$p \Rightarrow q \equiv \neg p \vee q$

$\{t \mid (\exists u)(u \notin Branch \vee u[2] \neq 'KG1P') \vee (\exists v)(v \in Deposit \wedge u[1] = v[1] \wedge t[1] = v[3])\}$

Till
Midsem syllabus : Tuple Calculus

* SQL

select $A_1, A_2, \ldots, A_n$ from $R_1, R_2, \ldots, R_m$ where P;

$\equiv \Pi_{A_1, A_2, \ldots, A_n}(\sigma_p(R_1 \times R_2 \times \ldots \times R_m))$

union intersect minus

and or not

(i) Find the name of all branches in Deposit relation.

```
select b_name          or    select distinct (b_name)
from Deposit;                from Deposit;
```

(ii) Find all customers who have account at IIT Branch.

```
select c_name
from Deposit
where b_name='IIT';
```

(iii) Find all customers having a loan, or account or both at IIT Branch.

~~select c_name~~

```
Select c_name
from Deposit
where b_name = 'IIT'
```
union
```
Select c_name
from Deposit
where b_name = "IIT
```

(iv) Find all customers having a loan at any branch and their city.

```
Select C.c_name, C_city
from Customer C, Borrow B
where C.c_name = B.c_name
and b_name = 'IIT'
```

~~(iv)~~ (v) Find all customers who have both loan and account at IIT Branch.

```
Select c_name
from ~~Custo~~ Deposit
where b_name = 'IIT'
and ~~c_name~~ c_name in
(Select c_name
from Borrow
where b_name = 'IIT')
```

(vi) Find all customers who have account at IIT Branch but no loan from IIT Branch.

```
Select c_name
from Deposit
where b_name = 'IIT'
```

```
(Select C_name
 from Borrow
 where b_name = 'IIT')
```

(vii) Find all branches that have more assets than any branch located in KGP.

```
Select T.b_name
from Branch T, Branch S
where T.asset > S.asset
and S.b_city = 'KGP'
```

```
Select b_name
from Branch
where asset > any
(Select asset
 from Branch
 where b_city = 'KGP')
```

> any < any > all < all

(viii) Find all customers with balance lies between 50000 and 60000.

```
Select C_name
from Deposit
where Balance > 50000 and Balance < 60
```

```
Select C_name
from Deposit
where Balance between
   50000 and 60000
```

(ix) List in alphabetical order all the customers having a loan at IIT Branch.

```
Select c_name
from Borrow
where b_name = 'IIT'
order by c_name;
```

avg, min, max, sum, count

(x) Find the average account balance at all branches.

```
Select b_name, avg(balance)
from Deposit
group by b_name.
```

(xi) Find the average account balance at all branches having average balance greater than 10000.

```
Select b_name, avg(balance).
from deposit
group by b_name
having avg(balance) > 10000
```

(xii) Find no. of tuples in customer relation.

```
Select count(*)
from Customer
```

(xiii) Find all customers who have deposit at IIT Branch but for whom there is no entry in the customer relation.

~~Select c_name~~
~~from Deposit~~
~~where b_name = 'IIT'~~
~~and c_name not in~~

```
Select c_name
from Deposit D
where b_name = 'IIT'
and not exists
    Select *
    from customer C
    where D.c_name = C.c_name
```

Example : Teacher (T_name, Dept, Tel_no, S_title)

Student (S_name, ~~cours~~ course, hall)

Study (S_~~title~~, s_name, level, status, marks)

Find the name of teachers of Maths department who teach 200 level subjects.

```
select T_name
from Teacher, Study
where Dept = 'Maths'
and level = '200'
and Teacher.S_title = Study.s_title
```

Find students of CSDP course living in VS or SN hall who study no subject ~~no~~ taught by professor XYZ.

```
Select s_name
from student
where course = 'CSDP' and
(hall = 'VS' or hall = 'SN')
and s_name not in
(select s_name
 from Teacher T, Study S
 where t_name = 'XYZ'
 and T.s_title = S.s_title)
```

QWEL

range of t_1 is R_1

range of t_2 is R_2

:

range of t_n is R_n

retrieve $(t_{i_1}.A_{j_1}, t_{i_2}.A_{j_2}, \ldots, t_{i_m}.A_{j_m})$

where P;

Q. Find all customers having an account at IIT Branch.

range of t is Deposit
retrieve (t.c_name)
where t.b_name = 'IIT')

Q. Find all customers having a loan at IIT Branch and their cities.

range of t is Borrow
range of u is Customer
retrieve (t.c_name, u.c_city)
where t.c_name = u.c_name
and t.b_name = 'IIT'

Q. Find all customers who have loan and account at IIT Branch.

```
range of t is Borrow,
range of u is Deposit,
retrieve (t.c_name)
where t.c_name = u.c_name and
      t.b_name = 'IIT' and
      tl.b_name = 'IIT'
```

Q. Find all customers who have account or loan or both at IIT Branch.

```
range of u is Deposit
retrieve into temp (u.c_name)
where u.b_name = 'IIT'
range of s is Borrow
append to temp(s.c_name)
where s.b_name = 'IIT'
```

Printing:

```
range of t is temp,
retrieve unique (t.c_name)
```

Q. Find all customers who have account at IIT Branch but dont have loan at IIT Branch.

```
range of u is Deposit
retrieve into temp (u.c_name)
```

```
range of t is temp
delete (t)
where t.c_name = s.c_name and
      s.b_name = 'IIT'
```

Printing :

```
range of t is temp
retrieve unique (t.c_name)
```

Q. Find all customer names in alphabetical order who have account at IIT Branch.

```
range of u is Deposit
retrieve u.c_name
where u.b_name = 'IIT'
sort by u.c_name
```

max min avg sum count → aggregate operators

Q. Find the average account balance for all accounts at 'IIT' Branch.

```
range of t is Deposit
retrieve avg (t.balance where t.b_name
                = 'IIT')
```

Q. Find all account numbers whose balance is more than the average balance at the branch where the account is held.

```
range of t is Deposit
retrieve (t.acc_no)
where t.balance > avg (t.balance by t.b_name)
```

8. Find all customers who have account at all branches in the city 'KGP'.

```
range of t is Deposit
range of u is Branch
range of s is Deposit
retrieve (t.c_name)
where
countu (s.b_name where
        s.b_name = u.b_name and
        u.b_city = 'KGP' and
        s.c_name = t.c_name)
=
countu (u.b_name
        where u.b_city = 'KGP')
```

Q. Find the name of subjects, such that all students studying the subject secure more than 80 marks.

```
range of t is study
retrieve (t.s_title)
where min(t.marks by t.s_title, t.course > 80)
```

Q. Find the students of CSDP, living in VS Hall or in SN Hall who study no subject taught by professor XYZ.

```
range of t is student,
append to temp1 (t.s_name)
where (t.course = 'CSDP' and (t.hall = 'VS'
                              or t.hall = 'SN'))

range of u is teacher
range of s is study
append to temp2 (s.s_name)
where u.s_title = s.s_title
and u.t_name = 'XYZ'.
```

Print :

```
range of t1 is temp1
range of t2 is temp2
delete(t1)
where t1.s_name = t2.s_name.
```

range of t3 is temp1
retrieve unique (t3.s_name)

Q. Find the maximum marks scored in each subject of the CSDP course.

range of t is study
range of u is student
retrieve t.s_title, max (t.marks by t.s_title)
where t.s_name = u.s_name
and u.course = 'CSDP'

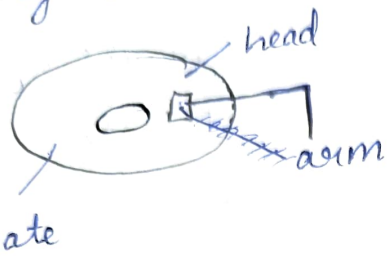
Q. Find the name of students who secured equal marks in 2 different subjects

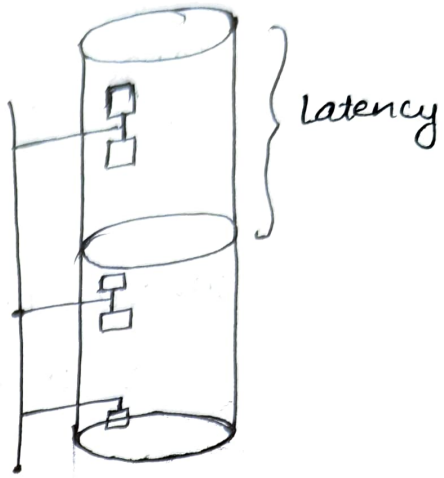
range of t is study
range of u is study
retrieve (t.s_name)
where t.s_name = u.s_name
and t.s_title < > u.s_title
and t.marks = u.marks.

FILE ORGANISATION

1 Block = 2400 Bytes

Data is transferred from primary to secondary memory block-wise.





Disk Latency time.

Seek time: It is the time it takes to move the arm to the correct cylinder.

→ Once the arm is placed on the correct cylinder, it takes some time to rotate and read the required data.

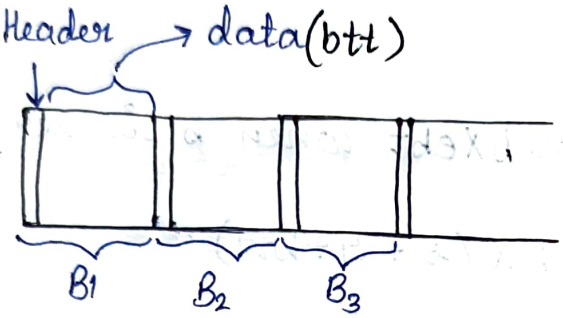
Average seek time(s) is the time it takes to traverse 1/3rd of the cylinder.

Average rotational latency time(r)

It is the average time it takes to locate the correct block on the track once the head is placed in the track. = $\frac{1}{2}$ revolution time

Block transfer time (btt)

btt is the time it takes to transfer the information of one block from these 2 memories once the head has been correctly placed at the beginning of the block.



Effective Block Transfer Time (EBT): Time taken to transfer the information of one block along with the header.

$$ebt = \frac{B}{b_1}$$

Parameters for IBM 3980

B	2400 bytes
btt	0.8 ms
C	600
ebt	0.84 ms
N	885
r	8.3 ms
s	16 ms,

Read 10 adjacent blocks

time = s + r + 10 × ebt

= 32.7 ms.

Random access of 10 blocks

time = 10(s + r + btt)

= 251 ms.

