Database Management System (DBMS)

—Vishvadeep Gothi playlist

Database:- The collection of data, usually referred to as the database, contains information relevant to an enterprise.

Database Management System:- The database management system (DBMS) is a collection of interrelated data and a set of programs to access those data.

↳ info retrieval, insertion, deletion, updation.

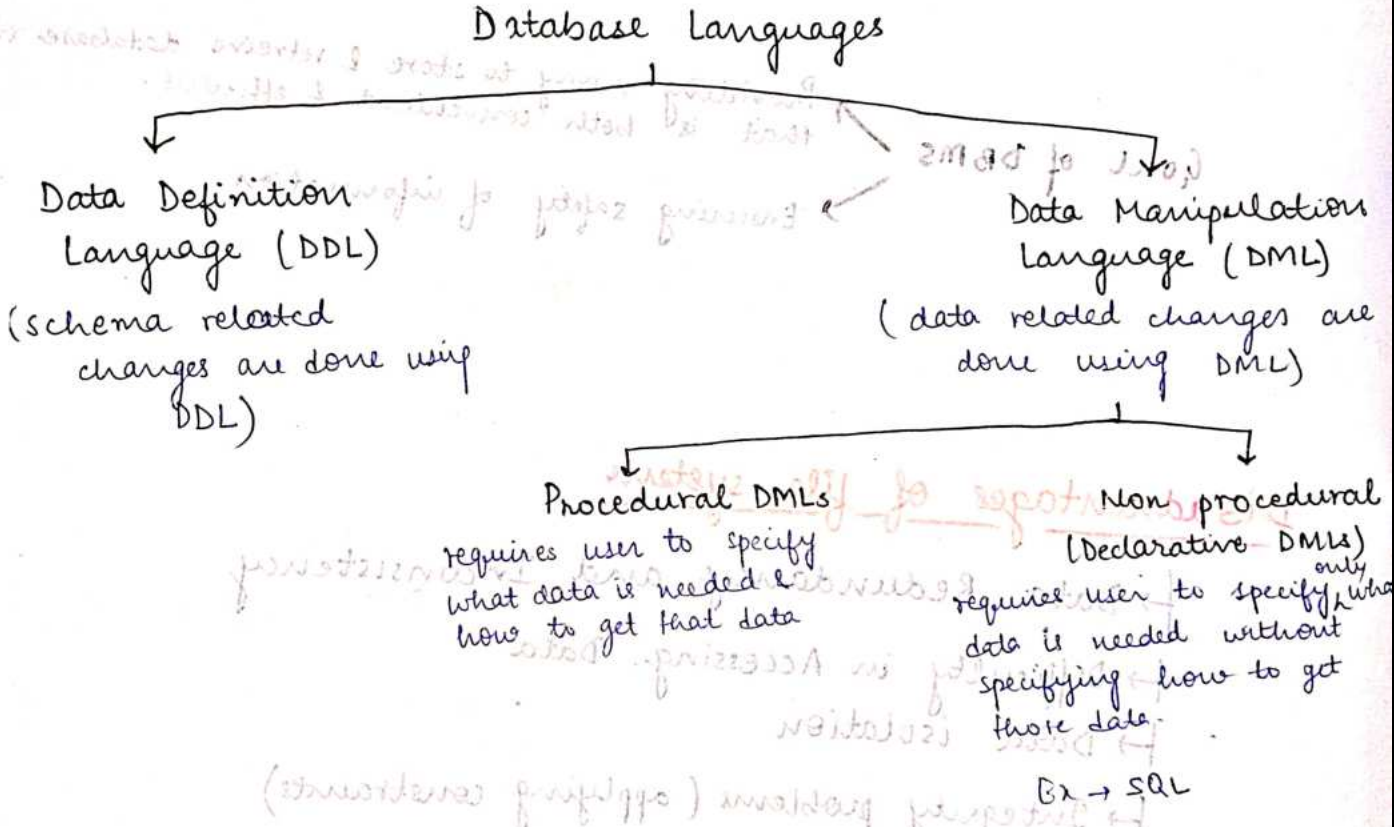
Goal of DBMS
- Providing a way to store & retreive database info that is both convenient & efficient.
- Ensuring safety of information

Disadvantages of file system

- Data Redundancy and Inconsistency
- Difficulty in Accessing Data
- Data isolation
- Integrity problems (applying constraints)
- Atomicity problem
- Concurrent-Access Anomalies
- Security problems

Instance – The collection of information stored in the database at a particular moment is called an instance of database.

