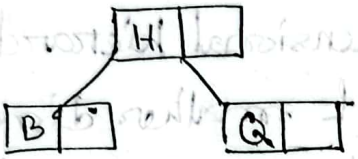
ABCD EFG HIJK LMNO P QRST UV WXYZ.

(4) a) Btree ; insert : Q, B, H, U, D, S, W, A, E, C, T ; m=3. so, 2 boxes

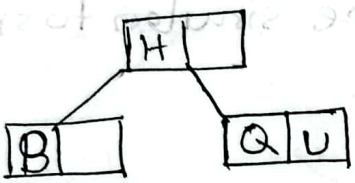
Q:

B:

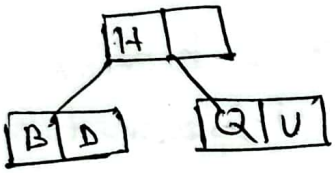
H:



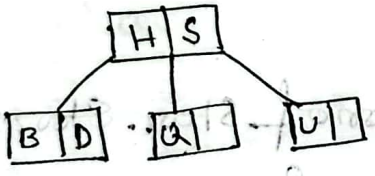
U:



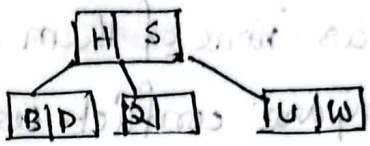
D:



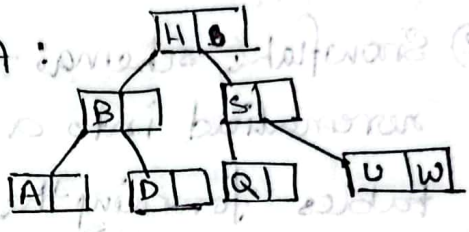
S:



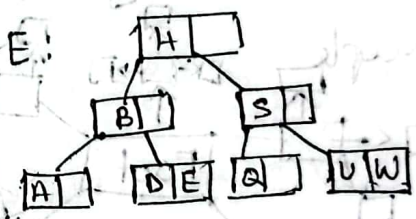
W:



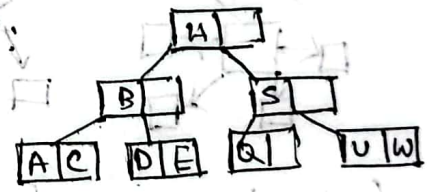
A:



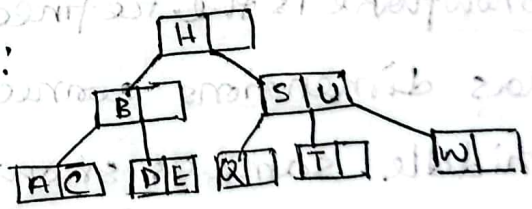
E:



C:



T:



(4) b) $T_1 = R_1(Q), R_1(Q), W_1(Q), W_1(Q)$

$T_2 = R_2(Q), W_2(Q), R_2(Q), W_2(Q)$

Here when we compare,

T_1 and T_2 both execute Read(Q) at first with no conflict.

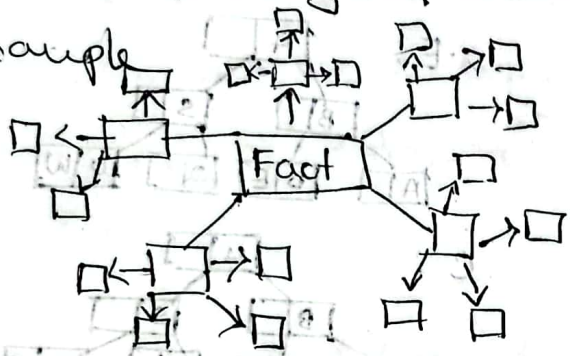
Next we see T_1 execute read and T_2 write which is a read-write conflict which means $T_1 \rightarrow T_2$

Third one is a write-read conflict which means $T_1 \rightarrow T_2$ and fourth is write-write conflict

that means (T1) → (T2) according to precedence rule. So, as none of them are creating any cyclic graph. they are conflict controllable.

(4) c) Snowflake schema: → Dimensional hierarchy is normalized into a set of smaller dimension tables forming the a shape similar to snowflake.

Example



Snowflake is the refined version of Star. Star schema has dimensions connected to a fact table in the middle same as snowflake but not the smaller dimension theough.

(5) a)

Id_Info:

BookId	genreId

pri. key → bookId
candidate → bookId, genreId

genre_details:

genreId	genretype

primary → genreId
candidate → both

Booking:

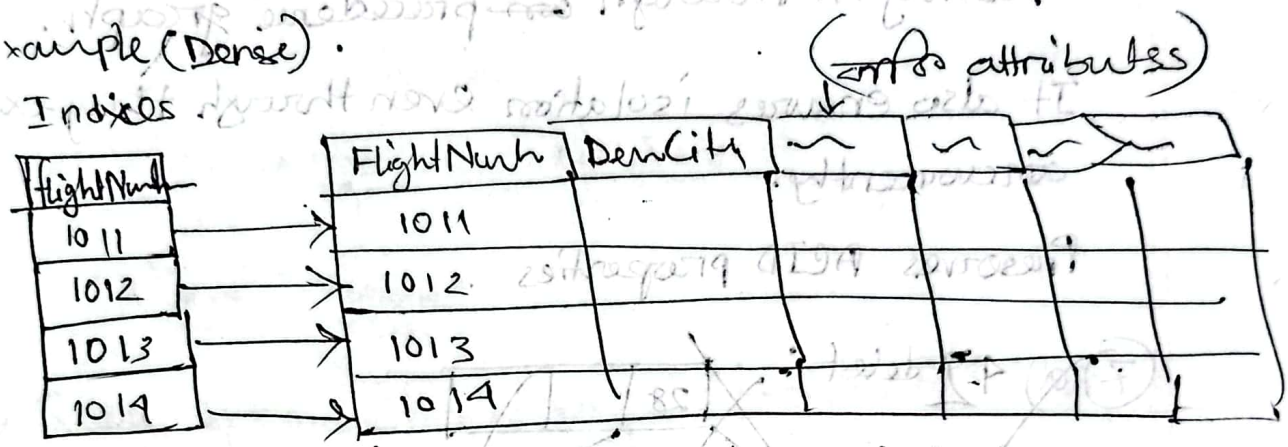
BookId	BookTitle	price

primary → BookTitle.
candidate → " + price
Foreign → BookId.

