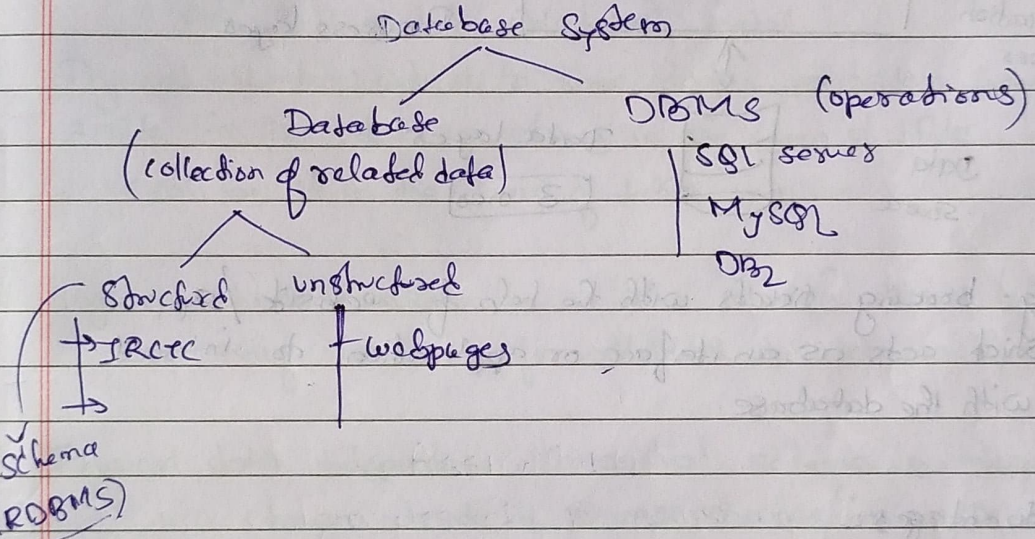
AMAS | Page No.
Date / /

DBMS - Gate & mashang

→ 3 schema, 3-level of abstraction
→ Models
→ ER Models, Relationships, Keys
* → Normalization →
→ Transaction control & concurrency. (ACID properties)
→ SQL & Relational algebra
→ Indexing
  → DDL, DML, DCL, TCL
  → Constraints
  → Aggregate func.
  → Joins
  → Nested query (In, Not in, any, all)



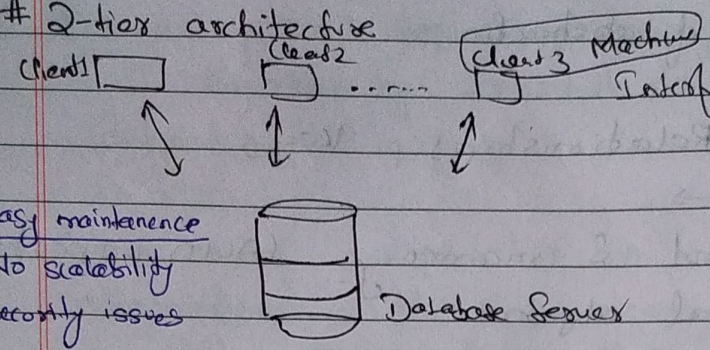
* Structured data is stored in the form of relation.
RDBMS is used to operate on structured data.

File Systems

1. → In dbms, fast searching is there
2. → " " ~~dbs or~~ data is independent of metadata
3. Concurrency: Multiple users can access the db at the same time still inconsistency is not there with the help of some protocols.
4. Security: (Role based Security)
   → User
   → Admin } Role based access control.
   → DBA
5. Data redundancy is not there in dbms.

AMAS | Page No.
Date / /

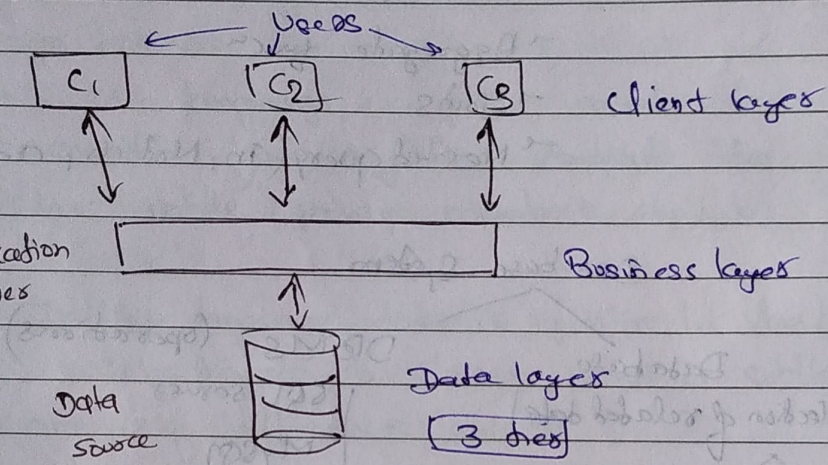
2-tier architecture



* Easy maintenance
* No scalability
* Security issues

e.g. Collecting tickets from window; Bank transaction

3-tier architecture



e.g. booking tickets with the help of website/app (which acts as an interface or application) to interact with the database.

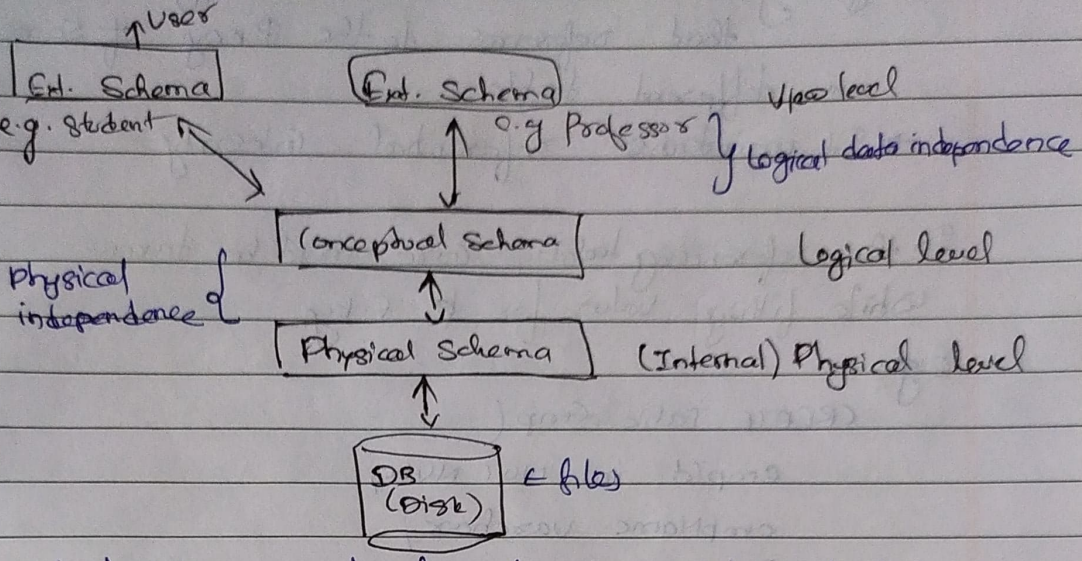
Advantages
→ scalable (huge amount of users can be entertained)
→ Security (data is not vulnerable to users)

Business layer o/ps processed query, understandable by db.

* 3 tier arc. needs maintenance.

Schema
= Logical representation of RDBMS
→ Tables
In E-R
→ Entities and Relationships

3-Schema architecture



Conceptual Schema: structure of representation of data
e.g. E-R model, Relational model.

Physical Schema: where to store data; particular locations, drives

Inside the DB, data is stored in the form of files, but we can see it in the form of tables etc.

Logical data independence is achieved through VIEWS (virtual table having user specific attributes in view table)

Physical data independence: Conceptual schema is independent of changes made in storage structure/indices/Data structure change

Concept of candidate key
keys: To uniquely identify each tuple in a table.
< set of all such attributes (keys) > is called candidate key.

Primary key: Smallest subset of Superkey which can uniquely identify each tuple.

Alternate keys: All other keys other than P.key.

P.keys < Unique + NOT NULL >

Only one Pkey exists for a given relation

Foreign key : An attribute or a set of attribute that references to the P.key of same or different table.
→ It maintains referential integrity

In the referencing table, care must be taken while filling values for f.key.

e.g.

```
CREATE Table Emp(
    empId int NOT NULL,
    empName varchar,
    PersonId int,
    Primary key (EmpId),
    Foreign key (PersonId) References Persons(PersonId));
```

(TableName → Persons)

If table has been created

```
ALTER Table Emp
ADD constraint fk
Foreign key (PersonId) References Persons(PersonId);
```

Referenced table
1. Insert (No violations)
2. Update
3. Delete (Delete all such tuples related to this f.key from referencing table, if any)

Soln :
- → ON delete cascade ⇒ (delete here as well)
- → ON delete set NULL (if f.key is not the P.key in referencing table)
- → ON delete NO Action

2. Update : (may cause violations
→ on Update ....

Referencing table
1. Insert (May cause violations
2. Deletion (No issues)
3. Update : Violations as these

→ Superset of candidate key is a superkey.

→ C.key is minimal superkey

Rollno → P key / cand. key

Roll no + Name → Superkey.

Table having $(A_1 A_2 \ldots A_n)$ attributes

A_1 → is candidate key.

How many superkey is possible?

2^n (if all are to be chosen or not chosen)

∴ A_1 is C key it is to be chosen any how

ans = 2^{n-1}

⇒ If $A_1 A_2$ is C key.

$2^{n-1} + 2^{n-1} - 2^{n-2}$

A_1 A_2 eliminating redundant keys.

ER Model:

→ Entity: Any object having physical existence having some attributes.

Relationships: Association among entities.

Type of attributes

1. Single vs Multivalued (RgId vs Phone No)

2. Simple vs Composite (Age vs Address → Lane, District)

3. Stored vs Derived (DoB vs Age → from DoB)

4. Key vs Non-key attr. (RegId vs ... → unique)

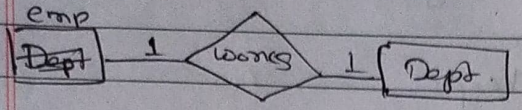
5. Required vs Optional

6. Complex (Composite + Multivalued)

e.g. Multiple addresses

Types of relationships

(i) One to One relationships



↓ It must has Pkey of all participating entities + [Descriptive attributes] (optional)

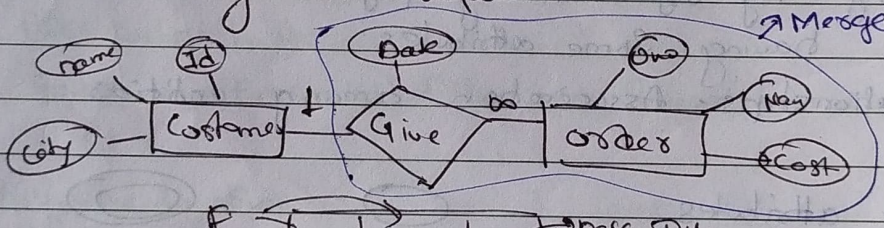
≈ Pkey is any of the foreign key in 1-1 relationship

≈ We can merge the non-primary attribute column to the table from where Pkey was considered.

e.g. Eid Ename age Dept. } Reduction
E1 D1
Pkey of relationship in relationship

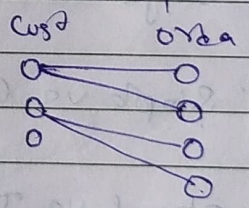
* Lower the cardenality, higher the duplicacy (duplicates) (1-M)