Schema – The overall design of the database is called database schema.

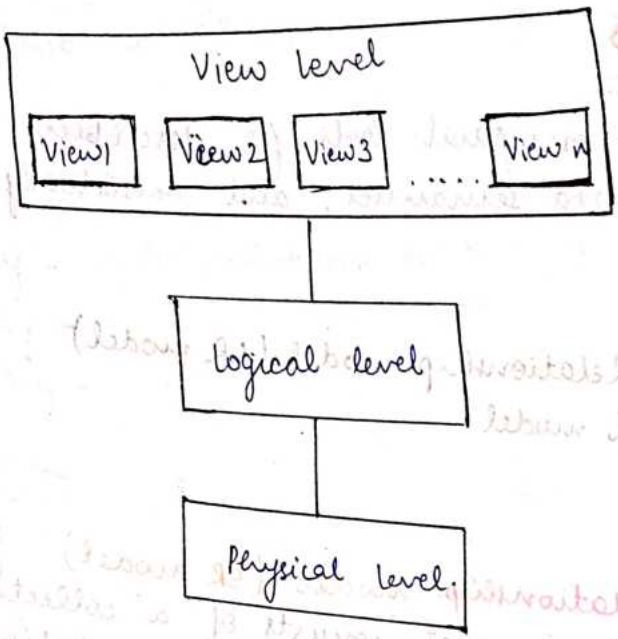


View of data –

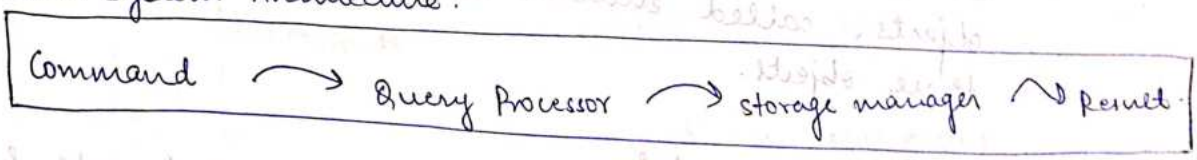
1. **Physical level –** how data is actually stored in physical memory.

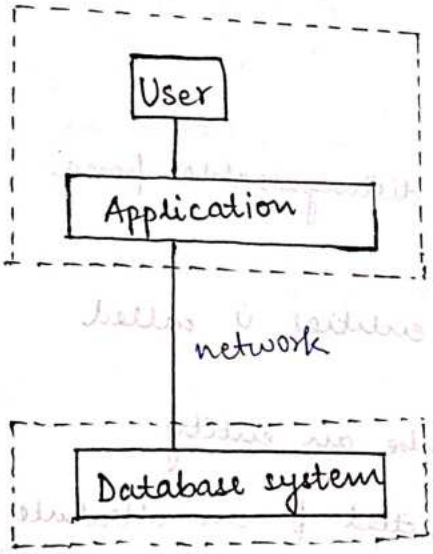
2. **Logical level –** tables, at this level, we see data distributed in the form of tables in database.

3. **View level –** defines which part of database can be viewed by a person.
   manager can view all the tables
   employee can view only his record.