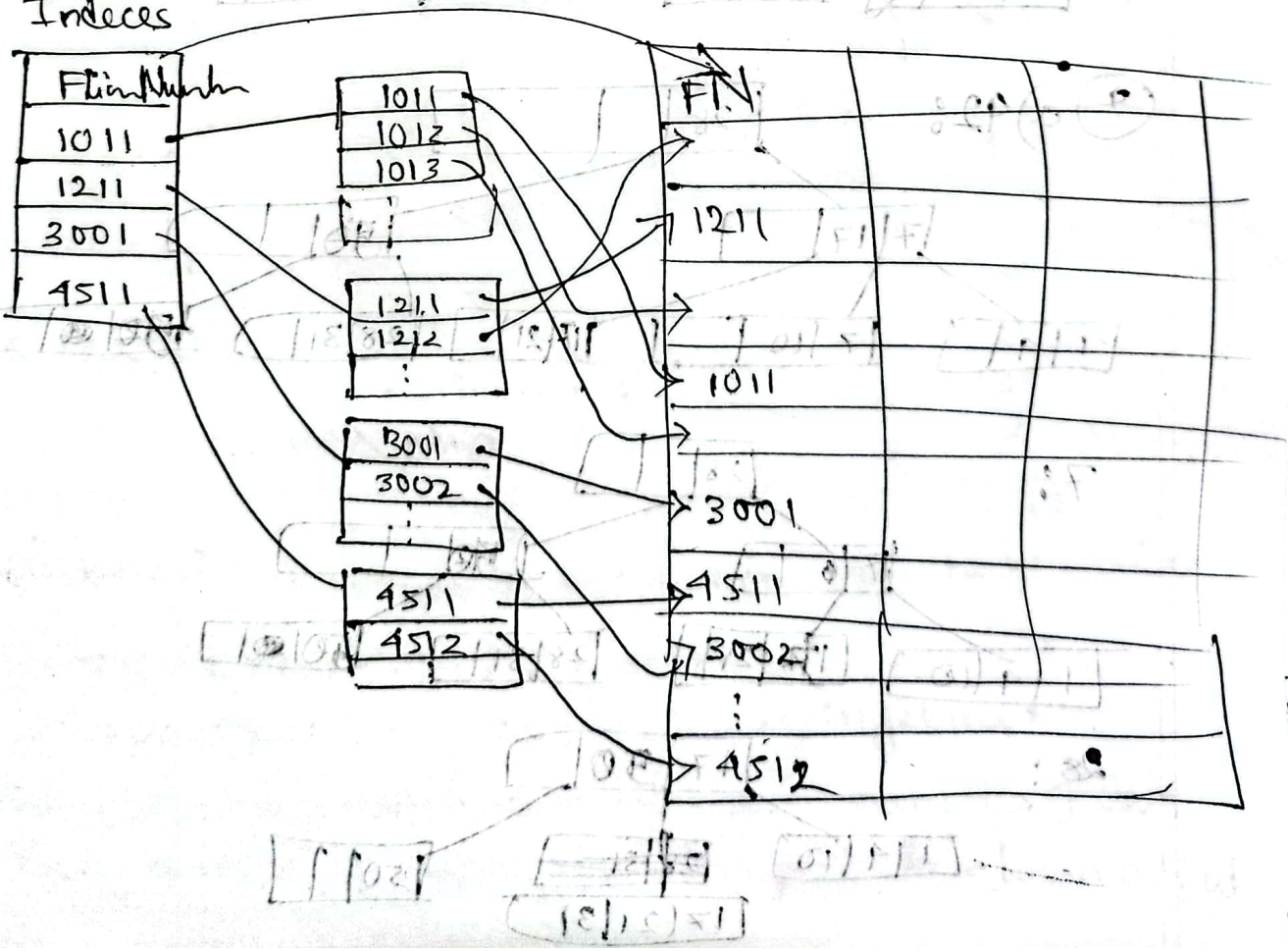
⑤ (b) If we apply secondary/nonclustering system in the database then speeding up is of data retrival is easier.

Example (Dense).

Indices



Non clustered is sorted and uses dense indexing. So retrive of it is easy :-

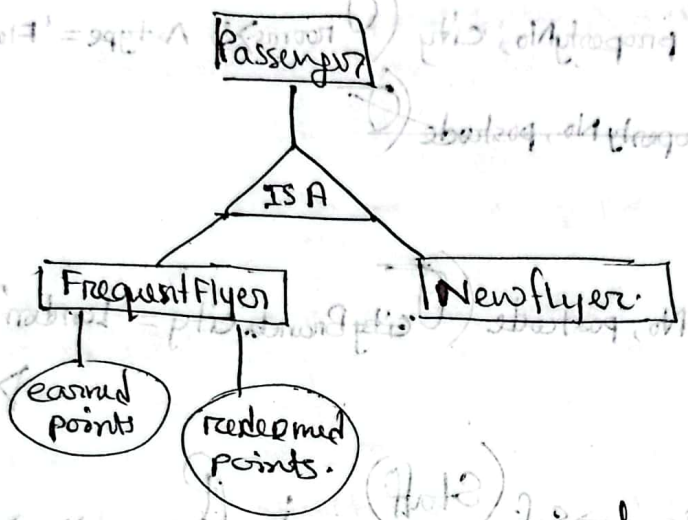


(5) c) serializability preserves database consistency and keeps the transaction order correct by testing it through ~~con~~ precedence graph.