(ii) One to Many relationships



Key	Id	O_no	Date	→ Desc. Attr.
	C1	O1	- -	
	C1	O2	- -	

1-M



1. Primary key = O_no (from many table)

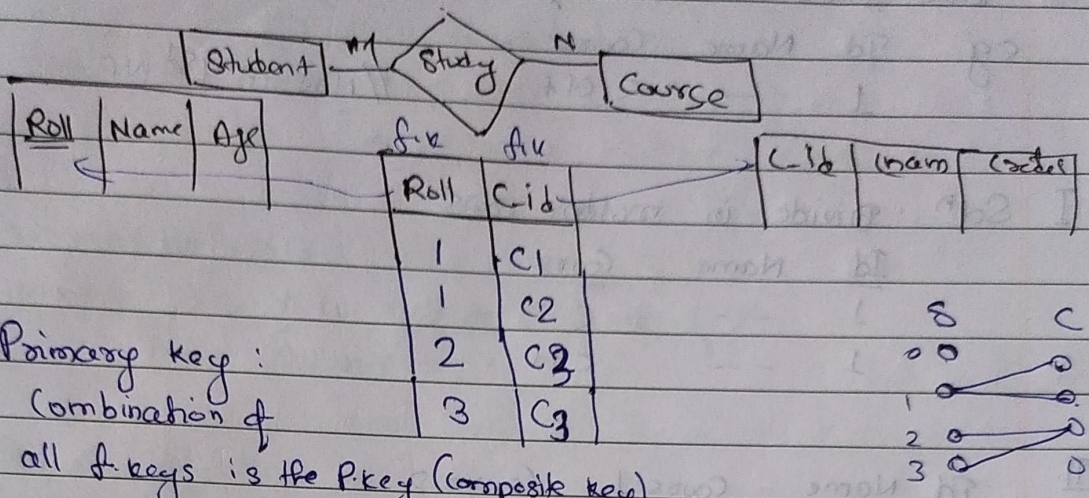
2. Reduction of # of tables

Merge orders with Give

Id	O_no	item	Mfg	Cost	Date
C1	O1				
C1	O2				
C2	O3				
C3	O4				

↑ p.key

iii Many to Many



Primary key: Combination of all f.keys is the P.key (composite key)

Non-reducable.

Normalization

A technique to remove or reduce redundancy from a table.

→ Row level → Column level.

Insertion, Deletion and updation anomaly.

SID	Sname	Cid	Cname	Ad	Fname	Salary
1.	--	C_1	Math	F_1	--	30k
2.	--	C_1	Math	F_1	--	30k

(i) Inserting a new course in this table will create a problem as P.key is SID and initially, no Student has taken the newly introduced course.

(ii) Deleting a student will also delete course details

(iii) Updation of a repet. colm val (i.e Salary) will consume extra amnt of time, as it will have to reflect changes in all the similar values.

Soln: Divide the tables.

AMAS | Page No.
Date / /

First Normal form
Table should not contain any multivalued attributes
eg Id Name Course
1 -- C, C++ // Not in 1st NF.

I Sol^n : Divide in multiple rows :

Id	Name	Course
1	--	C
1	--	C++

II. Id Name Course1 Course2.

III. Base Table. Referencing Table.

Id	Name
1	--
2	--

Id	Course
1	C
1	C++

# Closure Method (To find all possible candidate keys)
R(ABCD)
FD {A→B, B→C, C→D}
A^+ = ABCD
B^+ = BCD
C^+ = CD
D^+ = D

C.K = {A}
Prime attribute = {A}
Non-prime attribute = {B, C, D}

eg R(ABCDE)
FD = {A→B, BC→D, E→C, D→A}
Right side {BDCA}
↳ E not there, so for C.key E must be there in LHS.
E^+ = EC ✗
AE^+ = BECDA ✓

C.key = {AE}
check if A is there in RHS
D ⇒ C.key {AE, DE} ⇒ C.key = {AE, DE, BE}
BE^+ = ABCDE ✓
CE^+ = BBCDE ✗

AMAS | Page No.
Date / /

Functional Dependency.

→ Trivial F.D

If $X \to Y$ ⇒ Y is a subset of X.
and L.H.S ∩ R.H.S ≠ ∅
eg $AB \to A$ ⇒ ∩ = A or $A \subseteq AB$

→ Non-trivial F.D

If $X \to Y$, Y is not a subset of X.
L.H.S ∩ R.H.S = ∅

Properties of FD.

(i) Reflexivity: if Y is a subset of X, then $X \to Y$

(ii) Augmented: If $X \to Y$, then $XZ \to YZ$

(iii) Transitive: if $X \to Y$ and $Y \to Z$ ⇒ $X \to Z$

(iv) Union: if $X \to Y$ and $X \to Z$ ⇒ $X \to YZ$.

(v) Decomposition: if $X \to YZ$ ⇒ $X \to Y$ & $X \to Z$.

(vi) Pseudo-transitivity: if $X \to Y$ and $WY \to Z$ then $WX \to Z$
if $X \to Y$ and $Z \to W$ then $(XZ \to YW)$

Validity is checked for non-trivial FDs.

Second Normal form

1. Table must be in First Normal form
2. No partial dependency ⇒ All the non-prime attributes should be fully functional dependent on candidate key.

i.e. If composite P.key is there, non-prime attributes should be fully dependent on them, and not any part of the composite key (partial dependency)

AMAS | Page No.
Date / /

Soln. Divide the table.

Steps to check whether a given relation is 2^{nd} NF.

find Ckey, Prime Attribute, Non-prime attribute.
check for each NPA, if it is determined by a proper subset of Ckey, then it's not in 2^{nd} NF. Or L.H.S is a proper subset of Ckey and R.H.S is a NPA (then Not in 2^{nd} NF)

Third NF

→ Table must be in 2^{nd} Normal Form.
→ No transitive dependency should be there.
(i.e. A NPA should not determine a NPA).

∀ each F.D. check if L.H.S must be a Ckey or Skey or/ R.H.S must be Prime Attr

find Candidate key:

R(ABCD)
FD. AB→CD , D→A

(1) check R.H.S {C,D,A}
B is not there, we need B in L.H.S to form a Ckey.

$B^+ = \{B\}$

$(AB)^+ = ABCD$
Reflexive

Ckey = {AB} {DB}

if found (Ckey):
check if A or B appears in R.H.S
if yes replace with their determinant
ie

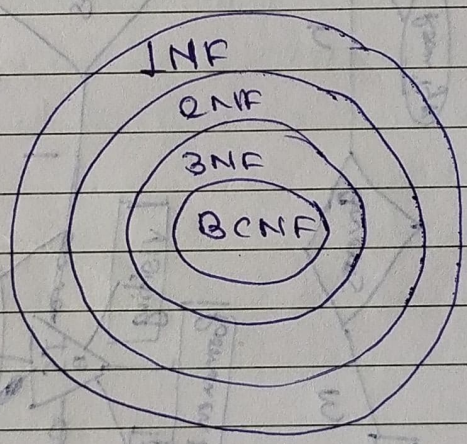
$AB \Rightarrow (DB)^+ = DBAC$

DB
↳ AB (Already done)

BCNF (Boyce Code Normal Form)

→ Table must be in 3NF

→ ∀ each F.D. L.H.S must be a ckey or Skey



DBMS

continued after normalization:

Lossless and lossy Decomposition

Whenever we tend to normalize the relation, we decompose them into one or more relation. In the process of again joining them, if there is a loss (inconsistency) of data then it was lossy decomposition else lossless. redundancy.

→ During decomposition, atleast one common attributes should be there. And the common attribute should be key/secy of either R_1/R_2 or both (unique).

After joining the decomposed relation, even if there are extra tuples it is lossy decomposition.

The extra tuples which appears in the joined table, they are called Spurious ~~table~~ tuple.

e.g R

A	B	C
1	2	1
2	2	2
3	3	2

R_1

A	B
1	2
2	2
3	3

R_2

A	C
1	1
2	2
3	2

R_1 Natural join R_2 gives

A	B	A	C	
1	2	1	1	●
1	2	2	2	X
1	2	3	2	X
2	2	1	1	X
2	2	2	2	
2	2	3	2	X
3	3	1	1	X
3	3	2	2	X
3	3	3	2	

(remove tuples where $R_1.A \neq R_2.A$)

⇒

A	B	C
1	2	1
2	2	2
3	3	2

Lossless decomposition. A was key of R_1 & R_2 both.

(No Spurious tuple)

Conditions of Lossless decomposition.

(1) $R_1 \cup R_2 \equiv R$

(2) $R_1 \cap R_2 \neq \emptyset$

(3) The common attr. must be c.key/s.key of either of decomposed tables or all tables.

Fourth Normal Form

→ BCNF + No multivalued dependency

5th Normal Form

Name	Mob	Email
Varun	M1	E1
Varun	M1	E2
"	M2	E1
"	M2	E2
Varun	M3	E1
"	M3	E2

↙ Multivalued dependency

$X \twoheadrightarrow Y$

→ Name →→ Mob

Name →→ Email

5th Normal Form

4th N.F + Lossless Decomposition.

Minimal cover in DBMS

Step1: In R.H.S only one attr

Step2: Remove redundancy

Step3: AC→D

→ Ct trash then C→D ≡ AC→D

LHS to one attr.

e.g find minimal cover of $\{A \to B, C \to B, D \to AB, AC \to D\}$

Step 1: $A \to B, C \to B, D \to A, D \to B, D \to C, AC \to D$

Step2: check if A→B can be removed $A^+ = A$ ✗

" " C→B, $C^+ = C$ ✗

D→A, $D^+ = DBC$ ✗

D→B, $D^+ = DABC$ ✓

D→C, $D^+ = DAB$ ✗

AC→D, $(AC)^+ = ACB$ ✗

We get $A \to B, C \to B, D \to A, D \to C, AC \to D$

Step 3:

$AC \to D$

$C^+ = CB$ (No A)

$AC \to D$

$A^+ = AB$ (No, C)

Final ans; (Union)

$A \to B, C \to B, D \to AC, AC \to D$

Q. Check highest NF for a given Func Dep.

→ Find Cand. key

→ Find Prime & NPA

→ For each FD check from BCNF to 1st NF.

If all satisfies break.

(Table is assumed to be in 1st NF)

Finding out NF of a relation

e.g. $R(ABCDEF)$ $CK = \{AB, FB, EB, CB\}$

$PA = \{A, B, C, E, F\}$ $NPA = \{D\}$

$FD = \{AB \to C, C \to D, C \to B, E \to F, F \to A\}$

2I. Remove partial dependency. { LHS must be proper subset of key and RHS must be a NPA.

$C \to D$ (NPA)

proper subset

Now divide the tables (in a lossless way)

$R(ABCDEF)$

$R_1(ABCEF)$ $R_2(CD)$ $C^+ = CD$

$AB \to C$ $C \to D$ ↳ key

$C \to E$

$E \to F$ (Now no PD)

$F \to A$

key $\{C\}$

$CK\{AB, FB, EB, CB\}$

AMAS | Page No.
Date / /

S II Remove transitive dependency
→ LHS is c/cg/s/sg or RHS is a PA (Not in TD)

(R1) (R2) (CD)

AB→C, C→E, E→F, F→A C→D

Already in 3rd NF.