Sequential access time ≈ b × ebt when b is very large

Random access time ≈ b × (s + r + btt)

Heap File Organization

T_F = time to fetch a record

T_N = time to fetch next record

T_I = time to insert new record

T_W = time to update a record

T_D = time to delete a record

T_X = time for exhaustive reading of a file

T_Y = time to reorganize a file.

b blocks: $T_F = \frac{b}{2} \times ebt$

Example: 100000 records of 400 bytes each

b = 16,667

T_F = 7 sec.

fetch 10000 records

time ≈ 19 hours

$T_N = T_F$

Time to insert new record (T_I)

First the header will locate the block which will take time $s + r + btt$. Then the header will wait for the start to come.

$$T_I = s + r + btt + (2r - btt) + btt$$
$$= s + r + btt + 2r$$

Time to delete a record (T_D)

$$T_D = T_F + 2r$$

$T_X = b \times ebt$ (beginning to end) = 14 sec.

T_X (independent fetch of all records) $= n \times T_F = n \times \frac{b}{2} \times ebt$

$= 190$ hours.

~~T_Y~~ $n \rightarrow$ no. of active records

$Bfr \rightarrow$ no. of records per block.

$$T_Y = b \times ebt + \frac{n}{Bfr} \times ebt$$

Bucket

1 Bucket = n blocks, n = 1, 2, 3, ...

1 Bucket = 2 blocks.

time = s + r + btt + ebt

Random time = bk (s + r + dtt)

↓

no. of buckets

Value of dtt will change according to the number of blocks in a bucket.

Random time : bk(s + r + dtt) = bk (s+r) + bk X dtt

If bucket size increases, then the no. of buckets bk falls but dtt rises ∴ bk X dtt remains constant but bk (s+r) falls ∴ Total time falls.

Example : bk = 1 block = 2400 bytes

dtt = 0.8

Two block size = 2 blocks

dtt = 1.64

bk = 1250

1250 X 0.84 = 1050

bk = 625

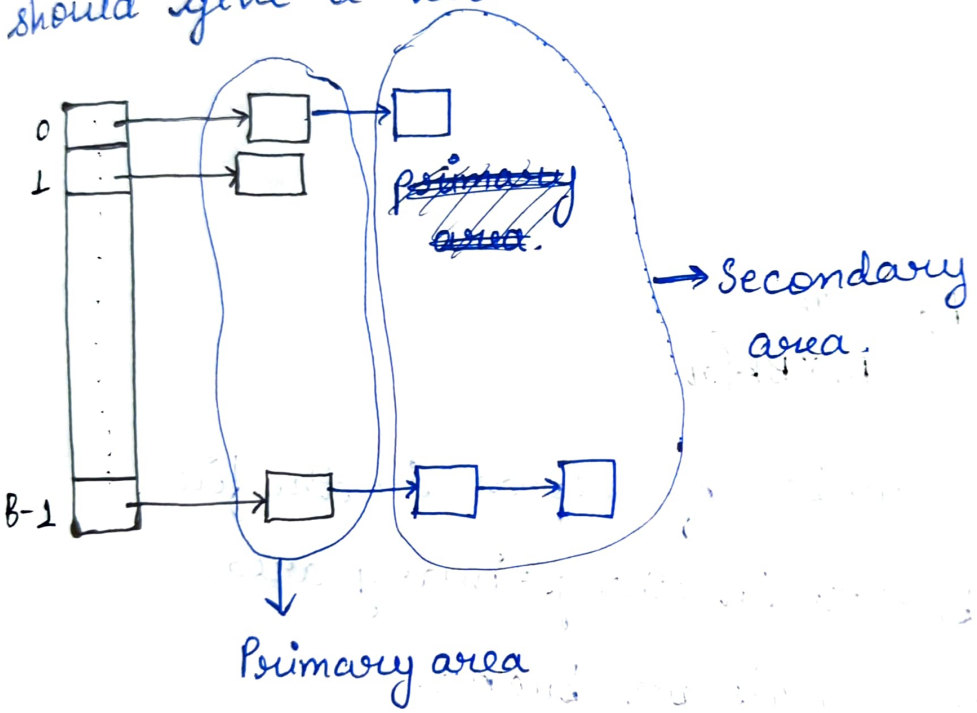
625 X 1.64 = 1025

For random access, the time is reduced significantly but for sequential access, the reduction is marginal.

*Hash File Organization

Records are stored according to the key value. Before making the file, the number of records are estimated.

If the expected number of records is m, ~~the~~ and the number of blocks reserved is n (0 to n-1). When a key value is supplied, the hash function should give a value between 0 and n-1.



When ~~inita~~ initially the file is formed, the records will be stored in the primary area and when the buckets are full, more records are stored in the secondary area.

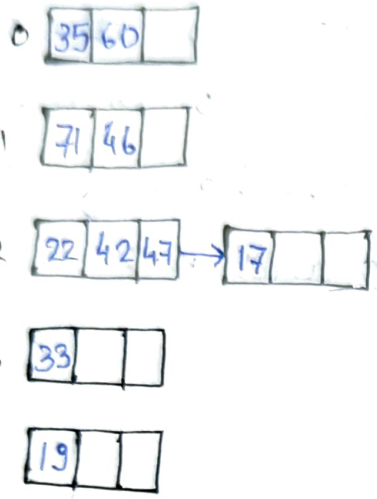
If there is secondary area, then at max the ~~whole~~ ~~ch~~ whole chain at that particular position has to be searched.

Example : 22, 35, 33, 42, 71, 60, 47, 19, 46, 17

$h(x) = x \bmod 5$

No. of buckets = 5

No. of records per bucket = 3.



Load factor : $L_f = \frac{n}{M \times Bkfr}$

$n \rightarrow$ Expected number of records in the file

$M \rightarrow$ No. of buckets in the primary area.

$Bkfr \rightarrow$ No. of records per bucket.

$T_f(\text{no chain}) = s + r + dtt$

(i.e. when address of the bucket is known)

$T_f(\text{successful}) = (s + r + dtt) + \frac{x}{2}(s + r + dtt)$

In the chain, in secondary area, there are x number of buckets and in the primary area, one bucket.

When chain is created, records are alloted the slots which are free, ∴ they are not in a sequential order.

Time for fetching the first bucket is $s+r+dtt$ and then in secondary area, average time will be $\frac{x}{2}(s+r+dtt)$.

$$T_F(\text{unsuccessful}) = s+r+dtt + x(s+r+dtt).$$

$$T_I = T_F(\text{unsuccessful}) + 2r$$

First last block of the chain will be fetched and inserted in main memory but then it has to be rewritten, ∴ the head will have to wait for $2r$.

$$T_D = T_F(\text{successful}) + 2r$$

For deletion, it is considered, that the record is present in the file, ∴ first it is found and then deleted.

T_U = T_D ; if key value is not changed

(Updation)

$T_D + T_I$; if key value is changed.

For sequential organization, heap file organizati is recommended.

* And for independent fetching of records, has file organization is recommended.

$$T_x = n \times T_f$$

In this case, the buckets are almost independen ∴ more suitable for random access.

* Extendible Hashing

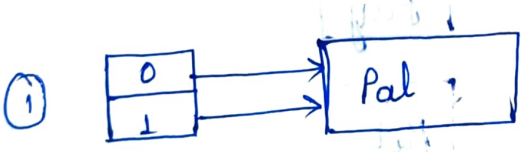
Initially no buckets are reserved. Buckets are created or removed according to need.

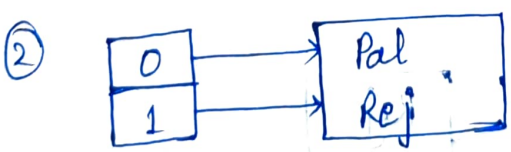
* Binary values of keys are considered. If the maximum number of bits is b, then i bits are considered for hashing.

Name	hCName (hash values)
⑤ Bose	00101101
⑥ Chandra	1101 0101
③ Dey	1010 0011
⑦ Mathur	1000 0111
① Pal	1111 0001
④ Ray	1011 0101
② Rej	0101 1000

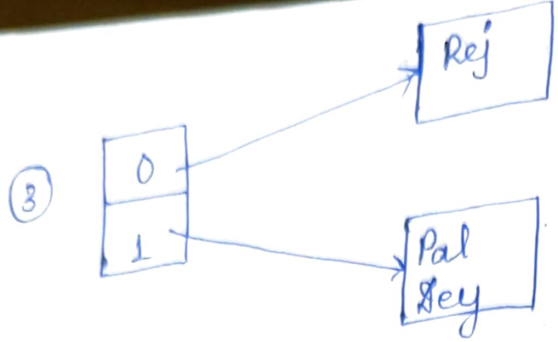
Each bucket can ~~store~~ atmost 2 ~~rec~~ records.

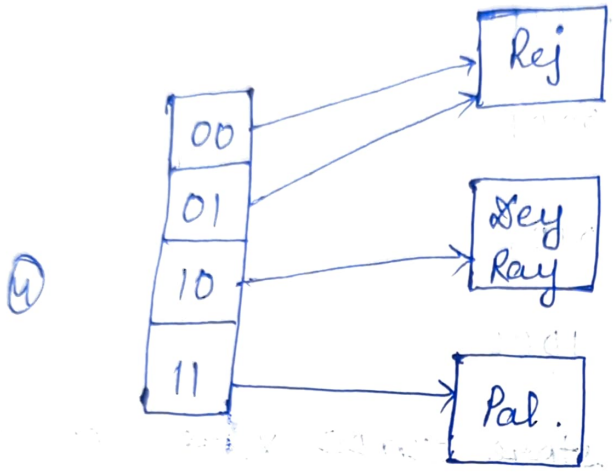
Let first record be Pal. Only first bit is considered. ∴ If the first bit is either 0, or 1, it will point to the same bucket as there is only one bucket.