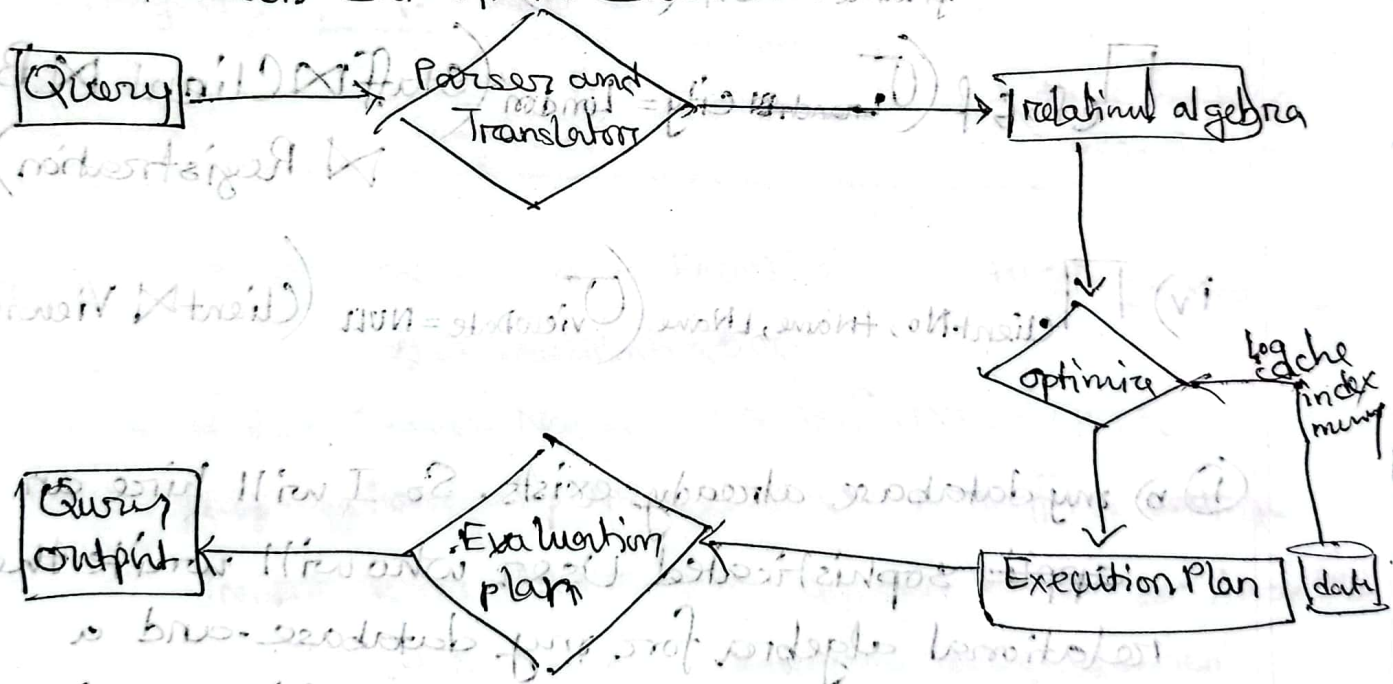
It also ensures isolation even through they execute concurrently.

Preserves ACID properties

(7) b)



(7) c) Query optimization → range of activities that extracts data from database.



Importance :- It accepts a query code and checks syntax and verifies relation by parsers.

Then translates into relational Algebra.

The optimizer fetch data from statistics of data

Then proceeds to make execution plan to show output.

SP-22

① a) i) $\Pi_{propertyNo,\ city}\left(\sigma_{rooms>5\ \wedge\ type='Flat'}\left(PropertyForRent\right)\right)$

ii) ~~$\Pi_{propertyNo,\ postcode}\left(\sigma\right.$~~

~~iii)~~

ii) $\Pi_{propertyNo,\ postcode}\left(\sigma_{~~City~~Branch.City='London'}\left(PropertyForRent \bowtie Branch\right)\right)$.

iii) ~~Client~~ $\rho_{fName \to S.f}(Staff)$; ~~$\rho_{branchNo \to branch}(Registration)$~~