S III check for BCNF
→ LHS must be ckey

R1 {AB→C, C→F, E→F, F→A}

✓ × × ×

Not in BCNF

R2 : C→D ✓

In BCNF (0% redundant)

Decompose R1

R1(ABCEF)

R3	R4	R5	R6
(AB→C)	(C→E)	(E→F)	F→A
(ABC)	(CE)	(EF)	(FA)
∝ ABγ	∝ Cγ	∝ Eγ	∝ Fγ

R2 (C→D)

R2 (CD)

∝ Cγ

Lossless join.
Let's say a query about B & F
combine R3 & R4 (C is ckey)

But if we want to join R3(ABC) & R5(EF)
→ we can't join directly;
first join R3 & R4 then with R5(EF)
(ABCE) (EF)
join

AMAS | Page No.
Date / /

Date: 03/08/21

Equivalence of Functional dependency

Given X & Y

check X covers Y $X \supseteq Y$

Y covers X $Y \supseteq X$

$\Rightarrow X \equiv Y$ (equivalent)

(i) check if X covers Y

eg $X = \{A \to B, B \to C\}$, $Y = \{A \to B, B \to C, A \to C\}$

for each F.D in Y, take closure from X to see if the closure covers the determinant from Y.

(i) for $A \to B$ (from Y)

$A^+ = ABC$ (closure from X)

↳ B covered

(ii) $B \to C$

$B^+ = BC$

↳ C covered

(iii) $A \to C$

$A^+ = ABC$

↳ C covered.

So, T && T && T $\Rightarrow$ X covers Y.

(i) (ii) (iii)

Similarly, check whether Y covers X

(i) $A \to B$ (X)

$A^+ = ABC$

↳ covered

(ii) $B \to C$

$B^+ = BC$

↳ covered

So, Y covers X

$\Rightarrow X \equiv Y$

Dependency preserving decomposition

If R is decomposed into R_1 & R_2

$\Rightarrow FD_1 \cup FD_2 = FD^+$ (then dependencies are preserved)

eg R(ABCD)

FD = {A→B, B→C, C→D, D→B}

decomposed into $R_1(AB)$, $R_2(BC)$, $R_3(CD)$

$R_1(AB)$	$R_2(BC)$	$R_3(CD)$
~~A→A ✓ trivial~~		
A→B ✓ (exists in R)	B→C ✓	B→D ✓ $B^+ = BCD$
B→A ✗ (doesn't exists)	C→B ✓	D→B ✓
	$C^+ = CDB$	~~$D^+ = DBC$~~

B→A (Not found)

$\rightarrow B^+ = BCD$ (closure from R)

So, final union of FDs (exists in R)

$FD^+ = FD_1 \cup FD_2 \cup FD_3 = \{A \to B, B \to C, C \to B, B \to D, D \to B\}$

(closure in FD^+) $C^+ = CBD$ ($C \to D$)

So, dependencies are preserved.

* Joins & subqueries gives similar results, later is complex.

Joins = cross product + ~~Select~~ statement (condition)

→ Cross join (cross product)

→ Natural join

→ Conditional

→ Equi join

→ Self join

→ Outer
  - left
  - right
  - full

Natural join

e.g. Find emp name who is working in a department.

emp

eno	ename	Add

dept

dno	Name	ew

Ans: Select e_name from (Emp, Dept) → cross product
where emp.eno = dept.eno ;

↓ natural join ↑

Select e_name from (Emp Natural join Dept);

In Natural join
- cross product is calculated ; and only the tuples where values for common attribute are same are taken into consideration rest are ignored.

e.g. In above example, e_no was common attribute.

Self-Join : Table joined with itself
(SQL query starts with from)

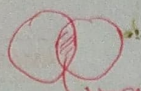
e.g. find student id who is enrolled in at least two courses

Study

Sid	Cid	Since
S_1	C_1	2016
S_2	C_2	2017
S_1	C_2	2017

Select T_1.Sid from Study as T_1, Study as T_2
where T_1.Sid = T_2.Sid and T_1.Cid <> T_2.Cid ;
(<> not equal to)

In Self-Join
=> Cross product with itself
=> Apply same equivalent (non-equivalent constraints according to need)



Natural join

AMAS | Page No.
Date / /

Equi-join

e.g. Find the ename whose department's location is same as their address

Emp

Eno	Ename	Address

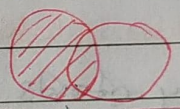
Dept

Dno	location	eno

SQL query: Select ename from emp, dept
where emp.eno = dept.eno and
empaddress = dept.location;

In equi-join:
Natural join + some additional constraint (equivalence)

Left outer join: It gives the matching rows and the rows which are in left table but not in right table.



Left outer

Emp

Eno	Ename	Dept-no

Dept

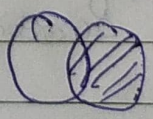
Dno	Dname	location

eg. Select eno, ename, dname, loc
emp left outer join dept
on (emp.dno = dept.dno);

for A, B ⇒ A∩B + (A−B)

Right outer join: It gives the matching rows and the rows which are in right table but not in left table.

for (A, B) = A∩B + (B−A)



In general, Left / Right / Full
= Natural join + Something

... emp Right outer join dept ...

Full outer join = (Left outer join) ∪ (Right outer join)

Relational Algebra (formal query language)
(procedural language)

What to do? + How to do?

Operations

Basic operation
- Projection (π)
- Selection (σ)
- Cross product (×)
- Union (∪)
- Rename (ρ)
- Set Difference (−)

Derived operation
- Join (⋈)
- Intersect (∩)
- (X ∩ Y) = X − (X − Y)
- Division (/, ÷)

Projection (π)

Query: Retrieve the roll number from Student (table)

$\pi_{RollNo}(Student)$

→ It projects only distinct values (no duplicate data in a column is projected)
↳ row

→ Projection is used at end.

Selection (σ)

Query: Retrieve name of student, whose RollNo = 2.

→ narrows search area. (Row level selection)

$\pi_{name}\left(\sigma_{Roll\_no=2}(student)\right)$

↓ selects that particular row

(projection at end)
column level partition

AMAS | Page No.
Date / /

Date: 04/08/21

Cross - product (At least 1 col should be common)

In answer

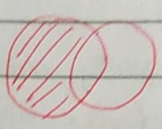
No. of columns = Sum of columns
No. of Rows = Product of rows

$R_1 \times R_2$

Set difference (−)

A − B = A but not B

= A − (A ∩ B)



1) No. of columns must be same
2) Domain of every column must be same.

e.g

Roll	Name
1	A
2	B
3	C

Eno	Ename
7	E
1	A

(Student − Employee) =

Roll	name
2	B
3	C

$\Pi_{name}(Student) - \Pi_{name}(Employee)$

name
B
C

Union

$A \cup B = A + B - A \cap B$

→ conditions same as set difference

(Student ∪ Employee) =

Roll	Name
1	A
2	B
3	C
7	E

$\Pi_{name}(Student) \cup \Pi_{name}(Employee)$

AMAS | Page No.
Date / /

Division operation

for 'ALL' and 'EVERY', use so division

$A(x,y) / B(y) =$

Find sid who are enrolled in all subjects $\rightarrow \pi_{cid}(S_1)$

form whose data is expected

Enrolled

Sid	Cid
S_1	C_1
S_2	C_1
S_1	C_2
S_3	C_2

Course

cid
C_1
C_2

→ Find cross product of sid (enrolled) × course
→ Sett difference of above o/p and enrolled
→ in cross product but not in Enrolled
↳ this gives disqualified candidates
→ Total − disqualified = qualified (students studying all subjects)

$\pi_{sid}(enrolled) - (\pi_{sid}(\pi_{sid,cid}(enrolled) \times \pi_{cid}(course) - Enrolled))$

Total − disqualified

Tuple Relational calculus (Non-procedural)

RC
- TRC
- Domain RC

Query is expressed as
$\{t \mid P(t)\}$
↳ tuples satisfying predicate $P(t)$ [conditions]

Operations: OR(∨), AND(∧), NOT(¬)
Quantifiers: $\exists t \in r\,(Q(t))$, if any tuple ⇒ true
$\forall t \in r\,(Q(t))$, is true for all tuples in relation r.

:= SEQUEL: Simple eng. query lang.
EF codd: father of DBMS.

AMAS | Page No.
Date / /

Unsafe expression:
e.g $\{s.name \mid \neg Supplier(s)\}$
⇒ gives ∞.

Free variable:
∃, ∀ is not used
Bounded
∃, ∀ is used before

SQL
IBM implemented the paper published by EF codd.
90% of data is non-structured, so NOSQL is in trend.

→ SQL is domain-specific language
(we can only use it in structured/relation data)
→ SQL is a declarative language
(only what to do)

But PLSQL is procedural as well.
Operations (like, Blw, In, Not in, conditional)
Clauses (distinct, order by, group by, from, having)
Aggregate func (min, max, count, avg, sum)
Joins & Nested Queries (subqueries)
PL/SQL (Triggers, functions, cursors, procedures)

SQL commands
DDL: Create, Alter, Drop, truncate, Rename
DML: Insert, update, delete, select. (CRUD)
DCL: Grant, Revoke
TCL: Commit, Rollback, Savepoint
Constraints: P-key, F-key, Check, Unique, Default, Not Null.

Create Table emp (
colname1 type, ...);

desc tablename; // Shows structure of table.

Number(scale, precision)

ALTER Command:

→ Add column
Alter table name add (address varchar(30));

→ Remove column(s)
Alter table name drop column colname

→ Modify datatype	modify colname datatype (new)
→ Modify datatype length	n
→ Add / Remove constraints	add Primary key (rollno);
→ Rename column / Table.	rename column colnam to newnam; (old)
	rename to new_tablename;

Difference b/w Alter and update

Alter	update
① DDL	① DML
② used to change structure	② used to change tuple's data.

e.g update tablename set salary = salary * 2;

Diff b/w Delete | Drop | Truncate

Delete	Drop	Truncate
① DML	DDL	DDL
② Deletes tuples	② Deletes structure also	② Truncate student; (It is a special case of delete)
eg Delete from tabname where id = 2;	desc tabnam doesnt exist (error)	③ Faster.
③ Slower	③ No rollback	④ Rollback not possible
④ Rollback possible. (if committed)		

AMAS | Page No.
Date / /

Constraints in SQL
(Restrictions before putting the data)

1) Unique
2) NOT NULL
3) Primary key (unique + Not Null)
4) check
   char (age > 18)
5) Foreign key
6) Default
   e.g Salary int default val ;

SQL queries and Subqueries

Emp

Eid	Ename	Dept	Salary
1	Ram	HR	10000
2	Amrit	MRKT	20000
3	Ravi	HR	30000
4	Nitin	MRKT	40000
5	Varun	IT	50000

Q1. Display employ name who is taking max salary

```
Select ename from emp
where salary = select max(salary) from emp ;
                    Inner query
```

Date: 05/08/21

Q2 Find 2nd highest salary & taking employee's name.

→ Inner queries runs for only one time

```
select ename from emp where salary
(select max(salary) from emp whose
   salary <> (select max(salary) from emp) );
```

↓ used in

AMAS | Page No.
Date / /

Q.3 Write a query to display all the dept names along with no. of emps having working in that.

Group by to group similar things together.

(i) Only the column after 'group by' can be written after 'select'

(ii) Apart from above aggregate func" can be used

```
Select dept, count(*) from Emp group by dept;
```

MRKT	2
HR	2
IT	1

Q4 Write a query to display all the names of dept where no of emps are less than 2.

Instead of 'where' we use having with group by

```
Select dept from emp group by dept
having count(*)<2;
```

Q.5 Display highest salary dept wise and name of emp who is having that salary

```
Select ename, salary from emp where
salary in (Select max(salary) from emp
          group by dept);
```

equal to can't be used (∵ inner query has multiple rows)

IN/NOT IN

```
Select * from emp where
address IN ('Chandigarh', 'Delhi');
```

Project

Eid	Pid	Pname	Location

Find the name of the employees who are working on a project

```
Select ename from emp where
eid IN (select eid from project);
```

Exists/Not exist
Correlated query (Top down)
For each outer query iteration, run the inner query

Q. Find the details of emp who is working on at least one project.

```
Select * from emp where
eid exists (select aid from project
            where emp.eid = project.eid);
```

It behaves like two nested for loops in C language

SQL aggregate functions
count(col_name) won't count a tuple if a NULL value is there.