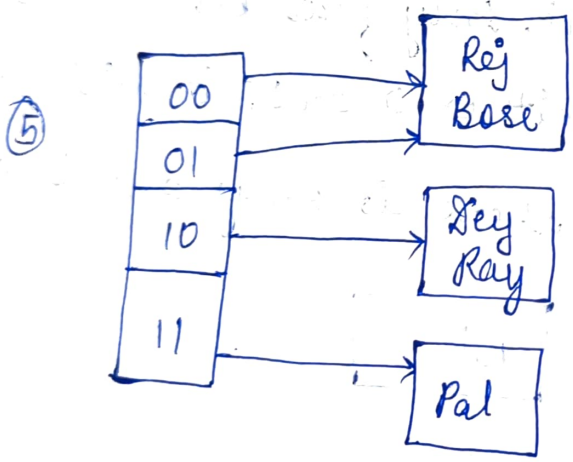


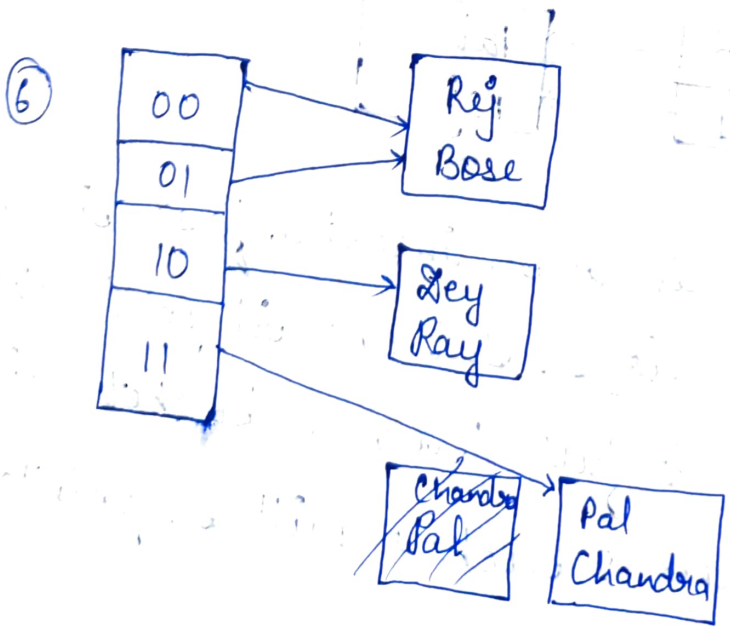


~~When~~ when Dey is to be inserted, first bit will be checked which is 1. But one bit points to a bucket which is full, ∴ a new bucket will be created. Now the 2 buckets will be pointed by 2 bits.

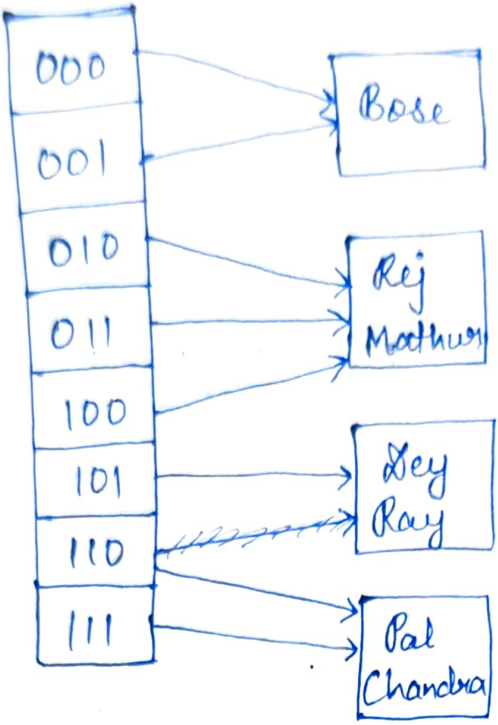

3
0
1
Rej
Pal
Dey


4
00
01
10
11
Rej
Dey
Ray
Pal.


5
00
01
10
11
Rej
Bose
Dey
Ray
Pal

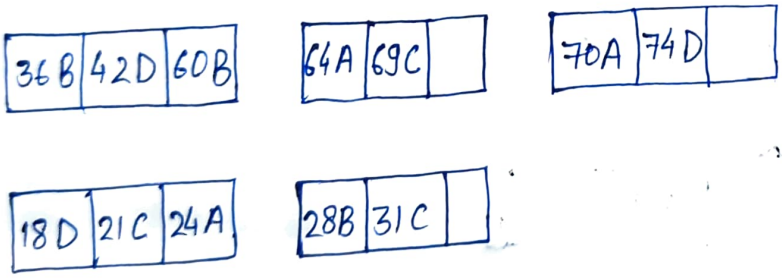

6
00
01
10
11
Rej
Bose
Dey
Ray
Chandra
Pal
Pal
Chandra

(7)

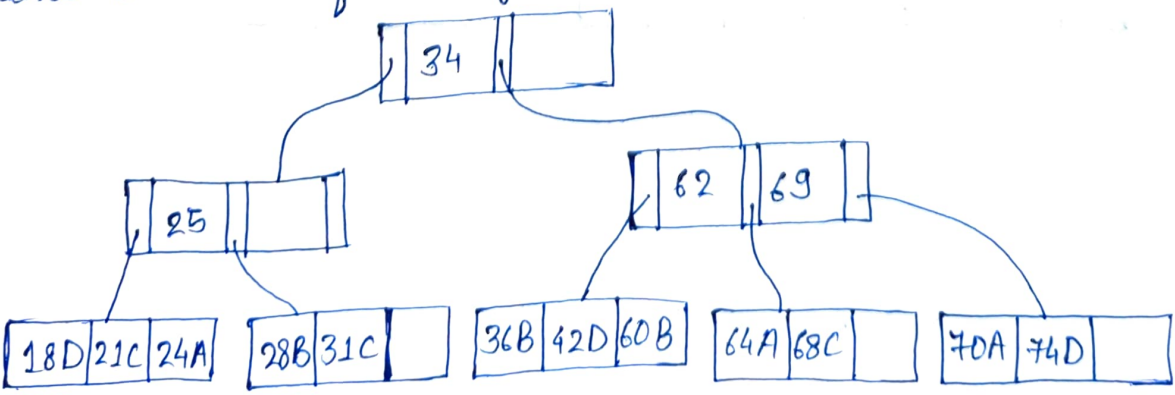


* B^+ Trees

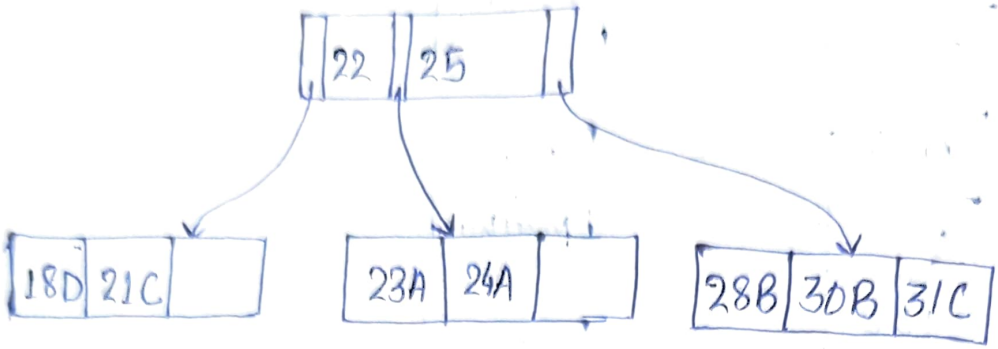
Let Roll No. and grade are being recorded. Let there be 2-3 records per leaf.



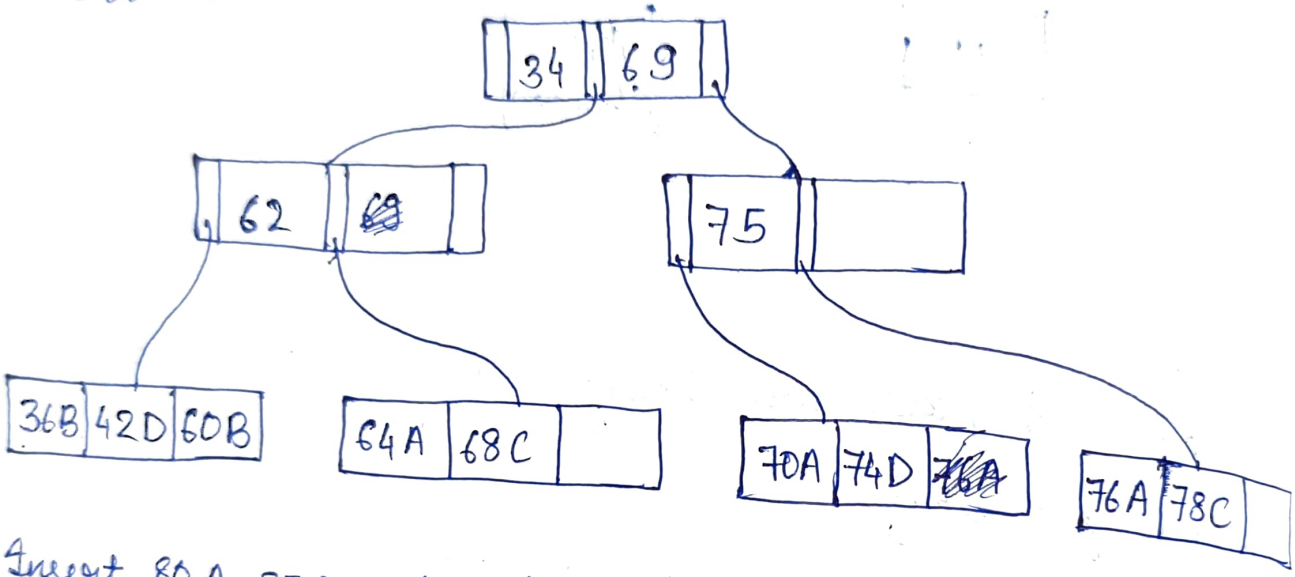
Leaves store data and non-leaves store pointers. It is not necessary that leaves and non-leaves have same number of data points and pointers.



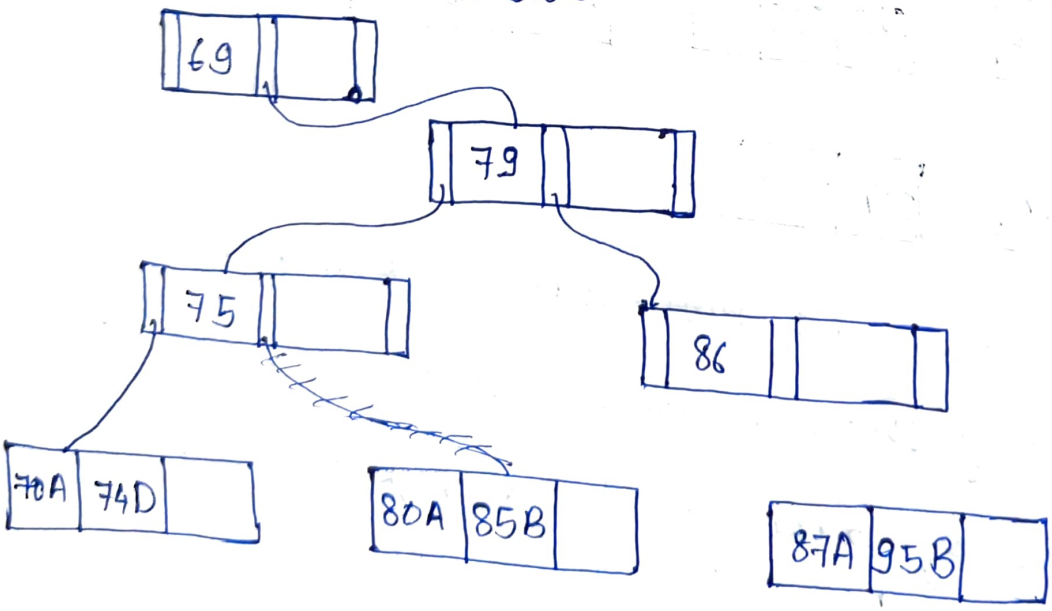
9) 22A and 30B are inserted.