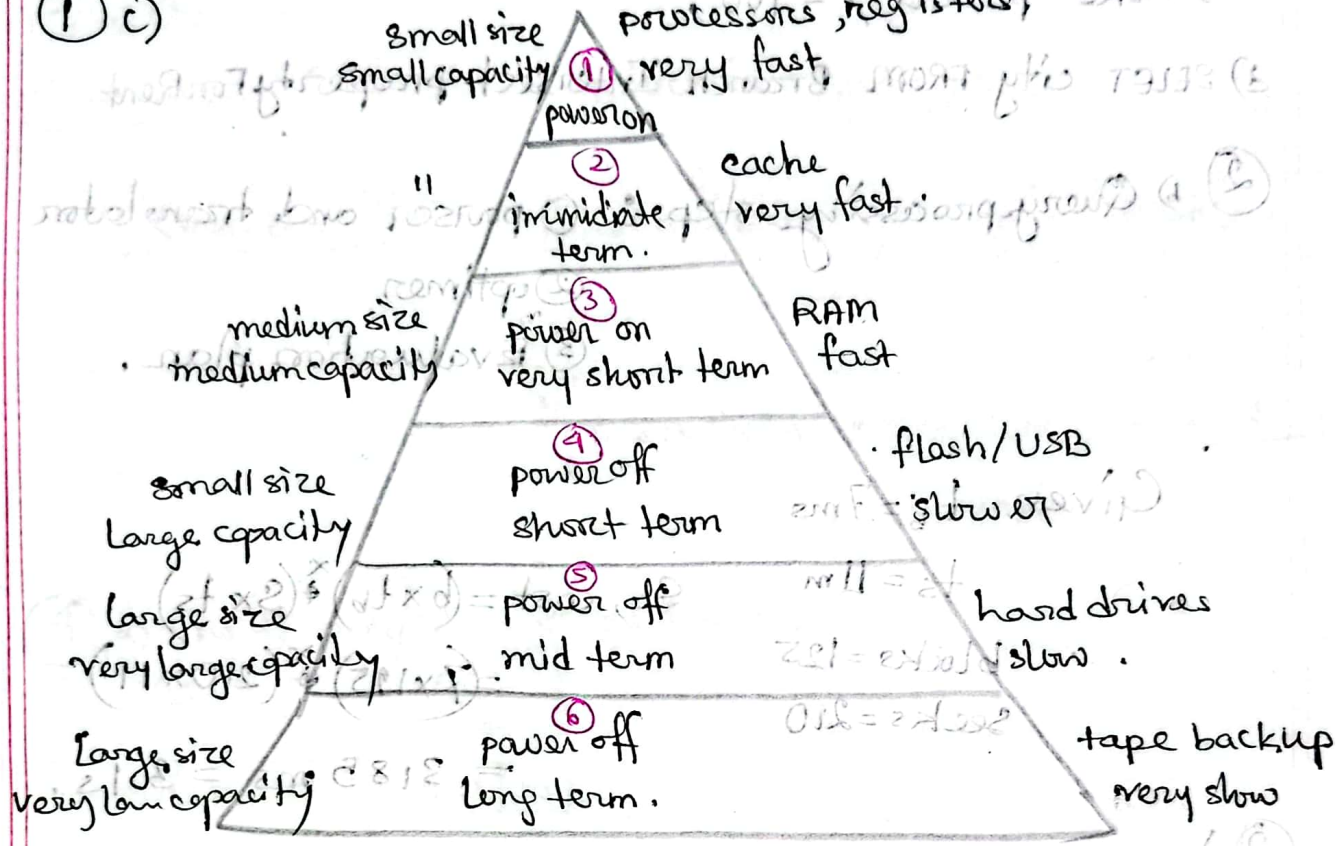
$\rho_{fName \to C.f}(Client)$;

$\Pi_{E.f,\ S.f}\left(\sigma_{~~branch~~.city='London'}\left(Staff \bowtie Client \bowtie Branch \bowtie Registration\right)\right)$.

iv) $\Pi_{clientNo,\ fName,\ LName}\left(\sigma_{viewDate=NULL}\left(Client \bowtie Viewing\right)\right)$

① b) my database already exists. So I will hire an ~~applic~~ sophisticated User who will write the relational algebra for my database and a specialized User who will make policies so bread butter comes frequently not bread-diaper. I will hire an application programmer to implement the ideas in coding language ~~used~~

(1) c)



(2) (a) a) SELECT fName, LName FROM Client WHERE prefType='Flat' AND maxRent < 500;

b) SELECT s.fName, s.LName FROM Staff INNER JOIN PropertyForRent p ON s.staffNo = p.staffNo ~~WHERE~~ INNER JOIN ~~BRANCH b ON~~ Branch b ON ~~s~~ s.branchNO = b.branchNO WHERE ~~b.brar~~ b.city = 'London' ORDER by p.propertyNo DESC;

c) SELECT ~~of~~ b.street, b.city, b.postcode, o.fName, o.LName FROM Branch b INNER JOIN PropertyForRent p ON b.branchNO = p.branchNo INNER Join Private_owner ~~o~~ o ON o.ownerNo = p.ownerno.

(PTO)

WHERE p.rent >= 100 ;

d) SELET city FROM Branch Intersect propertyForRent.

(2) b) Query processing steps: ① parser and translator
② optimer
③ Evaluation plan.

Given $t_b = 7ms$

$t_s = 11m$

blocks = 125

seeks = 210

So, cost = $(b \times t_b) + (S \times t_s)$

$= (7 \times 125) + (210 \times 11)$

$= .3185 ms = 3.1 s.$

(2) c)

14.9 **Answer:**

Suppose that the bank enforces the integrity constraint that the sum of the balances in the checking and the savings account of a customer must not be negative. Suppose the checking and savings balances for a customer are \$100 and \$200 respectively.

Suppose that transaction T_1 withdraws \$200 from the checking account after verifying the integrity constraint by reading both the balances. Suppose that concurrent transaction T_2 withdraws \$200 from the checking account after verifying the integrity constraint by reading both the balances.

Since each of the transactions checks the integrity constraints on its own snapshot, if they run concurrently each will believe that the sum of the balances after the withdrawal is \$100 and therefore its withdrawal does not violate the integrity constraint. Since the two transactions update different data items, they do not have any update conflict, and under snapshot isolation both of them can commit. This is a non-serializable execution which results into a serious problem of withdrawal of more money.

3 a) ERD :)

3 b) Client ~~⋈~~ Viewing which is right join

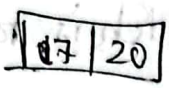
ClientNo	fName	LName	telNo	prefType	MaxRent	email

propertyNo	~~city~~ viewDate	comment	

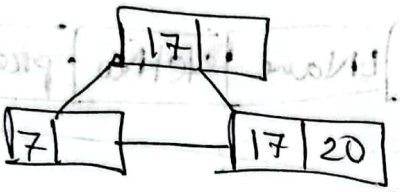
জায়গা ছিলনা

(4) a) 20:

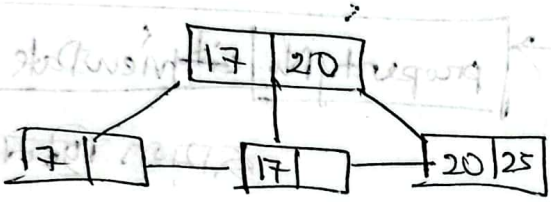
17:



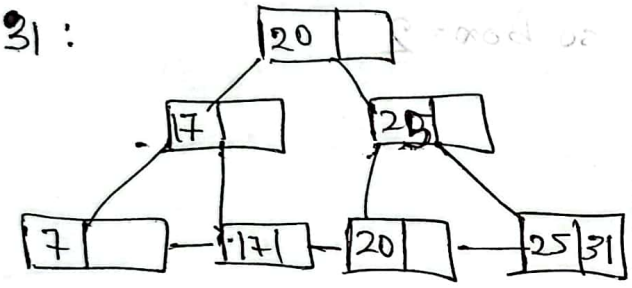
17:



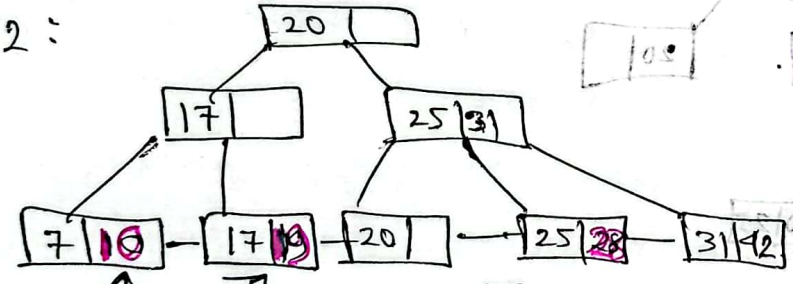
25:



31:



42:

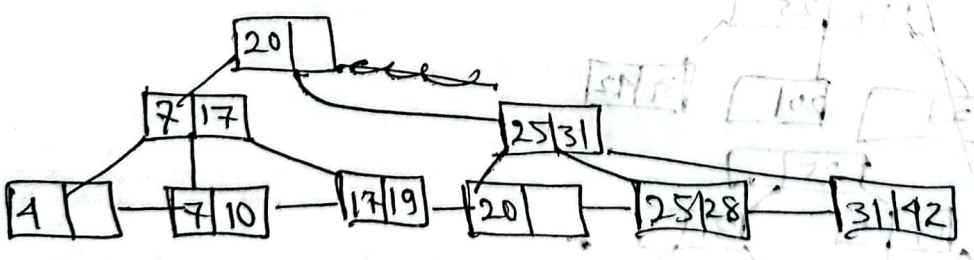


28:

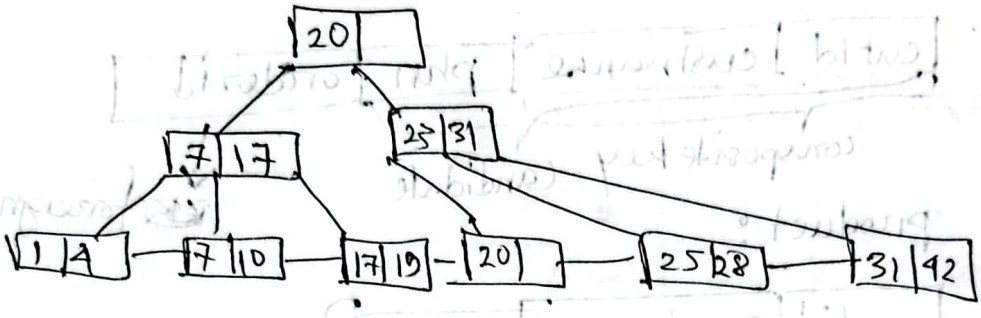
19:

10:

4:

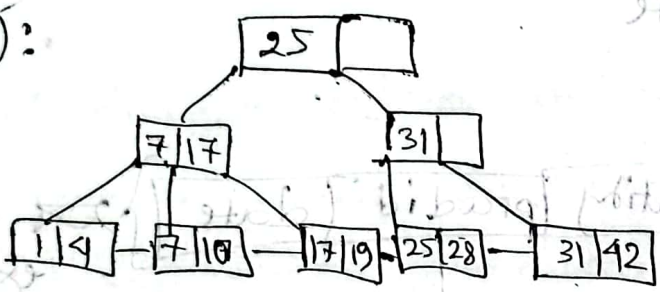


1:

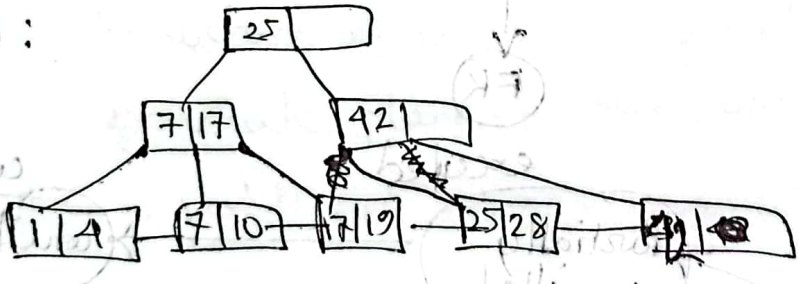


Delete:

(20):



31:



(5) a)

customer:

cust id	custname	phn	order id

composite key — candidate — foreign.

product:

prod id	prod name	price

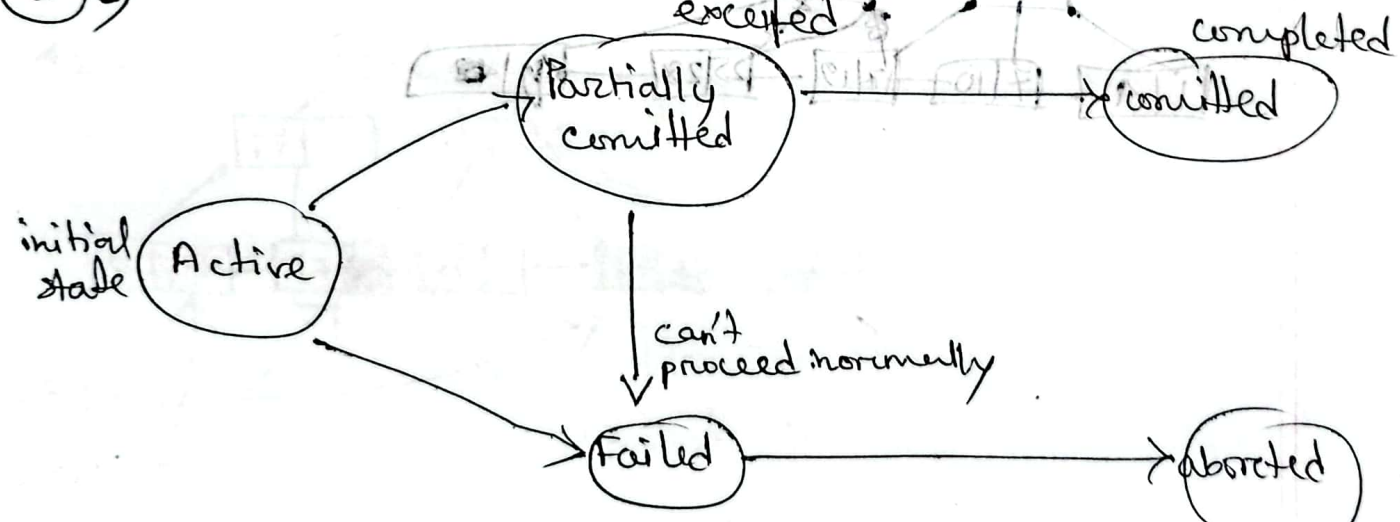
composite — candidate

order:

order id	quantity	prod id	date

Primary — FK — candidate except prod id

(5) b)



- rollback
- either restarts or kills

যত explain করত.

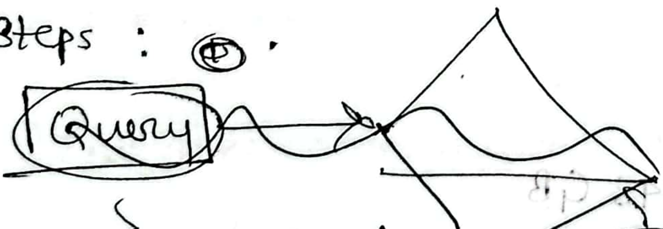
6) c)

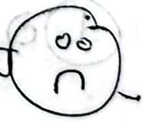
	E1	E2
One to One	(0,1)	(0,1)
One to many	(0, *)	(0,1)
many to One	(0, 1)	(0, *)
many to many	(0, *)	(0, *)

Describe

7) a) Optimizers need statistical data to make informed decisions about how to excute. It gives info on data distribution, cardinality, structure of db allowing optimizer to generate plans of evaluation or else it will lead to poor query output performance.

Steps : ①



তাই আমার diagram টা আঁকা দিস 

আসি নাই আমার

b) Dense index

Indices

JobID	JobID	Data বাকি সব
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	

This is primary index as the data are compulsorily sorted and there exists duplicate data.

7) ⊙ lec-22 - 10 slide-12 - 25তম পৃষ্ঠা

F-21

(1) a) i) $\Pi_{Title, Duration, MovieLanguage}(\sigma_{Title = \text{'\%the\%'} \vee Title = \text{'\%The\%'}}(Movie))$

ii) $\sigma_{(MovieLanguage = \text{'Japanese'} \wedge Duration > 120) \vee Country \in (\text{'JP'}, \text{'UK'})}(Movie)$

iii) $MovieId\, \mathcal{G}_{sum(ReviewStars)}(\sigma_{sum(ReviewStars) > 7}(Rating))$

iv) $\Pi_{MovieId, ReviewStars}\, \mathcal{G}_{avg(ReviewStars)}(\sigma_{avg(ReviewStars) < ReviewStars}(Rating))$

না গ্রেট 3 হবে

(1) ⊙ 4 disk → 2048 GB

RAID 1 → Storage = (2048 × 2) GB

= 4096 = 4TB.

RAID 5 → Storage = (4-1) × 2048

= 6144 GB = 6.14 TB.

(2) i) ~~SELECT * FROM~~

i) SELECT * FROM Hotel h INNER JOIN ~~CUSTOMER~~ Booking b ON h.room_no = b.room_no WHERE ~~b.cus_id IN (SELECT cus_id FROM customer WHERE city~~ Rent => 2500 AND b.cus_id IN (SELECT cus_id FROM customer WHERE city = 'Dhaka');

ii) SELECT cus_name FROM Customer WHERE cust_id IN (SELECT ~~cus_id FROM Booking DISTINCT (cus_id)~~ cus_id FROM Booking HAVING COUNT-DISTINCT(cus_id) > 1);

iii) SELECT * FROM ~~HOT~~ Hotel WHERE room_no NOT IN (SELECT room_no ~~Fr~~ From booking);

iv) SELECT cus_id FROM Booking HAVING COUNT-DISTINCT (room_no) > 1;

(2)b — normalization importance : Fully dependent করার জন্য :

~~exple stid - এর এলোমেলো হবে~~

organize করতে, eliminate data redundancy (repeatation).

expl: | StudentID | course_no | marks | faculty_name |

এখানে student ID ব্যবহার নিলে faculty_name-এর fully dependant হয়ে যাবে আর accuracy বাড়ি হয়ে যেতে পারে better normalize করতি → (PTO)