Sum
DISTINCT(Sum(salary)) Sums counts distinct value only.

AVG.
$$AVG(salary) = \frac{Sum(salary)}{Count(Salary)}$$

$$Distinct(Avg(Salary)) = \frac{Distinct(Sum(sal))}{Distinct(count(Salary))}$$

AMAS | Page No.
Date / /

Correlated Subquery (synchronized query)
↳ It is a subquery that uses value from outer query (top-down approach)

Q. Find all employees who work in a department

Emp: eid ename addr | Dept: Did Dname eid

```
Select * from emp where
exists (select * from dept
        where emp.eid == dept.eid);
```

↑ Compare outer query value ⇒ entire row

# Nested Subquery	Correlated query	Joins
Bottom-up	Top-down	(correspond + constraint attributes)
Select * from emp where eid in (select eid from dept);	Select * from emp where exists (select id from dept where emp.id = dept.id);	Select * from emp, dept where emp.id = dept.eid;
	exists returns True or False	→ Scores faster but saves space
⇒ less no. of comparisons		

Find Nth highest salary using SQL

```
select id, salary from emp e1
where N-1 = (select count(distinct salary) from
             emp e2 where
             e2.salary > e1.salary);
```

↑ no. of salary greater than e1.sal in e2.

For 4th highest, 3 higher salaries should be there in e2.

Like command

% → anything

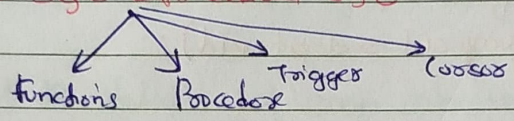
_ → places = no. of underscores

* →

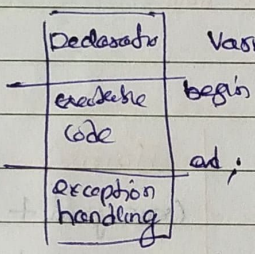
(Select * from R) intersect (select * from R)

⇒ (produces unique tuples from *)

PL-SQL & Procedural SQL (what & How to do both)



PL: Block



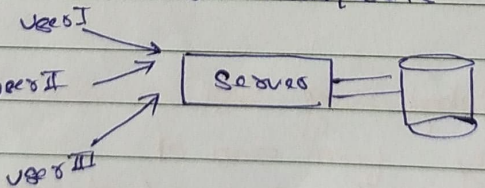
a:=10;

Transaction Concurrency:

Transaction: A set of operations used to perform a logical unit of work.

→ It generally represent change in database.

→ Basic operations in DB → Read/write or commit



ACID properties

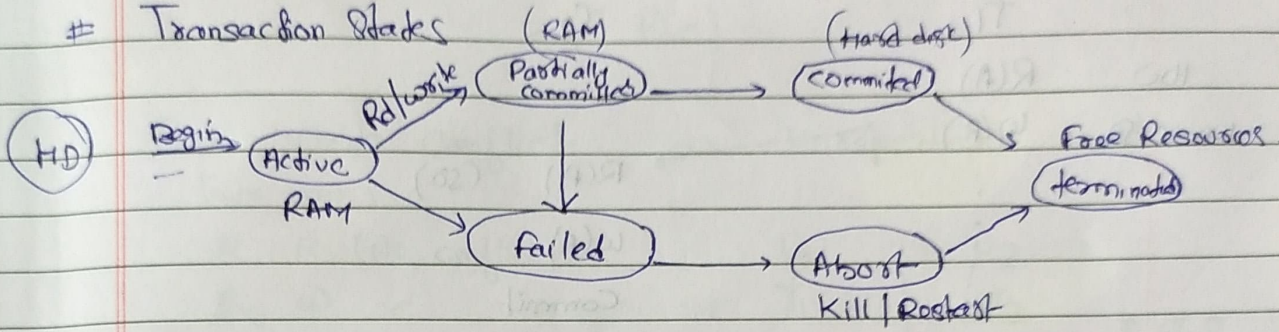
Atomicity: Either all or none

Consistency: Before the transaction starts and after the completion of transaction sum of money should be same.

Isolation:
Converting a parallel schedule into serial schedule.

Durability: All changes made to DB should be permanent.

Transaction States

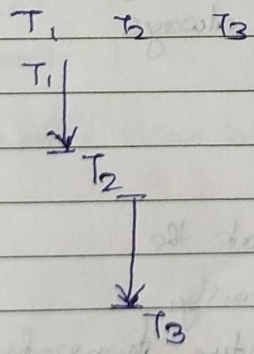


If 'n' instructions are there
(n-1) are general instructions → partially committed
nth instruction is 'commit' → committed.

Failed/Aborted transactions can't be resumed, it has to be started.

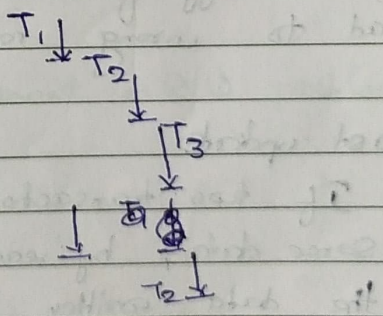
Schedule: chronological execution sequence of multiple transactions

Serial



once a transaction is not done with execution, no other trans. can start
→ Consistency
→ Large waiting time
→ Low throughput

Parallel



→ Good performance
→ High throughput
→

Types of problems in Concurrency

→ Dirty read
→ Incorrect Summary
→ Phantom read.
→ Lost update
→ Unrepeatable read

Dirty read

	T_1	T_2
100	R(A)	
100-50=50	W(A)	
	⋮	R(A) (50)
	⋮	W(A)
	⋮	Commit
	Failure	

(T_2 is reading/working on a wrong data)

∵ T_1 has failed.

Incorrect Summary:

While one transaction is performing some operations, and meanwhile another transaction T_2 enters and starts executing, and performs some agg. functions; this will always lead to wrong results.

Lost update

If two transaction tries to update the same data; by reading simultaneously, the data written by first committing transaction will be lost.

Unrepeatable read

T_1	T_2
R(x)	
	R(x)
W(x)	} both will have
R(x)	diff values.

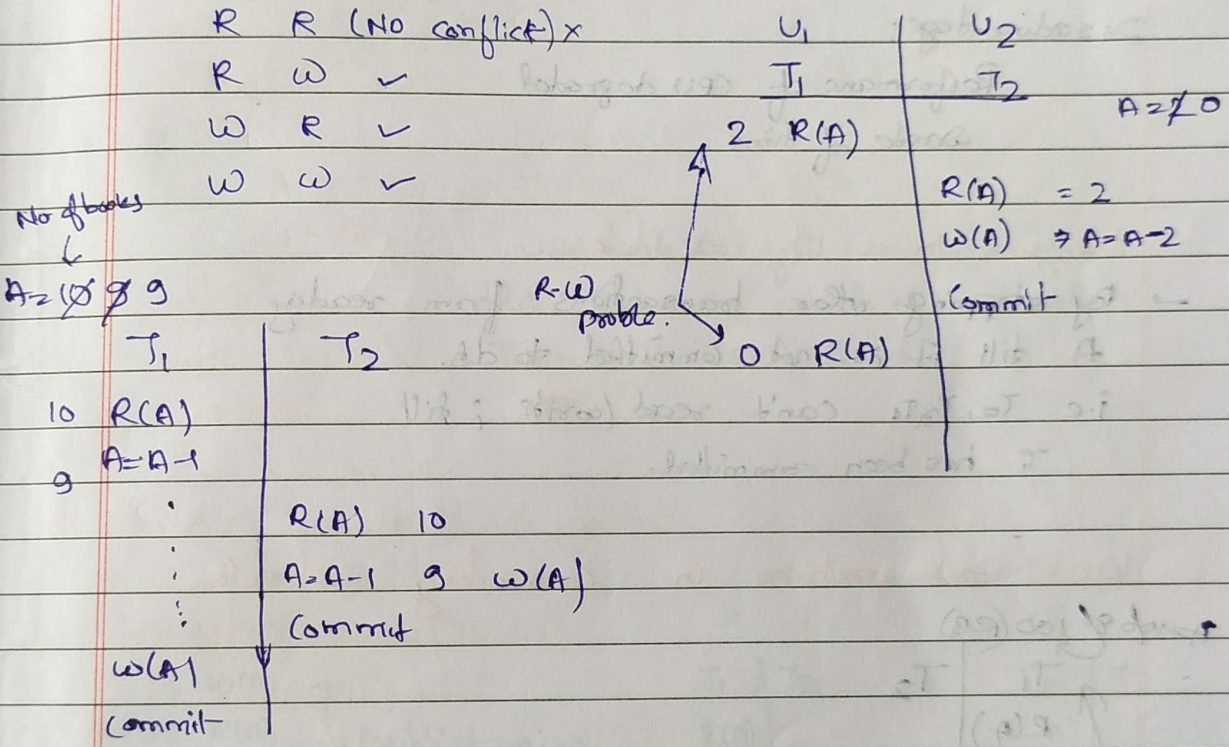
Phantom Read

T_1	T_2
R(x)	
	R(x)
Delete(x)	
	R(x)

Read - write conflict or Unrepeatable read.

Four cases

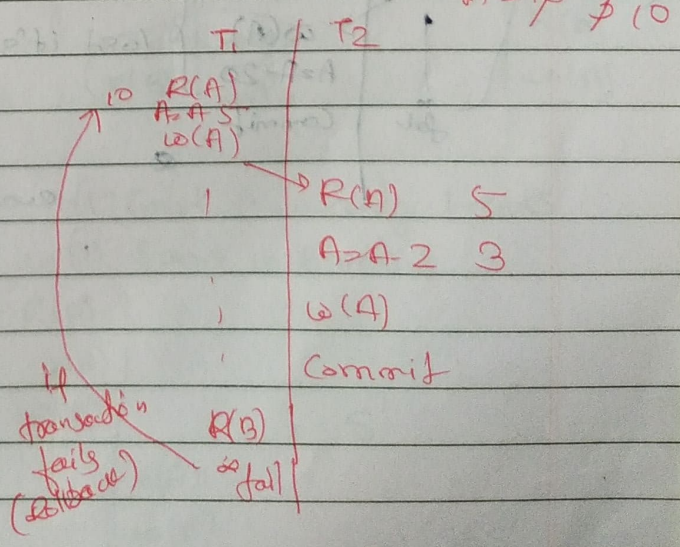
R R (No conflict) ×
R W ✓
W R ✓
W W ✓



⇒ 2 books issue but only one decremented ⇒ R/W problem

Date: 06/08/21

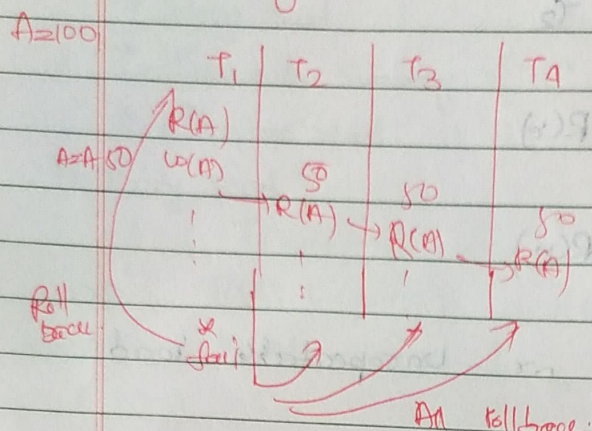
Irrecoverable Schedule



Here even if T_2 has completed it's transaction Database roll backed to its initial state. We lost all changes due to T_2.