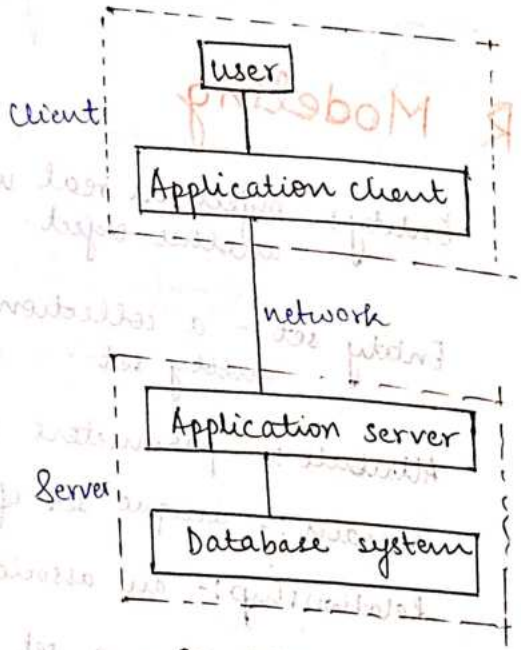


DataBase System Architecture.





2 tier architecture.



3 tier architecture

Data Models

A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

2 models
- Entity-Relationship model (ER model)
- Relational model

1. Entity Relationship model (ER model)
   ER ~~Reto~~ data model consists of a collection of basic objects, called entities, and of relationships among these objects.

2. Relational model
   uses a collection of tables to represent both data & relationships.

E-R Modeling

Entity :- object in real world that is distinguishable from another object.

Entity set :- a collection of similar entities is called entity set.

Attribute :- parameters which describe an entity

Domain :- unique set of values permitted for an attribute

Relationship :- an association among two or more entities

Relationship-set - a set of similar relationships

Key - an attribute or a set of attributes whose values can uniquely identify an entity in a set.

Descriptive attribute - attribute of a relationship.