ANOMALY

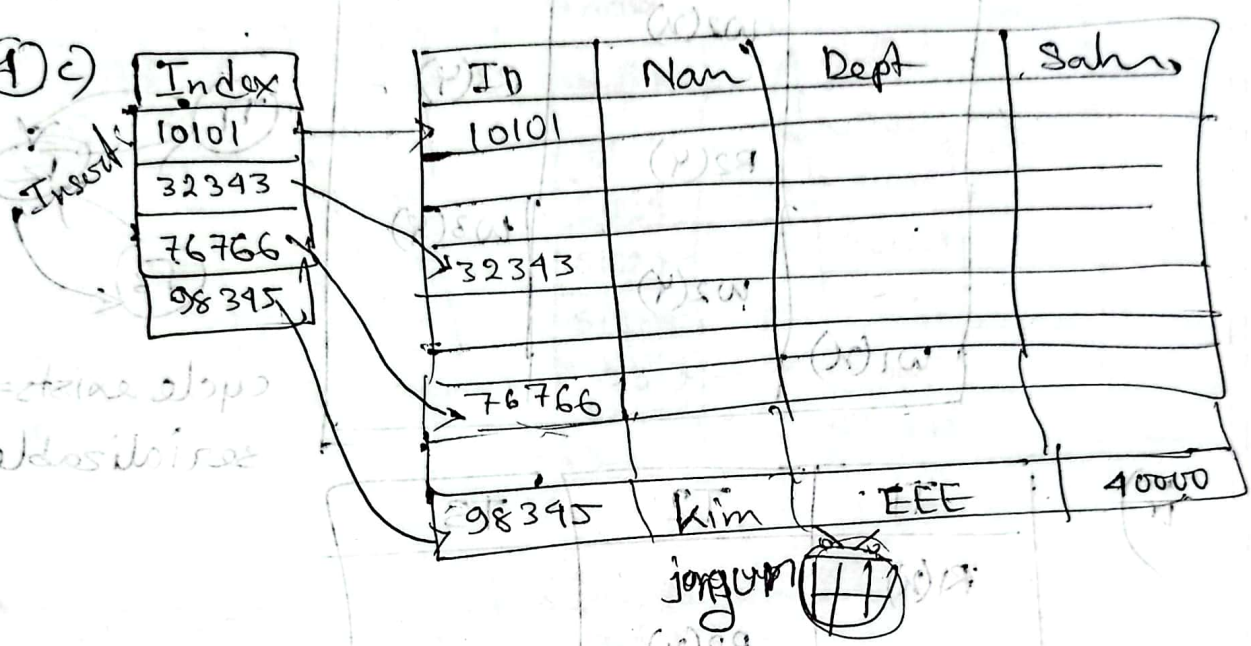
Student_Id	Cour-no	marks

Cour-no	Faculty_name

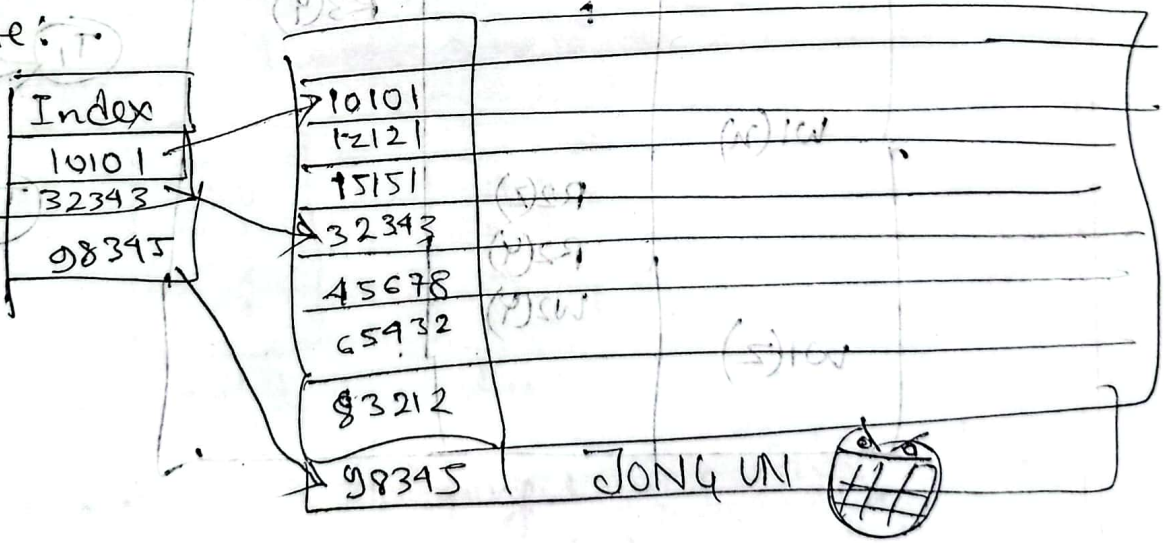
④ b) state diagram টি আঁকব

কখন → failed or committed ছাড়া option নাই।

④ c)