It is irrecoverable.

Cascading Vs Cascadeless schedule



All rollback if one fails

Disadvantage:

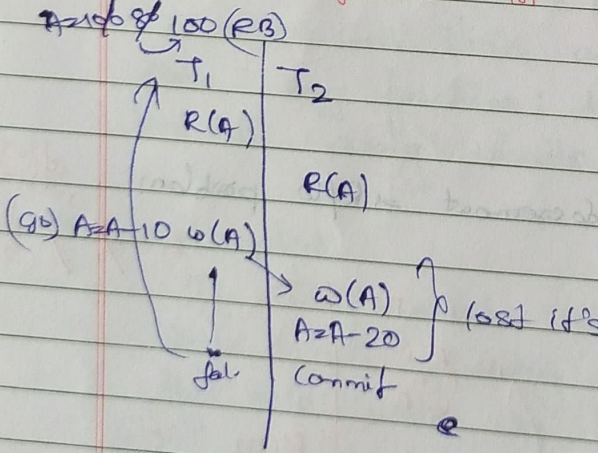
Performance of CPU degraded because of time.

We can make it cascadeless

→ By stopping other transactions from reading A till A is not committed to db.

i.e T_2, T_3, T_4 can't read ; till T_1 has been committed.

Write-write problem in Cascading schedule



even if T_2 has committed so, T_1 will roll back and re-initialize db value

Serializability → conflict / view

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)

$T_1 \rightarrow T_2$

T_1	T_2
	R(A)
	W(A)
R(A)	
W(A)	

$T_2 \rightarrow T_1$

Both are serial, no need to Serialize

T_1	T_2
R(A)	
	R(A)
	W(A)
R(A)	

$T_1 \rightarrow T_2$

Parallel

We need to find a clone for this schedule which is serial. Then it is serializable.

Finding conflict equivalent

R(A) R(A) } Non-conflict pairs

R(A) W(A)
W(A) R(A)
W(A) W(A) } conflict pairs

R(B) R(A)
W(B) R(A)
R(B) W(A)
W(A) W(B) } Non-conflict pairs

S

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	

NCP

S'

T_1	T_2
R(A)	
W(A)	
R(B)	
	R(A)
	W(A)

(i) Find non-conflict pairs
(ii) Swap in timeline

$S \Rightarrow$

T_1	T_2
R(A)	
W(A)	
	R(A)
R(B)	
	W(A)

NCP

$\Rightarrow$

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

$= S'$

$\Rightarrow S \equiv S'$

(conflict equivalent)

If $S \xrightarrow{CE} S' \rightarrow$ Serializable

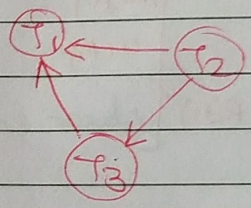
Conflict Serializability

- check if a schedule is conflict serializable

S

T_1	T_2	T_3
R(x)		
	R(y)	
	R(x)	
R(y)		
R(z)		
	w(y)	
	w(z)	
w(x)		
w(z)		

- Precedence graph (To draw edges)
- Choose conflict pairs in other transactions and draw edges



e.g. for (R(x)) we find w(x) in remaining transactions.

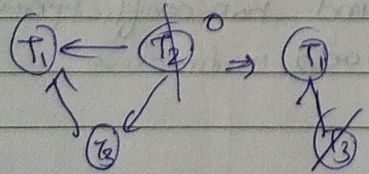
for (W(x)): find w(x) & R(x)

- Check if a loop/cycle exists

if NO loop → this schedule is conflict serializable. We can find atleast one serialized schedule → It is consistent

finding serialized schedule out of all possibilities:

(i) find indegree of all nodes, start with zero indegree node remove and add to answer



$T_2 \rightarrow T_3 \rightarrow T_1$

No loop/cycle → conflict serializable → Serial → Consistent

AMAS | Page No.
Date / /

View Serializable

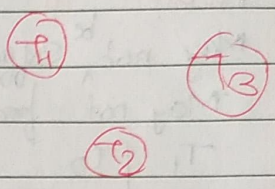
If loop is there, we need to check for view serializable

A=1000

S

T_1	T_2	T_3
R(A)		
A=A-40 60 W(A)	~~W(A)~~	
	W(A) A=A-40 20	
		W(A) A=A-20 0

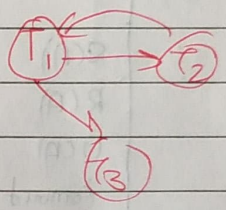
Blind write



Another schedule (S')

T_1	T_2	T_3
100 R(A)		
	~~A=A-40 (60)~~ W(A)	
~~A=A-40~~ (20) W(A)		
		W(A) A=A-20 (0)

S'



Here, both S and S' gives same result although S' has a loop and it's non-conflict serializable but ∄ S which is serial and equivalent to S' yielding same results.

1. # Shared-exclusive Locking

It shows how to make them recoverable / serialized. They are known as Concurrency control protocols.

Shared Lock (S) ⇒ if transaction locked data item in Shared mode then allowed to read only.

Exclusive lock (X) ⇒ if transaction locked data item in exclusive mode then allowed to Read and write both.

e.g.

T_1	T_2
S(A)	X(A) ← Exclusive lock
R(A)	R(A)
U(A)	W(A)
	U(A)

Shared lock → S(A)

Unlock A → U(A)

Unlock → U(A)

Compatibility table:

grant \ Request	S	X
S	✓	x
X	x	x

Problems in S/x locking:

(i) May not be sufficient to produce only serializable schedule

(ii) May not free from irrecoverability

(iii)

T_1	T_2	
R(A)		
R(A)		
W(A)		
U(A)		
:	S(A)	(in this case T_2 has already become committed, so it can't be rollbacked)
:	R(A)	
:	U(A)	
:	commit	
x fail		

Rollback

[So there is inconsistency.
dirty read.

(iii) May not free from deadlock.

T_1	T_2	
(G) X(A)		[infinite wait deadlock]
	X(B) (G)	
(wait) X(B)		
	X(A) (wait)	

(iv) May not free from starvation

a transaction has to wait due to a more poor transaction, but not infinitely.

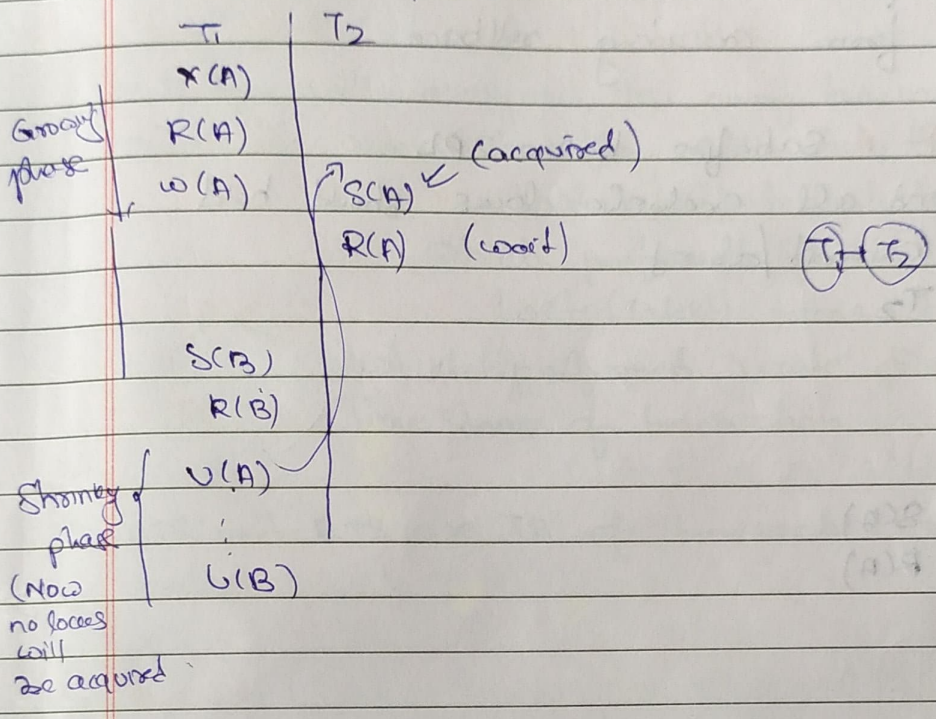
T_1	T_2	T_3	T_4
	S(A) (G)		
(W) X(A)	:	(G) S(A)	
:	U(A)	:	S(A) (G)
:		U(A)	

T_1 is starving

2-phase locking (2PL)

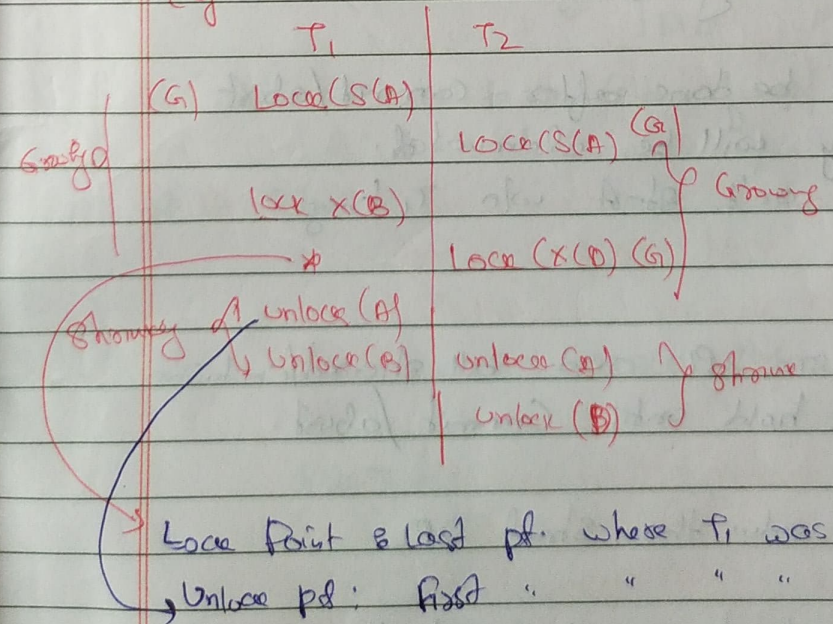
Growing phase: locks are acquired and no locks are released.

Shrinking phase: locks are released and no locks are acquired.



T_2's growing phase may also start while T_1's growing phase is going on based on compatibility.

eg.



Lock Point: Last pt. where T_i was locked

Unlock pt: First " " " " unlocked.

2PL always ensured serializability

AMAS | Page No.
Date / /

Drawbacks in 2PL

(1) May not free from irrecoverability

(2) Not free from deadlocks

(3) Not free from starvation

(4) Not free from cascading rollback

Strict 2PL : Satisfies basic 2PL and all exclusive locks should hold until commit/abort.

T_1	T_2
X(A)	
R(A)	
W(A)	
	S(A)
	R(A)
Commit	
U(A)	

Now
T_2/T_3/T_4 can read or ask for S(A), ∵ they will read from db once T_1 commits. So, cascading rollback is eliminated

Unlocking will only be done after commit/abort so, irrecoverability will be eliminated.
∵ T_2/T_3/T_4 can't start w/o T_1 being committed

Rigorous 2PL: Basic 2PL and all shared/exclusive unlocks should hold until Commit/abort.