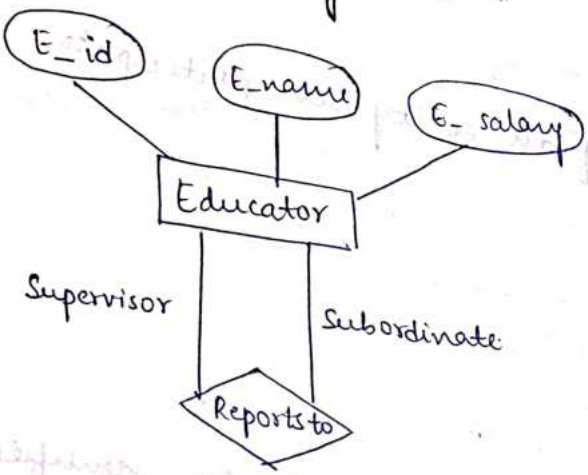
Entity set → 

Relationship → 

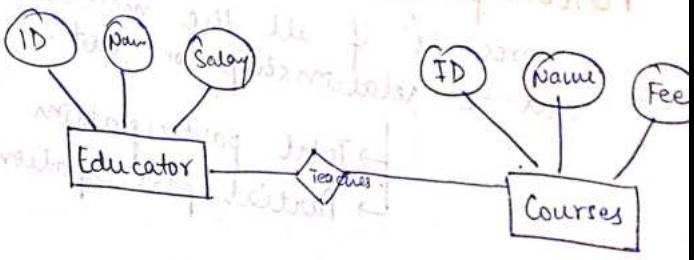
attribute → 

Types of Relationships –

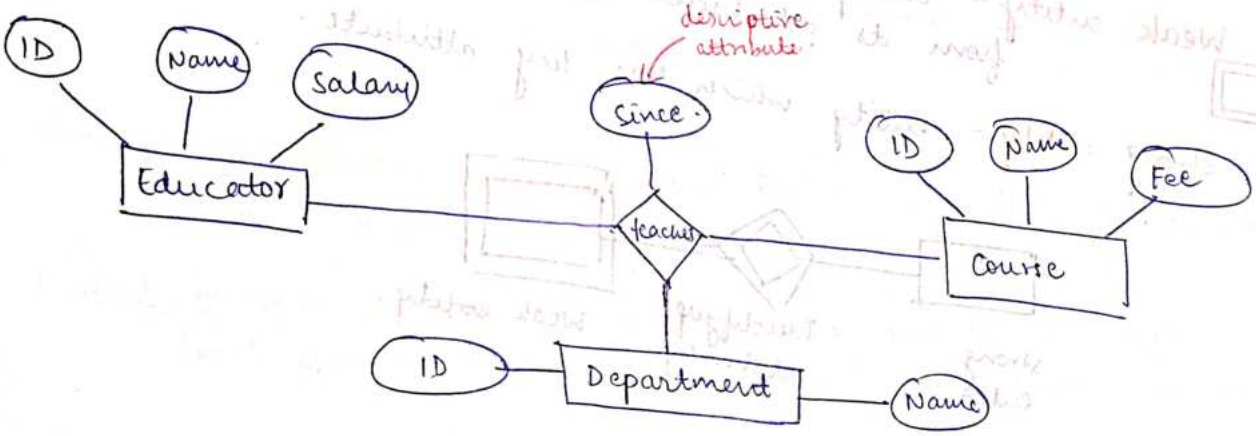
1. Unary – Relationship b/w entities of same entity set.
2. Binary – " " " 2 entity sets.
3. Ternary – " " " 3 entity sets.



Unary relationship.



binary relationship.



ternary relationship.

Mapping Cardinality

The maximum number of relationships in which an entity can participate is called mapping cardinality.

- One to one relationship
- One to many relationship
- Many to one relationship
- Many to many relationship

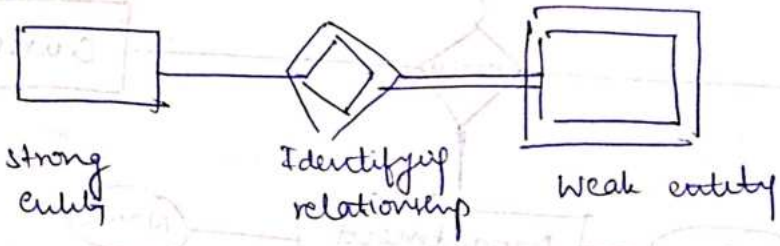
Participation constraints

represents if all the members of an entity-set participate in a relationship or not.

- Total participation
- Partial participation

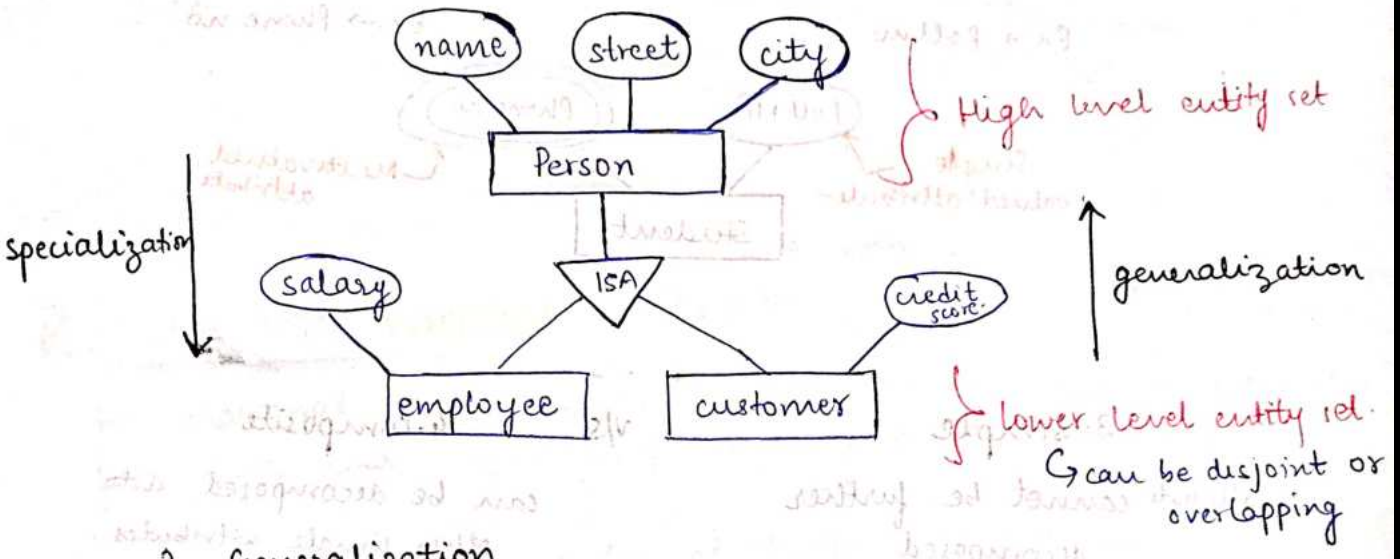
Weak entity - entity which cannot be uniquely identified from its attributes alone.