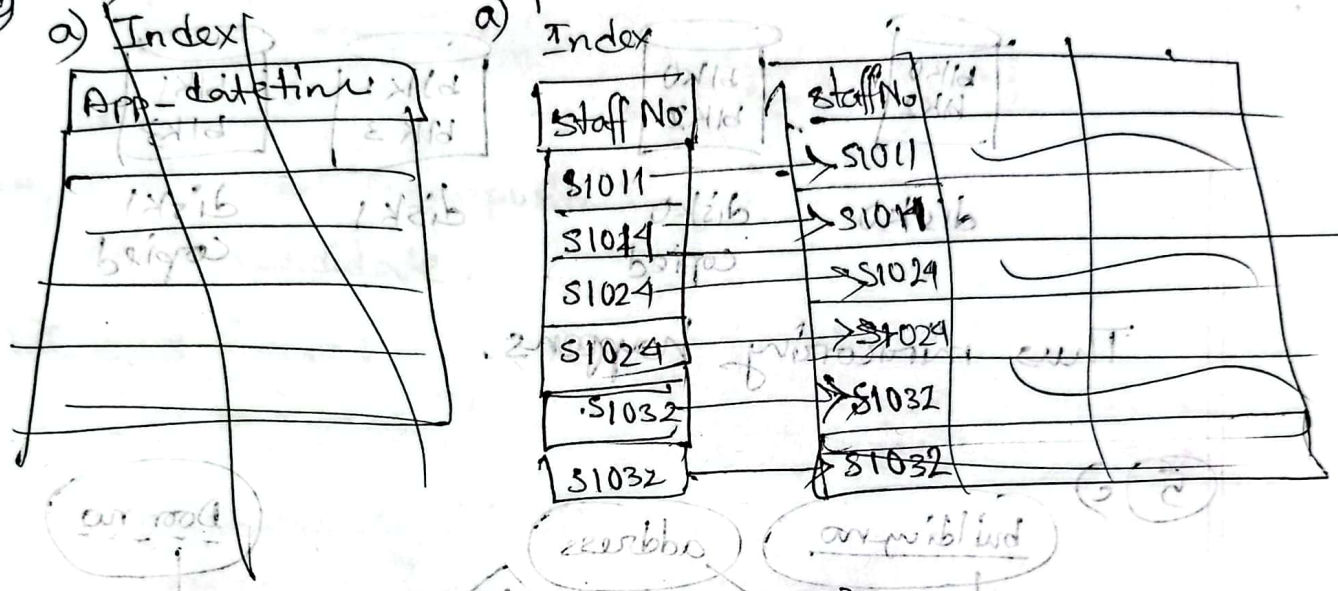


Delete:



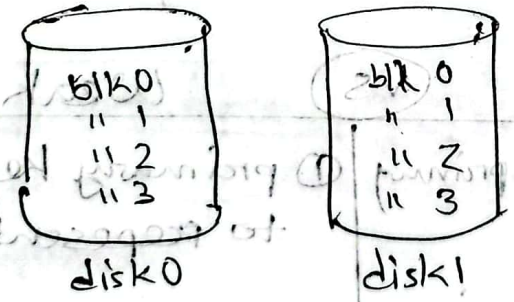
5 a) i) primary — duplicate value + sorted data.

a)



5 b) RAID-1 → mirroring use 2 disk

we keep blocks in a disk then copy it and save them in another

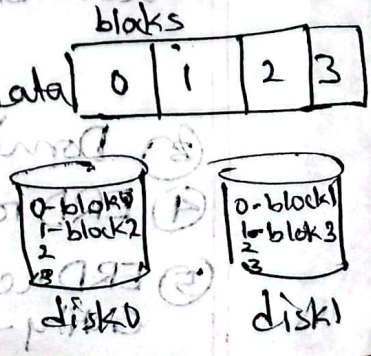


RAID-10 → both stripping + mirroring is used.

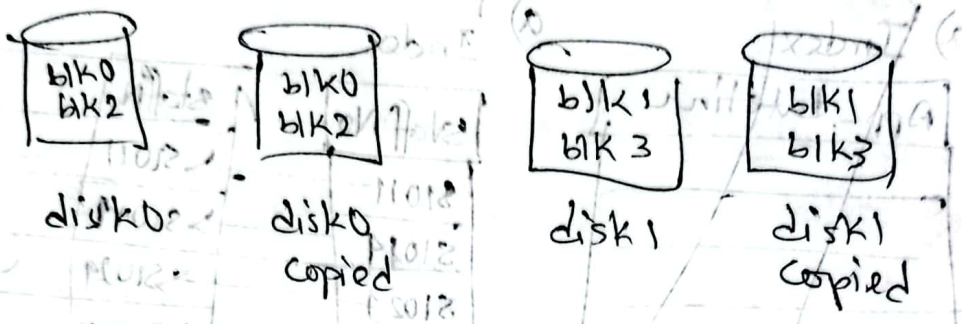
stripping is distributing data.

select disk = no. of disk % no of blks

sector = " " " "

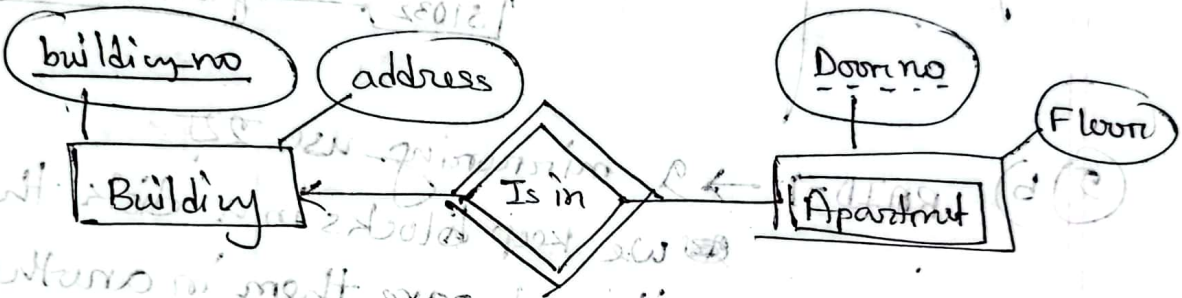


we also make 2 ~~copy of~~ copies of disk0 and disk1



Thus mirroring happens.

5) c)



Here, Apartment is weak entity, building is strong.

Strong	vs	Weak
① It always has a primary key		① primary key is not enough to represent it
② Here, building_no is enough strong to indicate the building		② door_no is not enough to represent information. there are lots of doors but which building, which door?
③ Dominate		③ Subordinate.
④ ERD represent → rectangle		④ ERD → double rectangle
⑤ ERD relation of two strong entity → diamond.		⑤ ERD - relation of a strong & weak entity → double diamond.

⑥ a) student :

s_id	name	[illegible]	hobby	dept_name

s_id → primary

hobby + s_id + name → composite key.

Also candidate.

Grade:

CGPA	grade

primary key = CGPA.

Also candidate.

Dept:

Dept_name	dept_budget

Course_Result:

s_id	course_id

Course:

st_id	Course id	course_title	course offer dept	Dept_name	CGPA

st_id → foreign

Course id + course_title → composite

CGPA → Foreign

(6) b) i) $\Pi_{A,C}(r \bowtie t)$

A	C
a	e
c	h
a	f
NULL	g
NULL	g