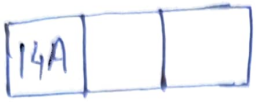
Insert 76A and 78C



Insert 80A, 85B, 87A and 95B.



14 A



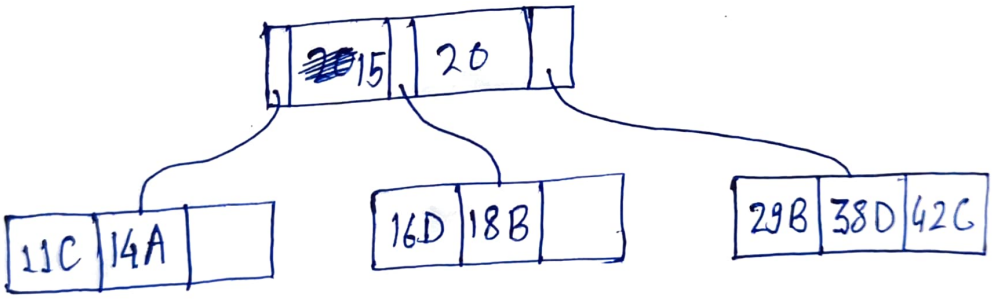
29 B

11 C

38 D

18 B

Inserting 16 D,



Let block size 800

Let record size 60

$\frac{800}{60} = 13$ (No. of leaves)

Key size 20

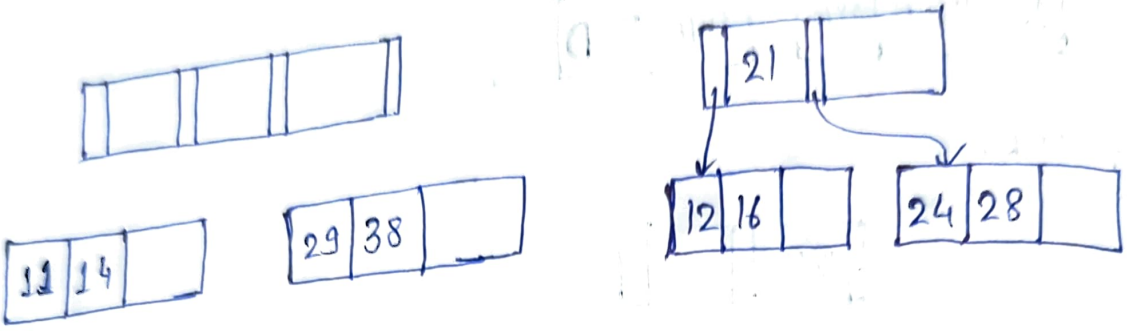
Pointer size 10

$n10 + (n-1)20 \leq 800 \quad n = 27$

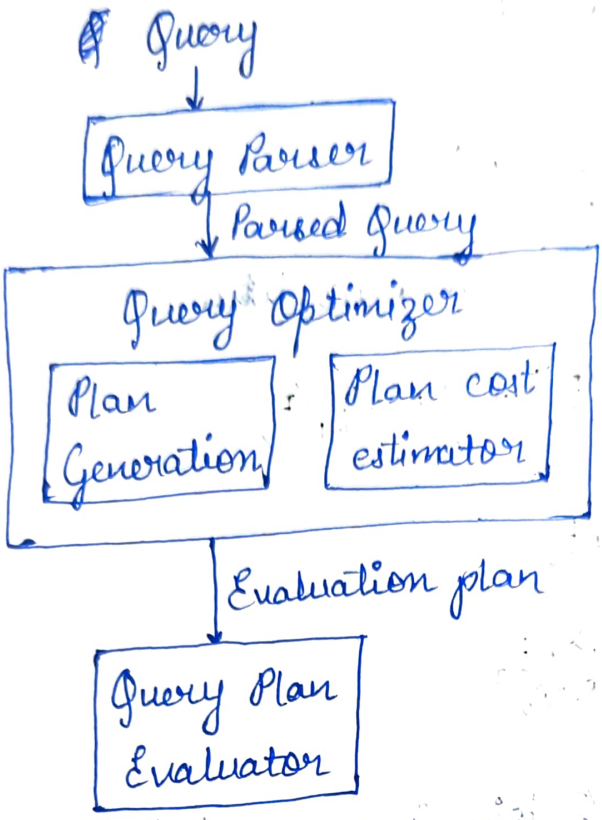
2/3 records in a leaf except root

3/4/5 ~~no~~ pointers in non-leaf except root

12, 16, 24, 28, 32, 20.



Query Optimization



Query Parser : It checks whether the syntax of the query is correct or not and also checks if the attributes and relations are correct.

It converts the SQL query into relational algebra expression in the form of a tree.

Plan generation : It transforms the query into different equivalent forms.

Plan cost estimator : estimates the cost of each plan and chooses the plan with the optimum cost.

★ Equivalence Rules

1. $\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$

2. $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$

Selection operation is commutative..

3. $\Pi_{L_1}(\Pi_{L_2} \cdots (\Pi_{L_n}(E)) \cdots)) = \Pi_{L_1}(E)$

4. $\sigma_{\theta}(E_1 \times E_2) = E_1 \underset{\theta}{\bowtie} E_2$

$\sigma_{\theta_1}(E_1 \underset{\theta_2}{\bowtie} E_2) = E_1 \underset{\theta_1 \wedge \theta_2}{\bowtie} E_2$

5. $E_1 \underset{\theta}{\bowtie} E_2 = E_2 \underset{\theta}{\bowtie} E_1$ θ join is commutative.

6. $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$

Natural join is associative.

7. $\sigma_{\theta_1}(E_1 \underset{\theta_2}{\bowtie} E_2) = (\sigma_{\theta_1}(E_1)) \underset{\theta_2}{\bowtie} E_2$

$\sigma_{\theta_1 \wedge \theta_2}(E_1 \underset{\theta_2}{\bowtie} E_2) = (\sigma_{\theta_1}(E_1)) \bowtie (\sigma_{\theta_2}(E_2))$

when θ_1 involves only E_1 and θ_2 involves only E_2.

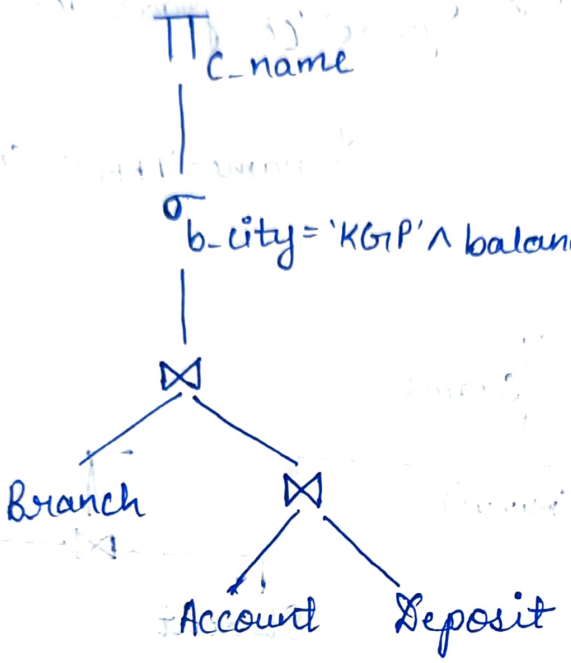
8. $E_1 \cup E_2 = E_2 \cap E_1$

Union and intersection are commutative and associative.

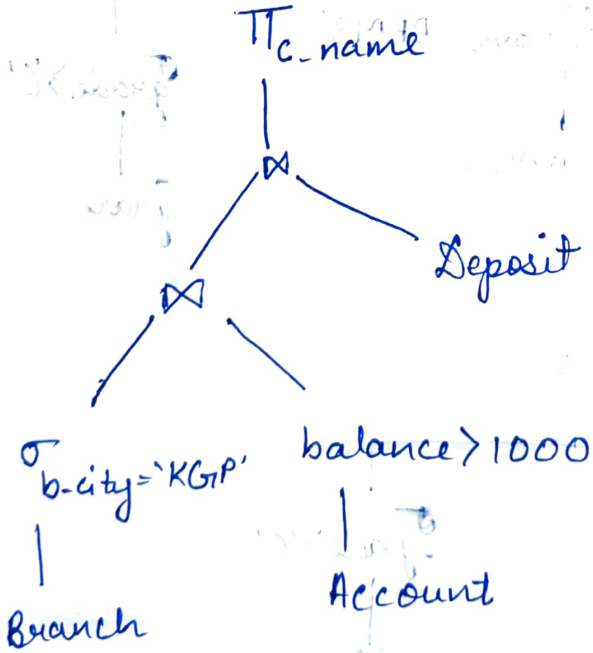
9. $\sigma_{\theta}(E_1 - E_2) = \sigma_{\theta}(E_1) - \sigma_{\theta}(E_2)$

10. $\Pi_L(E_1 \cup E_2) = \Pi_L(E_1) \cup \Pi_L(E_2)$

Example: $\Pi_{c\_name}(\sigma_{b\_city='KGP' \wedge balance>1000}(Branch \bowtie (Account \bowtie Deposit))$