It has no extra adv. than Strict 2PL.

Both the 2PL doesn't talks about deadline and starvation

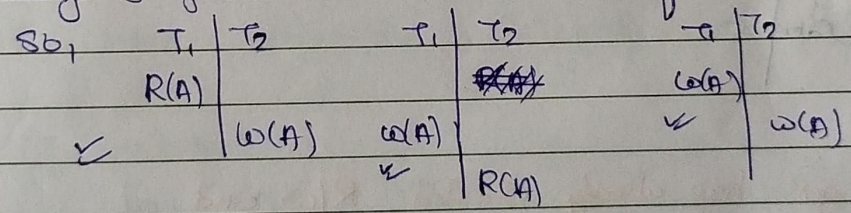
AMAS | Page No.
Date / /

Conservative 2PL: (practically not possible)
Take all locks on all data before starting.
→ It removes deadlock problem.

Time Stamp ordering protocol
→ Unique value assigned to every transaction
→ Tells the order (when they enter into the system)
→ Read_TS (R-TS): Last (latest) transaction no. which performed 'Read' successfully.
→ Write_TS (W-TS): Last (latest) transaction no. (ts, TS) which performed 'write' successfully.
→ TS (T_i): Time Stamp of transaction.

W-TS and R-TS are TS of transaction satisfying criterias

Rules
Priority is given to older transactions for completion



∴ we will complete T_1 (older first).

Rule I: Transaction T_i issues a read operation
(a) if W-TS(A) > TS(T_i), Rollback T_i
(b) otherwise execute R(A) operation
Set R-TS(A) = Max {R-TS(A), TS(T_i)}

Rule II: Transaction T_i issues W(A) operation
(a) if R-TS(A) > TS(T_i), then rollback T_i
(b) if W-TS(A) > TS(T_i), then rollback T_i
(c) Otherwise execute write(A) operation
Set W-TS(A) = TS(T_i)

(a) ie

T_1	T_2
	R(A) (A=10)
W(A) (A=20)	

X won't be allowed

Rule II (a)

younger has already read but, we will complete first. This won't be allowed

(b)

100 200

T_1	T_2
	W(A) = 20
R(A)	
Fail	Rollback

Rollback X Rule I (a)

don't allow T_1 to read.

(c)

T_1	T_2
	W(A)
W(A)	

Rollback

Rule II (b)

If younger has already started R/W and older wants to R/W afterwards, it's not allowed. Since, first older will be completed.

In TSO, $T_1 \rightarrow T_2 \rightarrow T_3$

older — younger

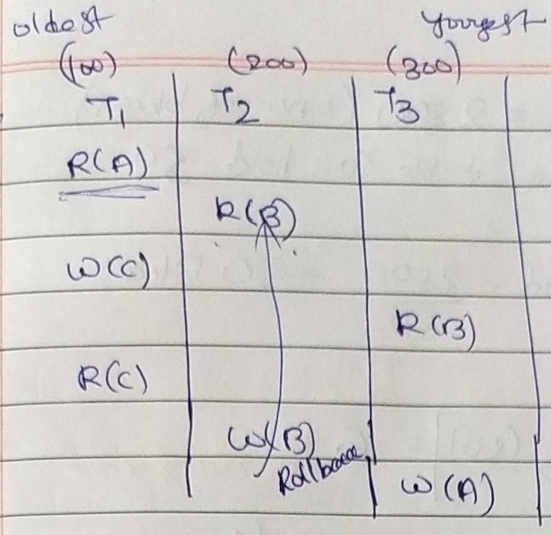
Indexing

Why indexing is used?

→ It reduces the searching time, because we can search our data accordy to indices

I/O cost is reduced dual to indexing.

Example



	A	B	C
RTS	0	0	0
WTS	0	0	0

R(A) for T_1

WTS(A) = 0 > 100 (false)

RTS(A) = MAX(0, 100) = 100

	A	B	C
RTS	100	300	0
WTS	300	0	100

Date: 07/08/21

Indexing

w/o indexing

Eg: Block size = 1000 B

record size = 250B × 10000 (or ordered)

As: No. of blocks = 2500 blocks

each block contains = $\frac{1000B}{250B}$ = 4 records

Avg time = $\frac{\text{worst time}}{2} = \frac{2500}{2} = 1250$

If ordered

$\log_2(2500) = 12$.

With indexing

Eg: Search a record from Index table if

Index table entry = 20B (key + Pointer)
10B 10B

Block size in Index = Block size in HD.

Each block in index table contains = $\frac{1000B}{20B}$ = 50 Nos.

Dense : for each record in HD, we make index

Sparse: Only one index per record (only possible in ordered).

AMAS | Page No.
Date / /

⇒ If sparse, no. of index = 2500 (= no. of blocks)
each block in Index table can hold 50.

So, total blocks needed = $\frac{2500}{50}$ = 50 Blocks

Searching time = $\lceil \log_2(50) \rceil$ = 6 + 1 (exact search)
(∵ Sparse ⇒ ordered)
So after 6 search we find exact block in HD then we search there in particular block for exact record.

(unordered)
⇒ In dense, no of indices = 10000
total blocks needed in index table = $\frac{10000}{50}$ = 200 Blocks

Searching time = $\log_2(200)$
= 8 + 1
↑ search record in that page/block.

Types of indexing
1) Primary 2) Clustered 3) Secondary

ordered file	Primary Index (1 2 3 4 5 ...)	Clustered Index (1 1 2 2 3 3 ...)
unordered file	(2 1 3 4 6 7 ...) Secondary Index	(2 2 3 4 4 ...) Secondary Index
	Key	Non-key

Primary Index

Conditions

→ Ordered data

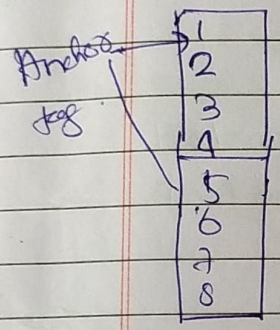
→ key (unique)

Try to make PIndex Sparse

each blocks has one index

ie. # of entries in Index table = # of blocks in HD

(Also known as Anchor record) → the first entry in each block



Time = $\log_2(N) + 1$

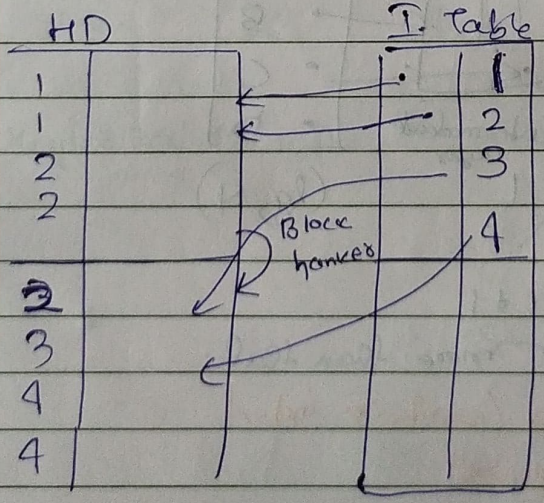
↑ No of blocks in IT.

This depends on Sparse/Dense

Clustered index

→ ordered data

→ Non-key



Time req. in index table

$\log_2 N + 1 + 1$

↓ ↓

No. of blocks in Index table. + max pointer

→ It is also sparse

→

Secondary index (second index)

→ unordered data

→ key / Nonkey data

Why Secondary index??

→ if queries are based on some non-key (primary) attributes like name, pancard.

2nd index is always dense. No cases of anchor data.

→ All records have individual indices

records (in ht D) = # indices.

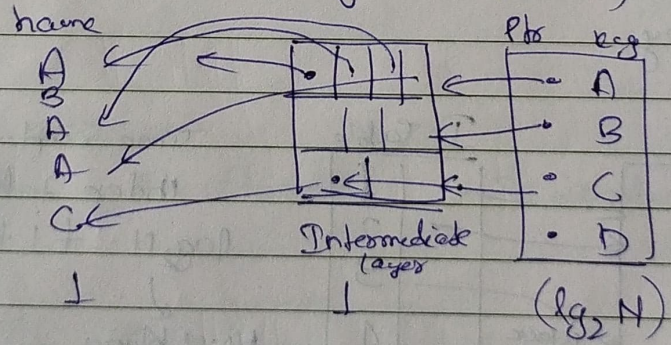
If unordered and key:

We will store the key in sorted form in index.

Time complexity: $(\lg_2 N) + 1$

If unordered and non-key.

→ Intermediate layer (Block of record pointers)



Time Complexity $\lg_2 N + 1 + 1$

more than leaf.

eg Appendix of Books (secondary index with Pg No. (Block of pointers)

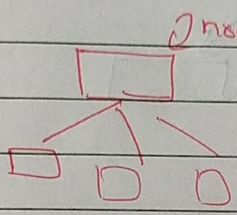
"ABC" 2, 6, 222, 282

AMAS | Page No.
Date / /

Introduction to B-tree and it's structure
(Dynamic Multilevel index)

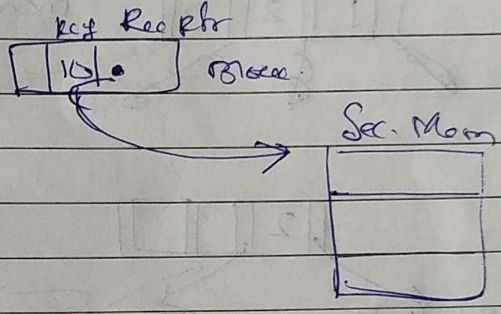
We have used B+ tree, to perform insertion of data and indices as well as in an efficient manner.

Block pointer :



block ptr = no. of children

Keys :
Data pointer
Record pointer

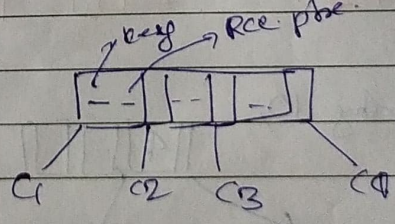


Order of a B-Tree = max no. of children a node can have.

P = order of B-Tree

Children	Root	Intermediate
Max	P	P
Min	2	$\lceil P/2 \rceil$

Keys = Rec. Ptr = P-1.



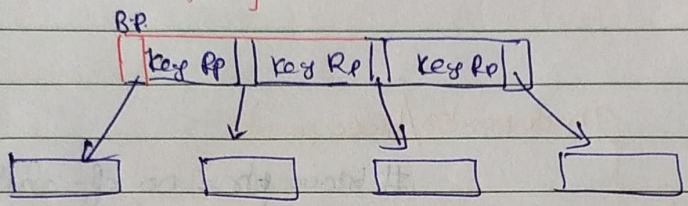
Date 08-08-21

Ex. order = 4

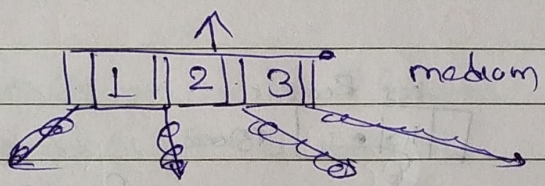
Insert values → 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

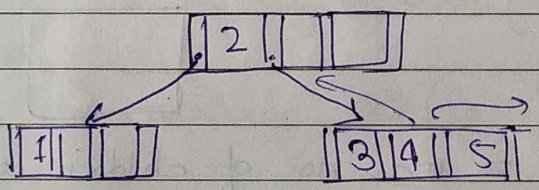
Ans- Max-key = 4 - 1 = 3

Min-key = $\lceil p/2 \rceil - 1 = 1$



Sol n.