Strong entity - entity which has key attribute.



Extended ER Features

1. Specialization
   The process of designating subgroupings within an entity set is called specialization

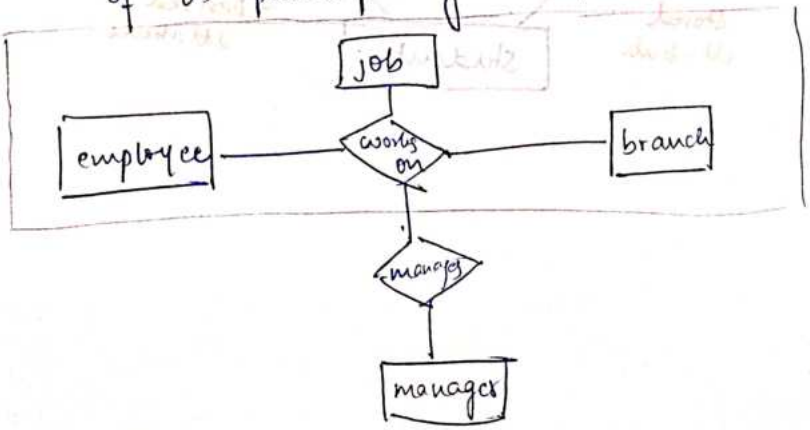


2. Generalization
   Commonality can be expressed using generalization.
   containment relationship b/w higher level entity set & lower level entity set.

Total generalization / specialization – Each member in higher level entity set must belong to lower level entity set.

Partial generalization / specialization – Some members in higher level entity set may not belong to lower level entity set

3. Aggregation
   Relationship that contains another relationship as one of its participating entity



Types of attributes

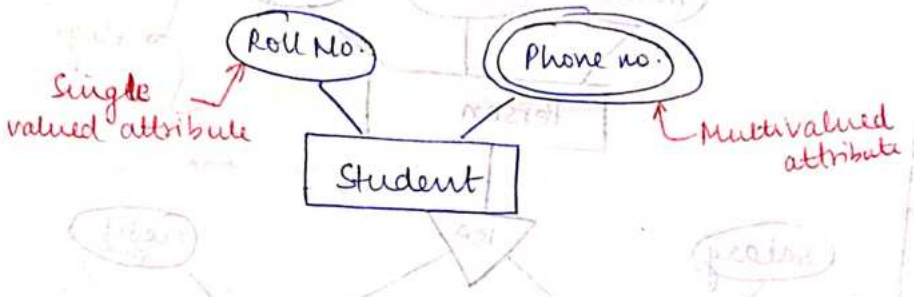
1. Single valued attributes can have only one value.

Ex → Roll no.

V/S

2. Multi valued attributes can have more than one value.

Ex → Phone no.



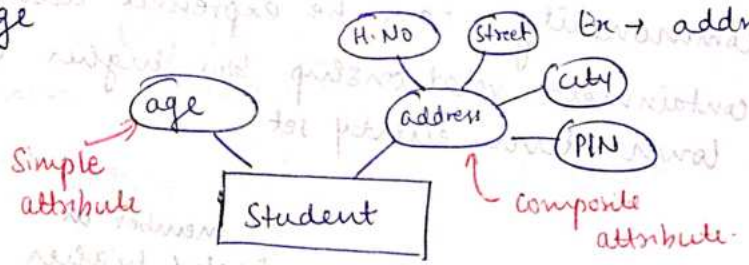
3. Simple attribute cannot be further decomposed

Ex → age

V/S

4. Composite can be decomposed into other simple attributes.

Ex → address.