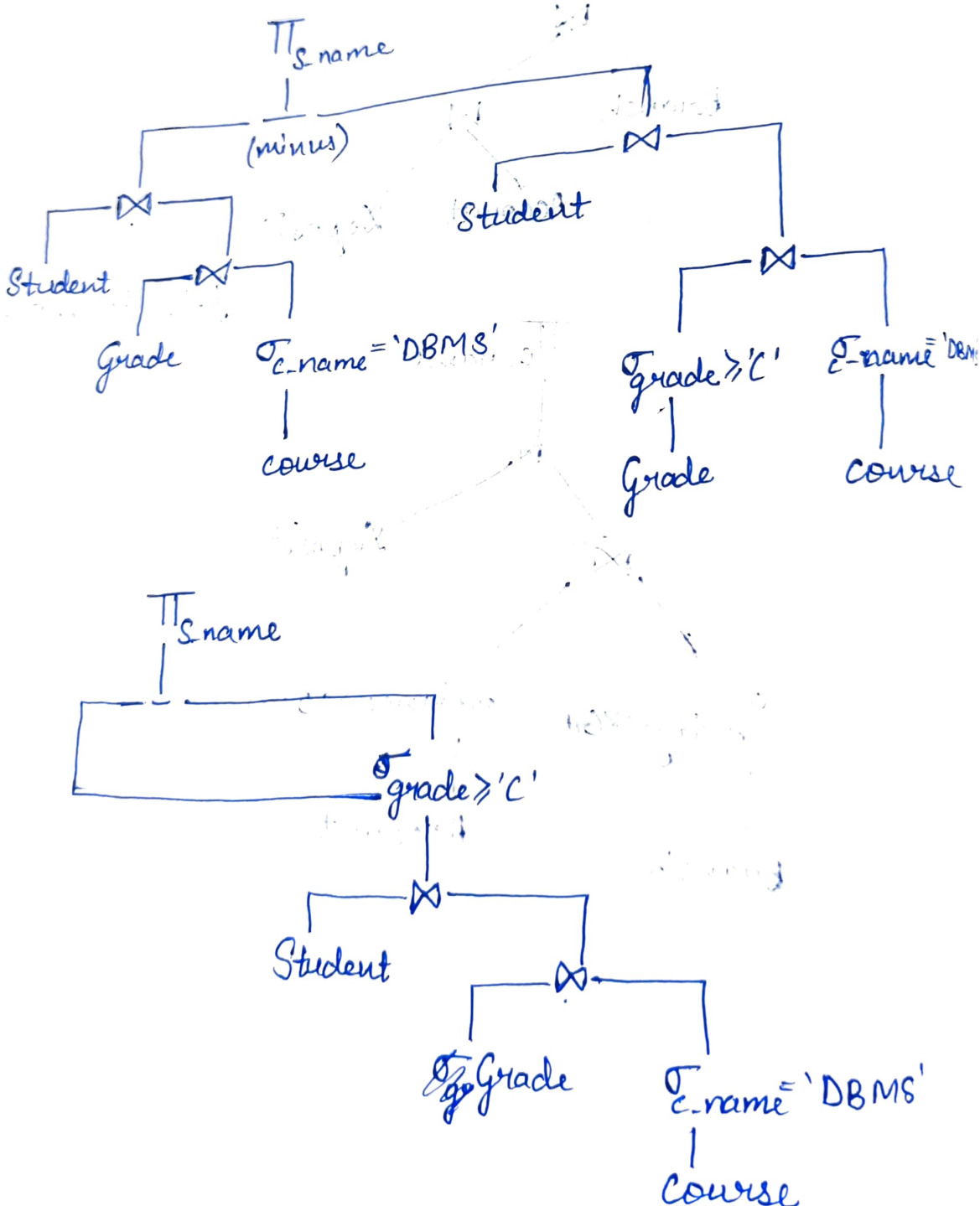


Join is moved up.

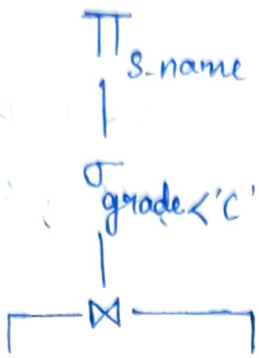


Example: Find the list of students who have not obtained grade 'C' or higher in 'DBMS' course.

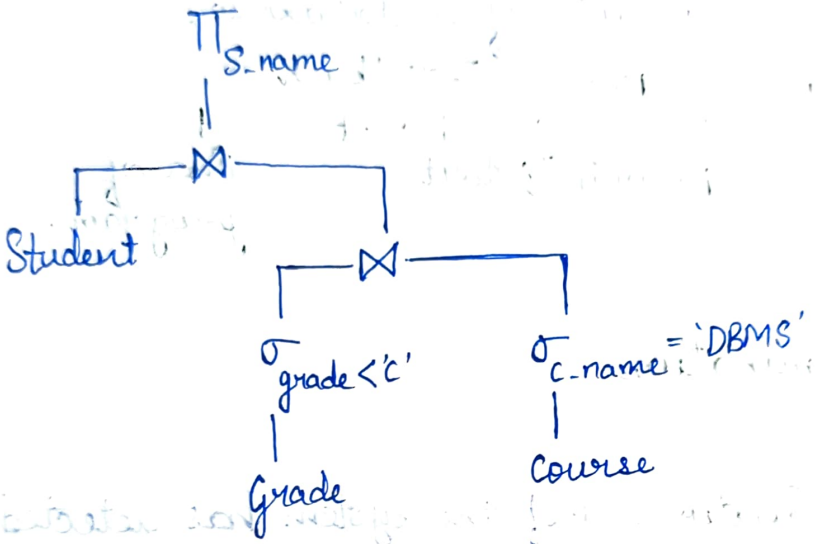
$$\Pi_{S\_name}(Student \bowtie (Grade \bowtie (\sigma_{C\_name='DBMS'}(course))) - (Student \bowtie (\sigma_{grade \geq 'C'}(Grade) \bowtie \sigma_{C\_name='DBMS'}(course)))$$



$R - \sigma_C(R) = \sigma_{\neg C}(R)$



(Now this selection will move down the tree)



★ Recovery

Backups have to be made in order to bring the data back to a previous stage at which data was correct in the event of failure.

• Recovery Schemes

Forward Error Recovery

In this scheme when a particular error in the system is detected, the recovery system makes an accurate assessment of the state of the system and then makes appropriate adjustments based on anticipated

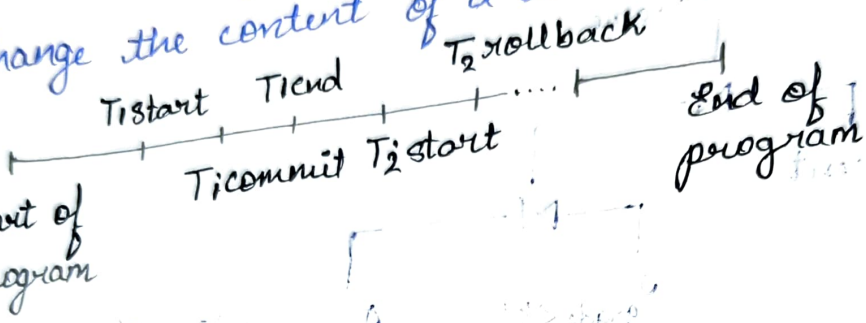
result had the system being error-free.

Backward Error Recovery

In this scheme the system is reset to some previous correct state which is free from errors.

- Transaction

Transaction is a problem unit whose execution may change the content of a database:



Successful termination

Suicidal termination: If the system has detected some error, and the system has rolled back that transaction

★ Properties of a transaction

ACID Test

- Atomicity proper

This property of a transaction implies that it will run to completion as an indivisible unit at the end of which no changes have occured to the database or the database has been changed in a consistent manner.

Consistency property

This property implies that if the database was in a consistent state before the start of the transaction then on the termination of the transaction, the database will be in a consistent state.

Isolation property

This property indicates that the actions performed by the transaction will be isolated or written on outside of the ~~transaction~~ transaction until the transaction is complete.

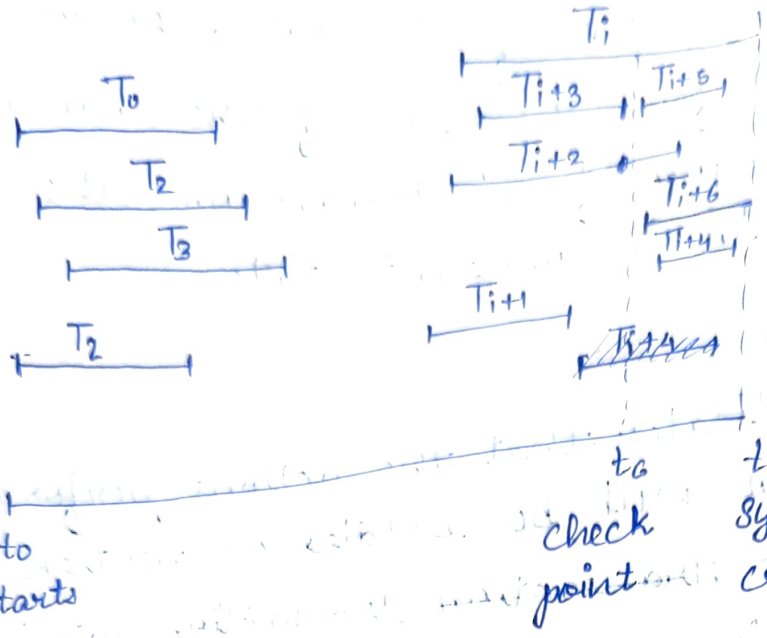
Durability property

This properties ensures that the commit action of a trnx on its termination will be reflected on the database.

*Recovery in Controlled DBMS

Log : It is a file in which all the transactions performed on the database is written. Trnx start, commit and end markers are recorded.

- Write ahead log-strategy



~~when~~ ~~the~~ when the system crashes only transactions T_i and T_{i+6} are getting executed.

Checkpoint : Database is written to stable storage.

Transaction-consistent check point

Here the transactions that are active when system timer ~~out~~ signals a check point, ~~active~~ are allowed to complete but no new transactions are allowed to be started until the checkpoint has completed.

Action consistent check point.

Active transactions are allowed to complete the current state before the checkpoint and no new actions can be started on the database until the checkpoint is complete.

Transaction-oriented check point

Take a checkpoint at the end of each transaction by forcing the log of the transaction on stable storage.

When the system crashes, transactions T_{i*}, T_{i+2}, T_{i+4}, T_{i+5} and T_{i+6} transactions should be UNDO.

UNDO (T_i, T_{i+2}, T_{i+4}, T_{i+5}, T_{i+6})

⇓

REDO(T_{i+5}, T_{i+4}, T_{i+2}).

UNDO (T_i, T_{i+6})

Do, undo, redo

A transaction on the current database transforms it from current state to new state. This is called Do operation.

Undo operation comes into picture when a trnx decides to terminate itself.

Trnx redo involves performing the changes made by a transaction that committed before a system crash. With a write-ahead log strategy, a committed trnx implies that the log for the trnx would be written to the secondary storage but the database

may or maynot have been identified before the system failure has been identified.

★ Concurrency Management

- **Schedule**

The order in which the statement of transaction are executed is known as schedule.