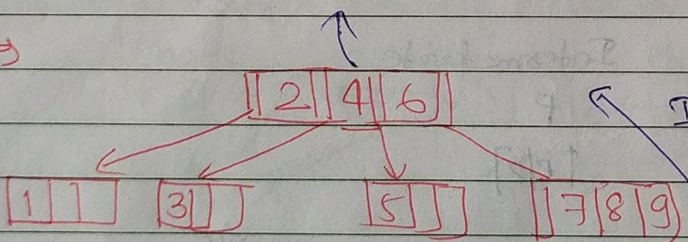


5 > 2

push median upwards if overflow.

and partition into 2 nodes

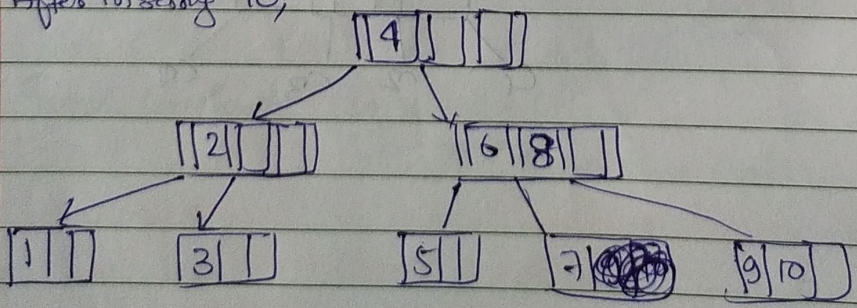
⇒



Insert 8 (overflow) median (4)

(Insert 10) (overflow) median = 8

After inserting 10,



AMAS | Page No.
Date / /

Find order of the B-Tree

Q1 key-size of 10 Bytes
Block size = 512 bytes, data ptr = 8 bytes
Block ptr = 5 bytes

Sol: We know;

Let p = order of tree

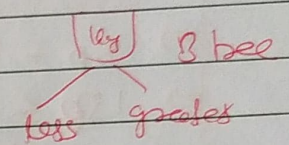
$5p + (p-1) \times 8 + (p-1) 10 \leq 512$

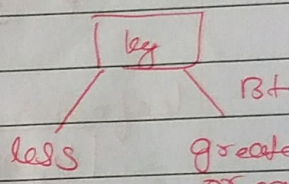
(5p: block, (p-1)×8: data, (p-1)10: key)

$23p \leq 530$

$p \leq \frac{530}{23}$ ≈ 23.04

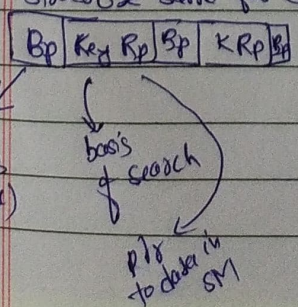
(p = 23) 24th won't fit

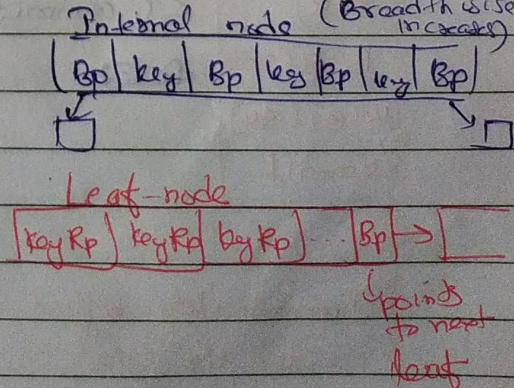




Difference b/w B-Tree and B+-Tree

B-tree	B+ Tree
1) Data is stored in leaf as well as internal nodes.	1) Data is stored only in leaf nodes.
2) Searching is slower, Deletion complex.	2) Searching is faster, deletion is easy (directly from leaf node)
3) Leafs not linked together	3) Leaf linked together like linked list.
4) No redundant keys are present	4) Redundant keys may be present





AMAS | Page No.
Date / /

Order of B+ trees

key size = 10 bytes
B size = 512 bytes
Data = 8 bytes ; block ptr = 5 bytes

Find order of leaf and non-leaf nodes?

Soln: Non-leaf nodes

$Bp \times p + key \times (p-1) \leq 512$

$\Rightarrow 5p + 10(p-1) \leq 512$

$\Rightarrow 15p \leq 522$

$p \leq 34.8 \Rightarrow (p = 34)$ order = no of max childs

leaf nodes = max no of (key, Dp) pairs:

key Dp	key Dp	--	Bp

$p(key_8 + Dp_5) + Bp \leq 512$

$\Rightarrow p(18) + 5 \leq 512$

$18p \leq 507$

$p \leq \frac{507}{18} \Rightarrow p = 28.2 \Rightarrow (p = 28)$ order

Immediate DB modification (Log based recovery)

A = 100
B = 200

T_1	logs
R(A)	<T, start>
A = A + 100	<T, A, 100, 200>
W(A) (200)	<T, B, 200, 400>
R(B)	<T, commit>
B = B + 200	
W(B) (400)	
commit	

Immediately reflect the changes in RAM to the DB

Why B-tree is slower than

If failure occurs:

① → if start and commit both there
→ Do REDO save all data to DB.

② → If commit not done
→ UNDO (save old values to DB, from logs)

* It is also known as UNDO/REDO strategy.

View of total database content is Conceptual view

Data warehouse Supports
(A) OLAP (B) OLTP
(C) OLAP and OLTP (D) operational DB

which of the following does not comes under five V's of Big Data
(a) volume (b) variety (c) velocity (d) visualization
(Extra value & veracity)

Hadoop is framework that works with variety of related tools. Common I/p include.
(A) MapReduce, Hive and HBase

which does not describes Hadoop
(a) open Source (b) Real time (c) Java based
(It is batch processing)
(d) Distributed computing approach

(late)

Deferred database modification (log based recovery)

→ log is a small file which stores all actions performed.

less

T_1	
R(A)	<T_1, start>
A→A+100	<T_1, A, 200>
W(A)	<T_1, B, 400> ← New value
R(B)	<T_1, commit>
B→B+200	
W(B)	
commit	

(stored when commit)

Recovery

If commit done :

→ REDO (Save all new values in DB)

If not committed :

→ Rollback to old values

→ Also known as NO UNDO / REDO method.

Like command in SQL

→ starting from 'A'

→ where name like 'A%';

→ Ending with 'n'

→ where name like '%n';

→ Contains 'ee'

→ where name, like '%ee%';

→ contains 'a' in 2nd place

→ where name like '_a%';

→ name contains 'a' in 7th place and length should be 5 characters

→ where name like '_a____';

AMAS | Page No.
Date / /

PL SQL

① Find the Sum of two numbers

```
declare
a int;
b int;
c int;
begin
a:=&a;          //uses input
b:=&b;
c:=a+b;
dbms_output.put_line('Sum of a and b='||c);
end;
```

② Find greatest of two numbers

```
declare
 a int;
 b int;
 begin
 a:=&a;
 b:=&b;
 if(a>b)
 then
 dbms_output.put_line('a is greater');
 else
  db.
 endif;
 end;
```

AMAS | Page No.
Date / /

For loop

```
Declare
a number(2);
begin
for a in 0..10
loop
  dbms_output.put_line(a);
end loop;
end;
```

While loop;

```
While a<b
loop
  a:=a+1
  .
  .
  .
end loop;
end;
```

Set serveroutput on

Single row functions and multi row functions.

(i) Single row function: Applied to single row and gives single o/p.

(a) Number func: round, trunc, mod

(b) Character func: A. Case manipulation
Upper, lower, initcap

B. Characters manip.
concat, substr, length, Instr, LPAD, RPAD, Trim, Replace
(finding word)

(c) Date functions
→ Months between, Add months, Next Day, Last Day, Round, Trunc

(d) General functions (Dealing with NULL values)
NVL, NVL2, NULLIF, COALESCE, CASE, DECODE (Switch)

(e) Data type Conversion
→ TO_CHAR, TO_NUMBER, TO_DATE

(2) Multirow functions: Applied to multiple rows but gives single op.
e.g Aggregate function.

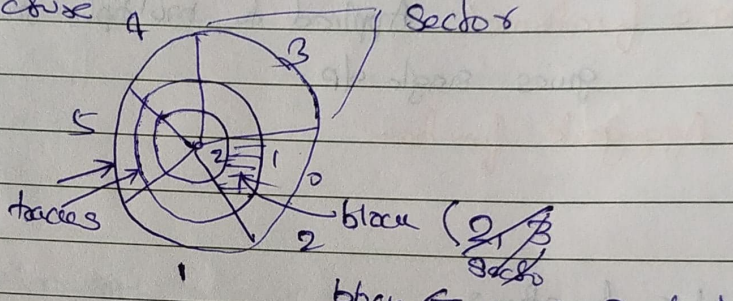
Character functions
(a) Case manip.
(b) character manip
1. Concat('varun', 'Singla')
2. Substr('varun', 2, 4) ⇒ aru
3. Instr('varun', 'u') ⇒ 4
4. Length
5. LPAD('varun', 10, '*') ⇒ *****varun
   (10 = total length; ***** = left pad)
6. RPAD
7. Trim('v' from 'varun') ⇒ arun
8. Replace('varun', 'v', 'T') ⇒ Tarun

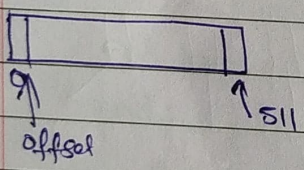
Abdul Wasi

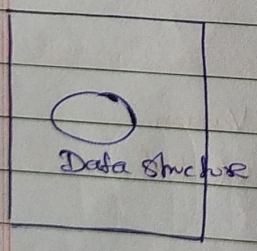
B & B+ Trees

Disk structure
How data is stored on Disk
What is indexing?
What is multilevel indexing?
m-Way Search Trees
B- trees
Insertion & Deletion in B-trees
B+ Trees

Disk Structure







DBMS: Organizing the data in the HD, so that it can be accessed efficiently

Data Structure: Organizing the data in MM, that is directly used by the program.

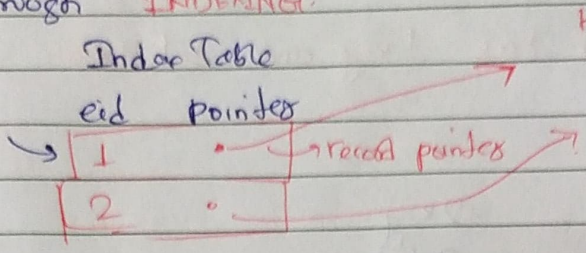
How data is stored in Disks?

→ Data is stored in Blocks
→ No of tuples stored in 1 block = Block Size / Record Size
= 128B 512 / 128 = 4 Nos

No of blocks req'd = Total No of Record / No of records in 1 Block
= 100 / 4 = 25 Blo

Indexing

Reducing the time to access the particular block is done through INDEXING.



⇒ Indices are stored in Hard Disk.

(i) In Dense, No. of records in indices = # of records

Total No. of indices in 1 Block = $\frac{512}{16}$ (block size / index size)

= 32.

No of indices = 100

No. of blocks needed = $\frac{100}{32}$ (Total indices / # indices in 1 Blk.)

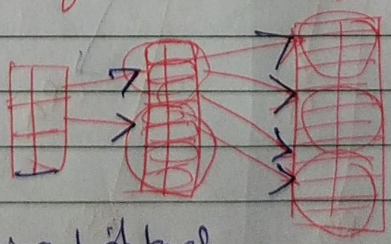
= 3.2 ≅ 4 Blocks

Sq Time = (4+1) (finding Block)

~~Multiple~~ Multilevel Indexing

If searching in an index table also takes a large time (if no of indices is huge).

This is the basis for B-tree usefulness in DBMS



For self-managing high level indexing, we use B-tree.

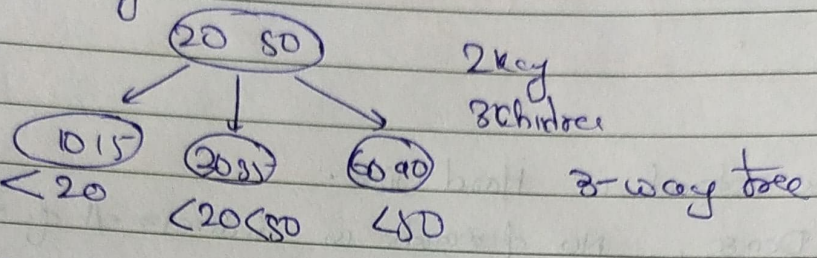
Removal and addition of high level indices based on no of entries is done automatically.

m-Way Search tree

→ Binary search has 1 key and 2 children

In m-Way

e.g.



2 key
3 children

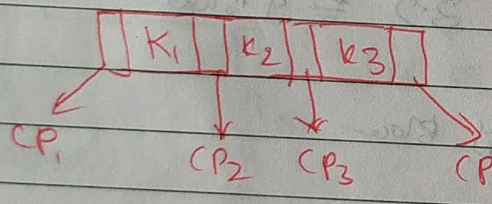
3-way tree

In a m-way tree

→ (m-1) keys and m-children are there

So Binary Search Tree is a subset of m-way tree with m=2.

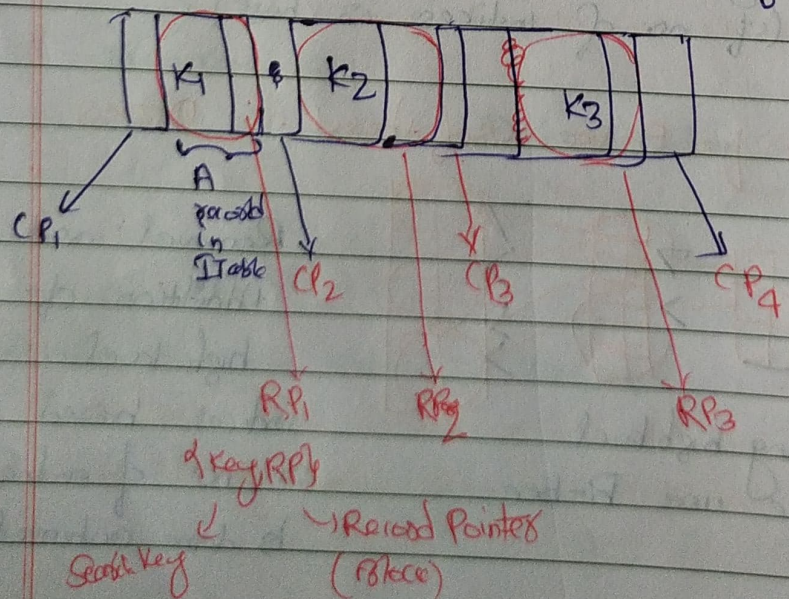
Node-structure for 4-way tree



CP_i → child pointer

k_i → key

Using m-way tree for indexing.



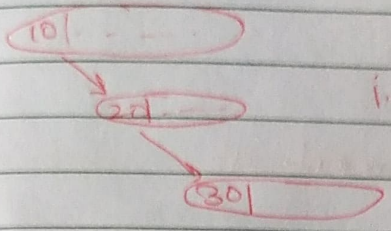
(Key, RP)

↓ Search key

↳ Record Pointer (Block)

Problem → no guidelines

→ There is no control of the creation of m-Way Search Tree
For 10-way Search tree, if keys are 10,20,30



i.e. In worst case, for n keys height will become n; that will result into linear search.

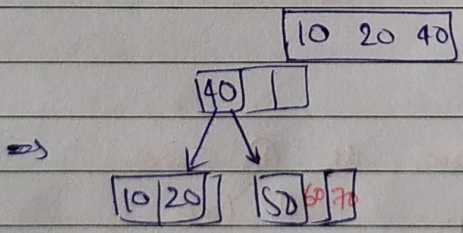
B-trees (m-way ST + Rules)

The guidelines are provided by B-Tree.

Rules:

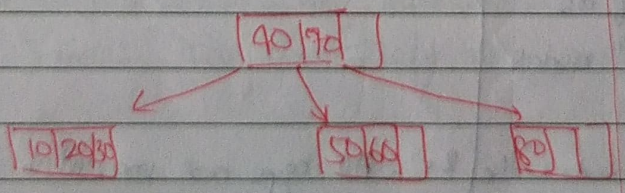
1. Every node must have $\lceil m/2 \rceil$ children, then only child nodes are filled
2. Root can can have min^m 2 children (1 key)
3. All leaf at same level.
4. Bottom-up creation process.

eg B-Tree: m=4 ; keys: 10, 20, 40, 50

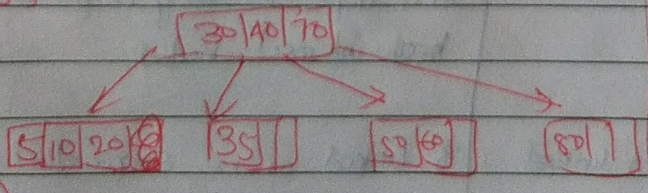


(for 50 no free space)

Insert 80.



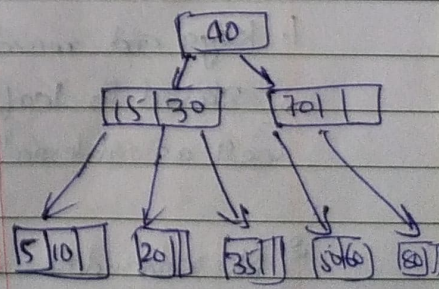
↳ 35



Ins(5)

(15) 20
5 10 15 (left biased)

15 30 40 70



The tree grows upwards.

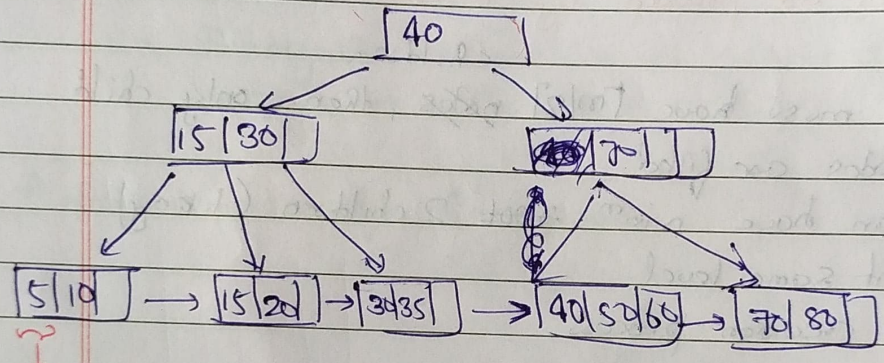
→ Every key or B-tree node has
{key, Rec. pointer}

B^+ Tree

→ Record pointers will only be present at leaf nodes.

→ So, every node will be present in it's ~~child~~ leaf node as well

Rev. example :-



consist of key and Rec. ptr.

Difference b/w B-tree & B^+ Tree

B Tree	B^+ Tree
1. Keys and records are stored in leaf as well as internal nodes.	1. only in leaf nodes.
2. No redundant node value are present.	2. All keys are present in leaf, their duplicates may be present in ~~high~~ internal nodes.
3. leaf nodes are not linked to each other	3. Linked in a sequential manner

B-Tree	B^+ Tree
4. Deletion in B tree is very complex as we need to take care of the child nodes.	4. Deletion is very simple as all keys are present in leaf nodes.

Functional dependency is a constraint that specifies relationships b/w two sets of attributes where one set can accurately determine another set.
Determinant → Dependent

E-R Model → High level data model
→ Conceptual design for the database
Develops a simple and easy view of data.
→ Pictorial representation

Concurrency control protocols
→ Lock Based
(i) Shared exclusive
(ii) 2PL → Growing Shrinking
(iii) Strict 2PL
(iv) Rigorous 2PL
(v) Conservative 2PL
→ Time stamp ordering protocol.

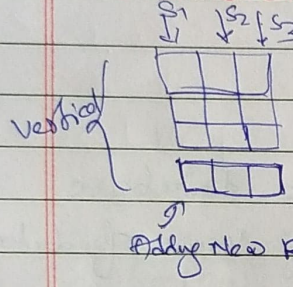
Inner Joins: results in intersection of two tables.
Outer Joins: Full outer, left outer, right outer

AMAS | Page No.
Date / /

Date: 29/08/21

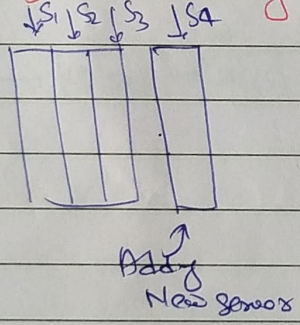
Scaling: We either compress or expand the system to meet the expected need.
Scaling operation can be achieved by adding resources to meet the smaller expection in the current system, or by adding a new system in the existing one, or both.

Vertical Scaling: When new resources are added to the existing system to meet the expectation, it is known as vertical scaling.



Easy and cheaper than horizontal scaling

Horizontal Scaling: When new server racks are added to meet requirements.



Difficult and costlier than Vertical scaling

Sharding: Helps the system to keep data into different resource according to sharding process.
"shards" means "small part of a whole".
It is a type of DB partitioning which makes it more manageable and can be accessed faster.

Features of Sharding.
1. Makes the DB smaller
2. Makes the DB faster
3. DB becomes much more easily manageable
4. Can be a complex operation sometimes
5. Reduces the transaction cost of the DB.

Denormalization: DB optimization technique, in which we add redundant data to one or more tables. Helps us in getting rid of costly join operations.

If tables are large, we may spend an unnecessary long time in performing join.

Under denormalization, we are okay with some redundancy and some effort to update the DB in order to get the efficiency advantages of fewer joins.

Pros:
→ Retrieval time is faster
→ Less bugs because of simple query retrieval.

Cons
→ Insert and update may become costly
→ Data may be inconsistent.
→ Data redundancies necessitates more storage.

Intension and Extension in a DB:

Intension is the permanent part of the relation, comprises of two things relation and relation schema and the constants. Integrity constraints defines the key constraints, referential constraints etc.

Extension: is the snapshot of the system at a particular time. Displays values for tuples in relation at the particular instance of time. Keeps changing with time as tuples are added, deleted, edited.

AMAS | Page No.
Date / /

SQL	NoSQL
1. Rdbms	1. Non-Rdbms
2. Has predefined schema	2. flexible schema
3. Can require upfront preparation	3. Changes can be made at any time
4. SQL dbs are vertically scalable (by increasing resources)	4. Are horizontally scalable.
5. SQL databases are table-based (suitable for multirow transactions)	5. Key-value pairs, document based
6. Follows ACID properties	6. Browses CAP theorem. (Consistency, Availability, Partition tolerance)
7. Support from vendors. e.g. MySQL, PostgreSQL	7. Only community support e.g. MongoDB, cassandra.
8. Not suited.	8. Best suited for hierarchical data storage.

CAP

Consistency: All client programs who are reading data from the cluster see the same data at any given pt. of time.

Availability: If we make a request, we must get a response no matter what happened inside the cluster.

Partition tolerance: Cluster (as a whole) continues to function even if there is a partition b/w two nodes.