T_1

1. Read (Aug_F_sal)
2. Aug_F_sal = f_1(Aug_F_sal)
3. write (Aug_F_sal)

T_2

4. Read (Aug_F_Salary)
5. Aug_S_sal = f_2(Aug_S_Sal)
6. write (Aug_S_sal)

S_1

1 2 3 4 5 6

S_2

4 5 6 1 2 3

S_3

1 4 2 5 3 6

T_3

1. Read (A)
2. A = A + 10
3. write (A)

T_4

4. Read (A)
5. A = A * 1.1
6. write (A)

A = 200

S_1

T_1 T_2

A = 231

S_2

T_2 T_1

A = 230

S_3

4 5 1 2 6 3

T_5	T_6
1. Read (A)	7. sum = 0
2. A = A - 100	8. Read (A)
3. write (A)	9. sum = sum + A
4. Read (B)	10. Read (B)
5. B = B + 100	11. sum = sum + B
6. write (B)	12. write (sum)

A = 500
B = 1100

1 7 8 2 3 9 4 5 6 10 11 12

Inconsistent read problem : When ~~transaction~~ system allows to read a variable 2 times, then the 2nd read statement might be inconsistent as it does not

Given a set of n transactions, the schedule will be serialisable if following conditions hold.

(i) All transactions are correct in the sense that if any one of the transactions is executed by itself on a consistent database. The result ~~must be~~ databases will be consistent.

(ii) Any serial execution of the trnx is also correct & preserve the consistency of the database. The result obtained is also correct.

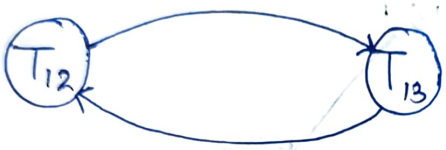
A directed edge from node T_i to T_j indicates one of the following:

1. T_i performs the operation write(A) then T_j performs the operation Read(A).

2. T_j performs the operation write(A) then T_i performs the operation read(A).

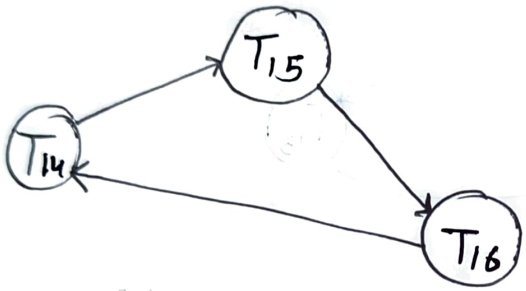
Schedule	T_9	T_{10}	T_{11}
Read(A)	✓		
$A = f_1(A)$	✓		
write(A)	✓		
Read(A)	.....	✓	
$A = f_2(A)$		✓	
write(A)		✓	
Read(B)		✓	
$A = f_3(B)$		✓	
write(B)		✓	
Read(B)			✓
$B = f_4(B)$			✓
write(B)			✓

Schedule	T_{12}	T_{13}
Read (A)	✓	
$A = f_1(A)$	✓	
Read (A)		✓
$A = f_2(A)$		✓
write (A)		✓
write (A)	✓	



S	T_{14}	T_{15}	T_{16}
Read (A)	✓		
Read (B)		✓	
$A = f_1(A)$	✓		
Read (C)			✓
$B = f_2(B)$		✓	
write (B)		✓	
$C = f_3(C)$			✓
write (C)			✓
write (A)	✓		
Read (B)			✓
Read (A)		✓	

S	T_{14}	T_{15}	T_{16}
$A = f_4(A)$		✓	
Read(C)	✓		
write(A)		✓	
$C = f_5(C)$	✓		
write(C)	✓		
$B = f_6(B)$			✓
write(B)			✓



A Locking Scheme

- Lock : Lock is a variable associated with various data items.
- Exclusive lock

Whenever locked, transaction can write or update while no other transaction can even read (exclusive use to one item only).

- Shared Lock.

One can read data item & can work with it but can't modify it. Other transactions can also read.

	T_5	T_6
Lock+(sum)		✓
sum = 0		✓
locks(A)		✓
Read(A)		✓
sum = sum+A		✓
unlock(A)		✓
lockx(A)	✓	
Read(A)	✓	
A = A-100	✓	
write(A)	✓	
unlock(A)	✓	
lockx(B)	✓	
Read(B)	✓	
B = B+100	✓	
write(B)	✓	
unlock(B)	✓	
lockx(B)		✓

Read(B) ✓
Sum = sum + B ✓
write(sum) ✓
unlock(B) ✓
unlock(sum) ✓

- Two Phase Locking

It has 2 phases, a growing phase where the number of locks increases from 0 to a maximum value for the transaction and one contracting phase where the no. of locks held decreases from max! value to zero.

If a schedule follows 2 phase locking then the series is serializable.

Let there be $T_1, T_2, \ldots, T_x$ transactions and assume there is a cycle.

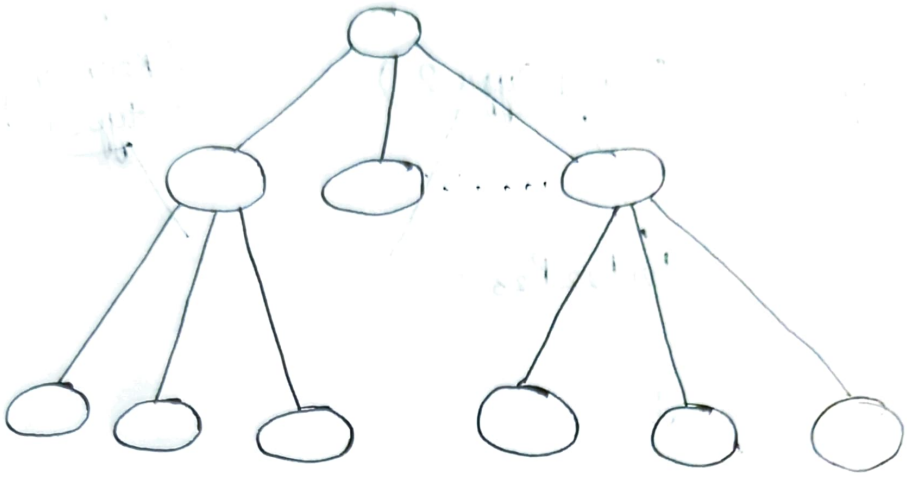
$$T_a \rightarrow T_b \rightarrow T_c \rightarrow \ldots \rightarrow T_x \rightarrow T_a$$

T_b's lock will be executed after T_a's unlock is performed. But after T_x, T_a cannot again lock the data items according to 2-phase locking.

Any data item in a database can be locked only once

• Intention Share (IS)

When a ~~data item~~ node is locked in IS mode, then the node or its descendants cannot be exclusively locked. The descendants may be locked in S or IS mode locked.

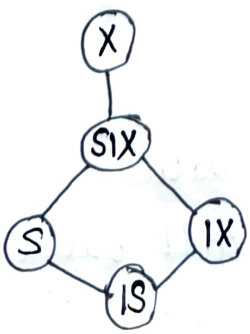


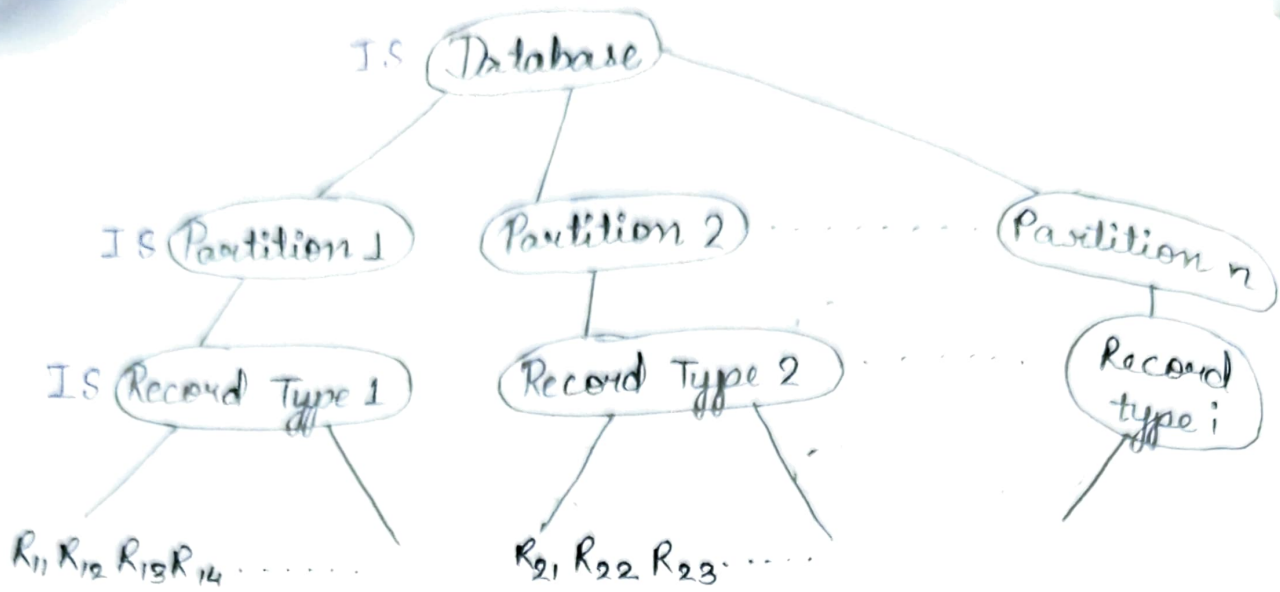
• Intention Exclusive (IX)

When a node is locked in IX mode then the node itself ~~be~~ cannot be exclusively locked. However, the descendants can be locked in any of the locking modes.

• Shared and Intention Exclusive (SIX)

The node locked in SIX mode and all the descendants ~~and all the descendants~~ will intrinsically locked in shared mode. Any of the descendants can be locked in X or SIX mode.





R_{13} → To be retrieved

R_{13} → S

R_{22} → update R_{22} → X

Time Stamp Based Ordering Protocol

When a trnx starts executing then it is assigned some value which is equal to the system clock value when the trnx started.

T_i → time stamp value of t_i

$T_i(t_i)$ & $T_j(t_j)$

s.t. $t_i < t_j$

T_i → Older transaction T_j → younger transaction

$X : \{x, wx, Rx\}$

x : value of X

wx : write timestamp value

Rx : It is the system call value when the dataitem was written for the last time.

R_x: Read timestamp value.

The system clock value when the transaction last read the data item

• Read operation

A trnx T_a with timestamp value t_a issues a read operation for the dataitem $X\{x, w_x, R_x\}$.

1) This request will be allowed if $t_a \geq w_x$.

After the o/p : $X\{x, w_x, t_a\}$

2) This request will fail if $t_a < w_x$.

T_a is an older trnx than the one which wrote the value of X.

• Write operation

A trnx T_a with timestamp value t_a issues a write operation for dataitem $X\{x, w_x, R_x\}$.

1) $t_a \geq w_x$ and $t_a \geq R_x$ then the write operation will be allowed. The trnxs which last read and wrote the value of X should be older than Ta. w_x will be replaced by t_a.

2) $t_a < R_x$: A longer trnx is already using the value of X. $\therefore T_a$ is not allowed to modify the value of X. T_a is rolled back and a new timestamp value is provided

3) $R_x \leq t_a < w_x$, younger trnx has already written the

value of X, and T_a is not allowed to write the value of X.

Schedule	T_{22} (t_{22})	T_{23} (t_{23})
1. Sum = 0	✓	
2. Read (A)	✓	
3. sum = sum + A	✓	
4. Sum = 0		✓
5. Read (A)		✓
6. A = A - 100		✓
7. write (A)		✓
8. Read (B)	✓	
9. Sum = sum + B	✓	
10. show (sum)	✓	
11. Sum = sum + A		✓
12. Read (B)		✓
13. B = B + 100		✓
14. write (B)		✓
15. Sum = sum + B		✓
16. ~~write~~ (sum) Show		✓

Initially, A: 400, w_a, R_a B: 500, w_b, R_b

Step(2): A: 400, w_a, t_{22} "

Step 5: A: 400, w_a, t_{23} "

Step 7: A: 300, t_{23}, t_{23} "

Step 8 : " B: 500, w_b, t_{22}

Step 10 : sum = 900

Step 12 : " B: 500, w_b, t_{23}

Step 14 : " B: 600, t_{23}, t_{23}

Schedule	T_{22}	T_{23}
1) sum = 0		✓
2) Read (A)		✓
3) A = A-100		✓
4) write (A)		✓
5) Read (A)	✓	
6) sum = sum+A		✓
7) Read (B)		✓
8) B = B+100		✓
9) write (B)		✓
10) sum = sum+B		✓
11) show (sum)		✓
12) sum = 0	✓	∉ T_{22} : t_{22}' (> t_{23})
13) Read (A)	✓	
14) sum = sum+A	✓	
15) Read (B)	✓	
16) sum = sum+B	✓	
17) show (sum)	✓	

A: 400, w_b, R_a B: 500, w_b, R_b

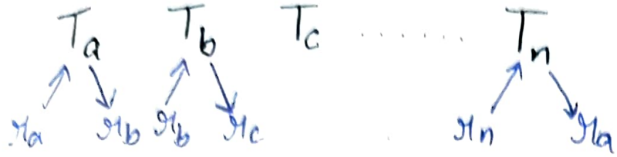
Step 2 A: 400, w_a, t_{23} "

Step 4 A: 300, t_{23}, t_{23} "

Step 5

Step 5 Read is not allowed therefore it will rollback and will be forced to be executed in the end serially.

*Deadlock and its Resolution

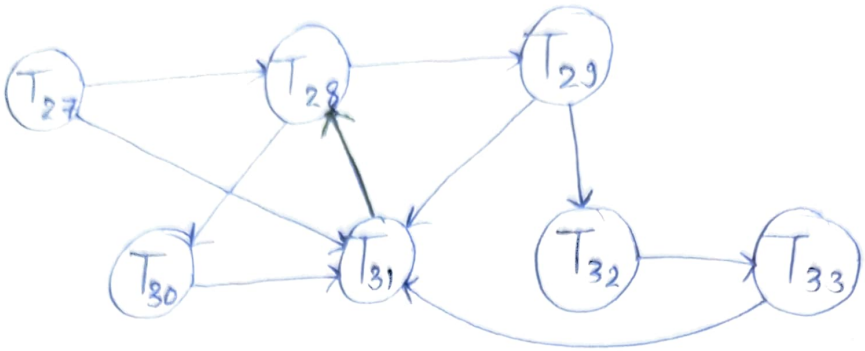


Wait-for graph

T_id	Items_locked	Items_waiting_for
T_{27}	B	C, A
T_{28}	C, M	H, G
T_{29}	H	D, E
T_{30}	G	A
T_{31}	A, E	C* (added later)
T_{32}	D, I	F
T_{33}	F	E

T_i is waiting for a dataitem currently allocated to and is being used by T_j, then there will be a directed edge from T_i to T_j.

If there is a cycle in the wait-for, then there is a *deadlock.



After adding c^* in the waiting list of T_{31}, then a cycle is created.

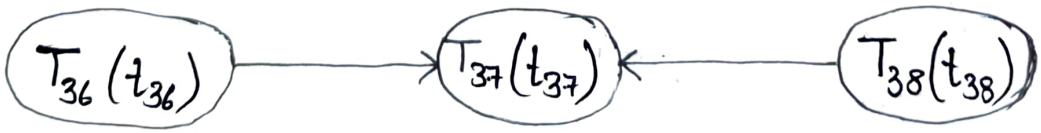
* How to prevent deadlock?

* Wait-Die Scheme

Wait and die scheme for deadlock prevention is as follows:

i) If the requesting trnx is older than the trnx that holds the lock on the requested data-item, the requesting trnx is allowed to wait.

ii) If the requesting trnx is younger than the trnx that holds the lock on the requested data-item, the requesting trnx is aborted and rolled back.



$$t_{36} < t_{37} < t_{38}$$

$T_{36} \rightarrow$ wait, $T_{38} \rightarrow$ abort (Roll-back)

★ Wound-Wait Scheme

i) If a younger trnx holds a data item requested by an older one, younger trnx will be aborted and rolled back.

ii) If a younger trnx requests a data item held by older trnx, the younger trnx is allowed to wait.

$t_{39} < t_{40}$

t_{40} → aborted and rolled back.

★ Distributed Database (DDB) {DDBMS}

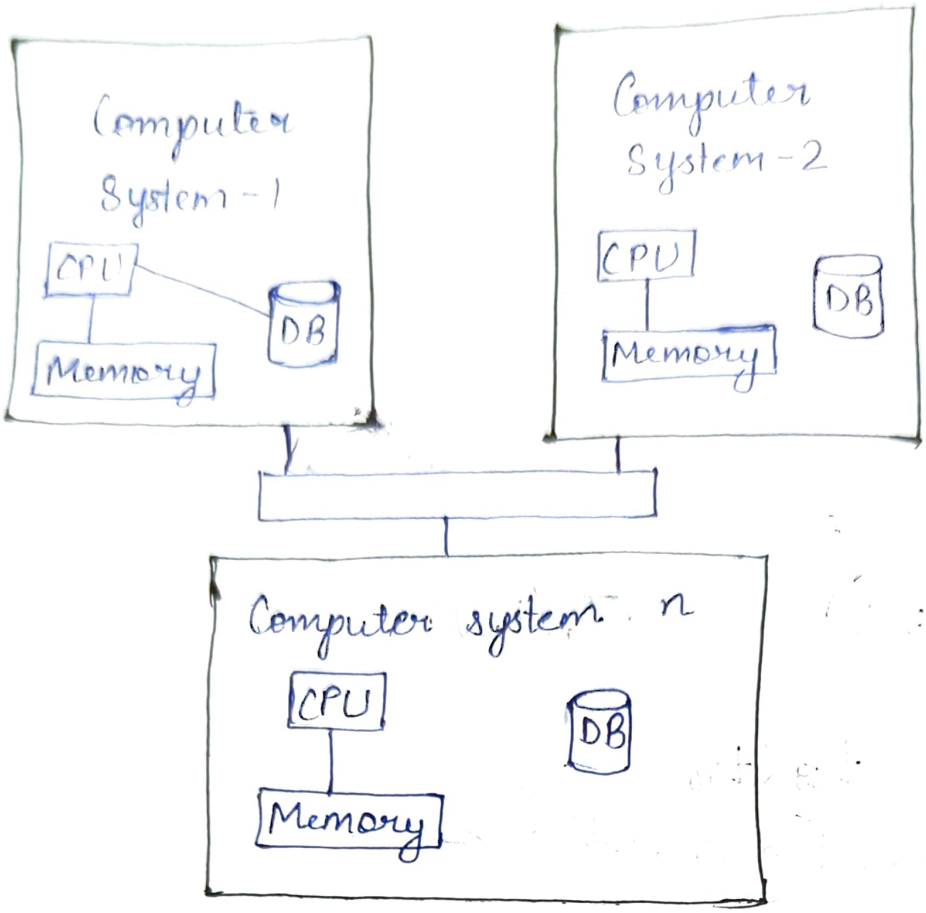
Database is stored at multiple locations

- Shared Memory Architecture

Multiple processors share the secondary disk as well as the primary memory also.

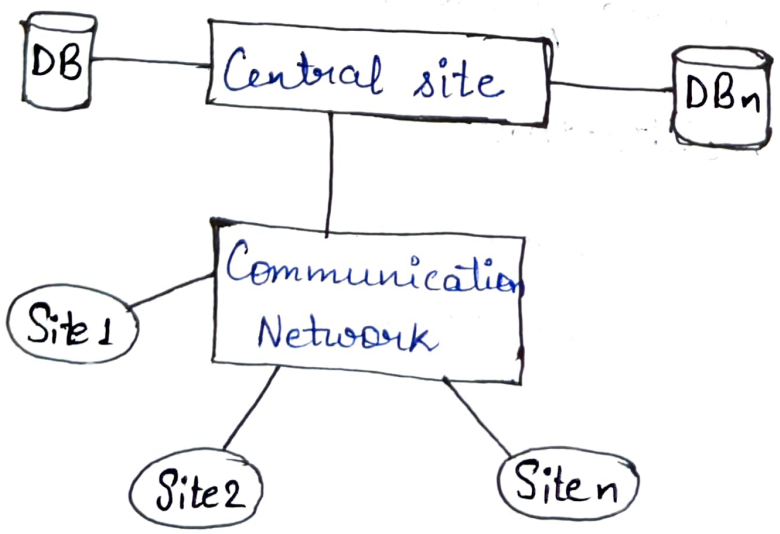
- Shared Disk architecture

Multiple processors share the same secondary disk but they have their own memory.



In the above architecture, all databases have their own primary and secondary memory but they are connected by network to share information.

- Centralized database.



• Distributed DB

Databases are distributed on different sites. When a query comes it is directed accordingly.

* Advantages

• Management of distributed Data with different levels of transparency

Distribution transparent

the ~~physical~~ ~~to~~ physical (memory) location of the file will not be known by the user.

• Distribution or network transparency

This refers to freedom for the user from the operational details of the network.

Location transparency: When we want to execute a command, it ~~to~~ should be independent of the location of the database.

Naming transparency: Once a name is specified, the name object can be accessed unambiguously without ~~addit~~ additional information.

- Replication transparency

It makes the user unaware of the existence of multiple copies.

Better availability: If some site is down, their data can be used from a copy in another site.

Performance ↑

Reliability

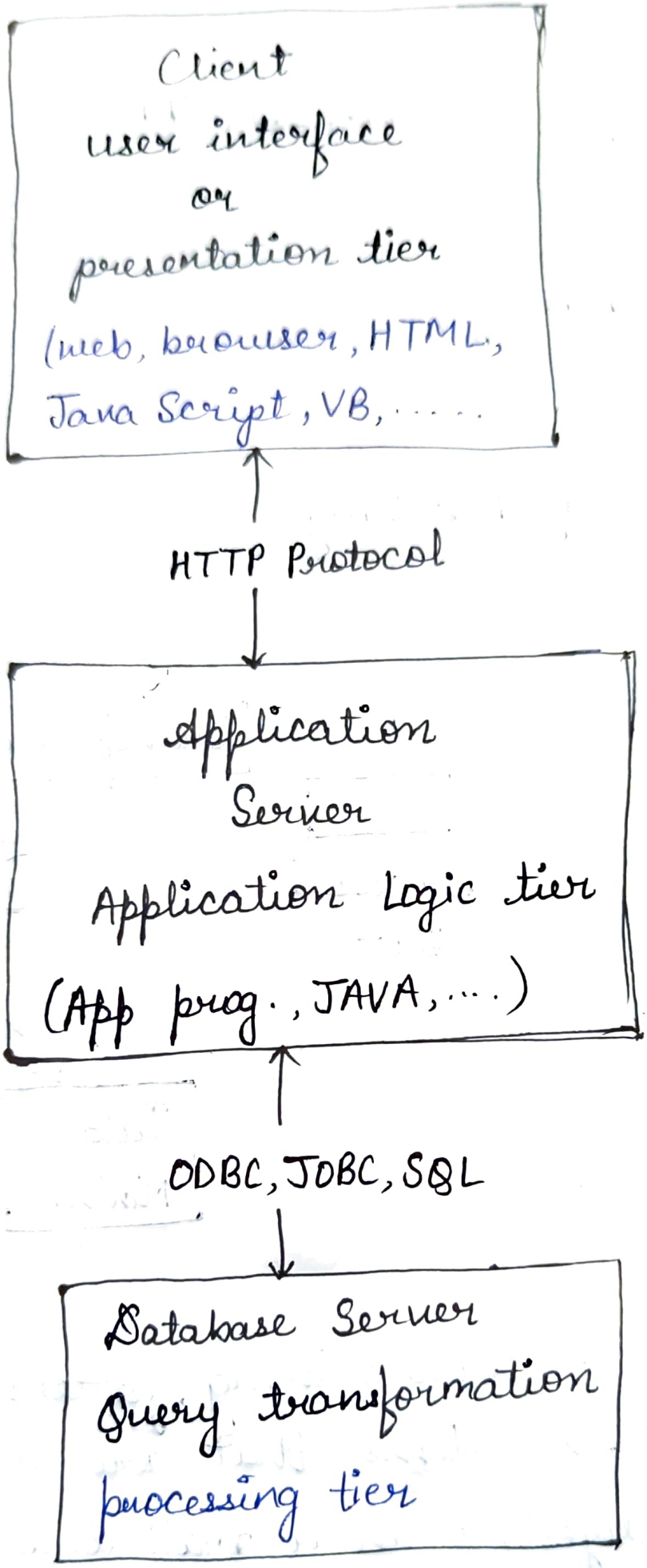
- Fragmentation transparency

Horizontal fragmentation: Database is fragmented row wise or record-wise.

Vertical fragmentation: Column-wise.

Mixed fragmentation: Both horizontal and vertical fragmentation.

- Increased reliability and availability
- ~~Reliability~~ Reliability is the property that a system is not down at a given time.
- Improved performance
- Easier expansion


Client
user interface
or
presentation tier
(web, browser, HTML,
Java Script, VB, ...
HTTP Protocol
Application
Server
Application Logic tier
(App prog., JAVA, ...)
ODBC, JOBC, SQL
Database Server
Query transformation
processing tier

★ Data Mining

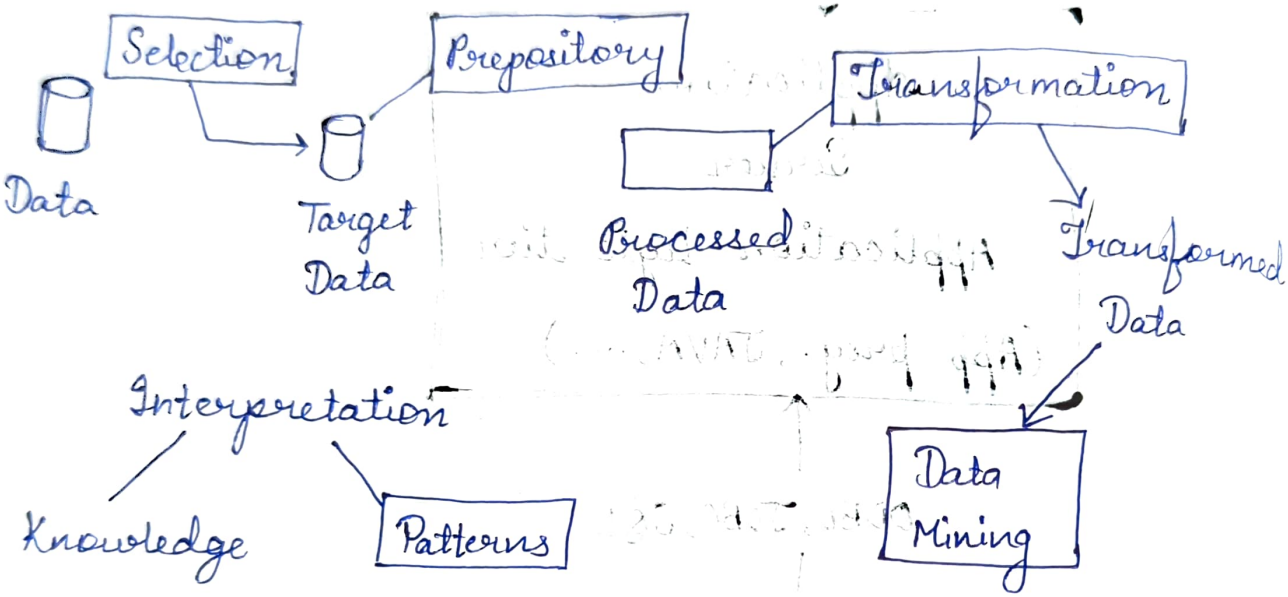
Knowledge Discovery in Databases (KDD)

In the ~~KDD~~ KDD process, there are 3 major parts:

★ ~~Association~~ Association Rule Mining

★ Classification

★ Clustering / Similarity Matching



In case of association rule mining, patterns are formed such that the items that are more likely to be purchased together in a mall are kept together based on history of purchases.

In case of classification, when new data is acquired using the previous classification how can new predictions be made in order to increase profit.
Example: New loans to be offered to new potential

customers.

In case of clustering, data points are classify assigned to different clusters.

TID	Item
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Item Set

Item set is a collection of one or more items.

K-item set

An item set which contains k items.

Support Count(σ)

Frequency of occurence of an item set in a set of transactions.

$$\sigma\{\text{Milk, Beer, Diaper}\} = 2$$

Support

Fraction of transactions that contain an item set.

$$s\{\text{Milk, Beer, Diaper}\} = \frac{2}{5}$$

Frequent Item Set

It is an item set whose support $\geq$ Minimum support threshold.

Association Rule discovery

$\{Diaper\} \rightarrow \{Beer\}$

$\{Milk, Bread\} \rightarrow \{Coke\}$.

→ Person who purchases milk and bread is also likely to purchase Coke.

Let there be d unique items

Total no. of itemset $= 2^d$

Total no. of possible association rules $= 3^d - 2^{d+1} + 1$

★ Apriori Algorithm

1. Calculate Minimum support.
2. Find out frequent itemset. (any subset of frequent itemset must be frequent).

∴ If a set is not frequent then any of its superset will also not be frequent.

L_k : Set of ~~candidate~~ k-itemset
~~frequent~~
Candidate

Join step:

C_k is generated by joining L_{k-1} with itself where C_k is the candidate itemset of size, L_k is the frequent itemset of size k.

Prune step: $(k-1)$

Any $k-1$ itemset that is not frequent cannot be a subset of frequent k itemset.

L_1 : {frequent items};

for $(k=1; L_k != \emptyset; k++)$ do begin

C_{k+1} = candidates generated from L_k;

for each transaction t in database DO

increment the count of all candidates in C_{k+1} that are contained in t

L_{k+1} = candidates in C_k which are frequent according to support.

Example :

TID	Items
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

min support = $\frac{2}{9} = 22\%$

Minimum confidence = 70%.

Pruning step: $(k-1)$

Any $k-1$ itemset that is not frequent cannot be a subset of frequent k itemset.

L_1 : {frequent items};

for $(k=1; L_k != \emptyset; k++)$ do begin

C_{k+1} = candidates generated from L_k;

for each transaction t in database

DO

increment the count of all candidates in C_{k+1} that are contained in t

L_{k+1} = candidates in C_k which are frequent according to support.

Example :

TID	Items
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1, I2, I3, I5
T9	I1, I2, I3

min support = $\frac{2}{9} = 22\%$

Minimum confidence = 70%

S1

IT	SC
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_1

IT	SC
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

S2 L_1 Join L_1

{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

C_2

{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

L_2

S3 L_2 Join L_2

{I1, I2, I3}	2
{I1, I2, I5}	2

C_3

{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

$\{I1, I2, I3\}$

$\{$

supersets of non-frequent sets of 2 items will not be frequent, ∴ we will not consider them.

(3) $C_4 : L_3$ Join L_3

$C_4 \neq \emptyset$

$\{I1, I2, I3, I5\}$ is not frequent because $\{I3, I5\}$ is not frequent.

$L = L_1 \cup L_2 \cup L_3$

$L = \{\{I1\}, \{I2\}, \{I3\}, \{I4\}, \{I5\}, \{I1, I2\}, \{I1, I3\}, \{I1, I5\}, \{I2, I3\}, \{I2, I4\}, \{I2, I5\}, \{I1, I2, I5\}, \{I1, I2, I5\}\}$

Association Rule:

I, s is subset of I

$S \rightarrow (I - S)$ if

$$\frac{\text{Support-count}(I)}{\text{Support-count}(S)} >= \text{min. confidence}$$

$\{I1, I2, I5\}$

$\{I1, I2\}, \{I1, I5\}, \{I2, I5\}, \{I1\}, \{I2\}, \{I5\}$

R1: $I1, I2 \rightarrow I5$ $\quad \frac{SC\{I1, I2, I5\}}{SC\{I1, I2\}} = 50\%$

Reject.

R_2 : I1, I5 → I2

$$\frac{SC\{I1, I2, I5\}}{SC\{I1, I5\}} = 100\%$$

R2 is selected