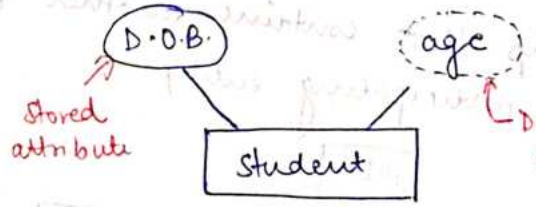
5. Stored attributes whose value is actually stored in memory.

Ex → D.O.B.

V/S

6. Derived attributes which can be derived from a given attribute.

Ex → age



7. Prime attribute which is part of some key

v/s

8. Non prime attribute attribute which is not part of some key.

9. Complex attributes — Multivalued + Composite

Ex → address, phone no.

Relational model

The relational model uses a collection of tables to represent both data and relationships among data

Database schema → logical design of database.

Relation_name (attribute1, attribute2 ..... attributen)

UNIQUE
NOT NULL

Domain constraint → specifies the important condition that we want each relation to satisfy.

Degree or Arity → number of attributes in a relation

Cardinality → number of tuples in a relation

NULL → representation of no value

Keys :- attribute / group of attributes that can uniquely identify a tuple in a relation

Types of keys —

1. Super key - All possible keys of a relation. superset of all keys.

2. Candidate key - Minimal superkey is called candidate key.
   → A key whose proper subset is not a key.

3. Primary key -
   chosen from candidate key for implementation

4. Alternate key -
   all candidate keys other than primary key

5. Foreign key -
   key referring to some value in other table
   can be NULL

Referential Integrity

The value in the foreign key should be present in the column of other table it is referring to

The foreign key should always refer to primary key

Functional Dependencies

Belief $A \rightarrow B$ (B is functionally dependent on A)
if each value in A is associated with exactly one value in B in the relation.

⊙ A functional dependency $\alpha \rightarrow \beta$ always holds if α is the key to the relation R.

$\alpha, \beta \rightarrow$ attributes / set of all attributes in R.

Closure of an attribute -
set of attributes that can be derived from the given attribute.

Ex → R(A, B, C, D)
FDs → { A → B , A → D , B → C }

$A^+ = \{A, B, C, D\}$
$B^+ = \{B, C\}$
$C^+ = \{C\}$
$D^+ = \{D\}$

Ex → R(A, B, C, D, E)
FDs → { A → B, B → C, C → D, D → E, E →

$A^+ = \{A, B, C, D, E\}$
$B^+ = \{A, B, C, D, E\}$
$C^+ = \{A, B, C, D, E\}$
$D^+ = \{D, E, A, B, C\}$
$E^+ = \{E, A, B, C, D\}$

→ Trivial functional dependencies.
These functional dependencies are satisfied by all relations

Ex → $A \rightarrow A$
$\alpha \rightarrow \beta$ if $\beta \subseteq \alpha$

Armstrong's axioms:

↳ Reflexivity rule
$\alpha \rightarrow \beta$ holds if $\beta \subseteq \alpha$

↳ Augmentation Rule
if $\alpha \rightarrow \beta$ holds, then $\gamma\alpha \rightarrow \gamma\beta$ also holds

↳ Transitivity Rule
if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ holds, then, $\alpha \rightarrow \gamma$ also holds.

Additional Rules

↳ Union Rules –

if $\alpha \to \beta$ and $\alpha \to \gamma$ hold, then,

$\alpha \to \beta\gamma$ also holds.

↳ Decomposition Rule –

if $\alpha \to \beta\gamma$ holds, then,

$\alpha \to \beta$ and $\alpha \to \gamma$ also hold.

↳ Pseudo transitivity rule.

if $\alpha \to \beta$ and $\gamma\beta \to \delta$, then,

$\alpha\gamma \to \delta$ also holds.

Closure of a set of functional dependencies.

Finding all possible FDs from given set of FDs

Ex→ R (A, B, C, D)

FDs = {A → B, B → C, AB → D}

F^+ = {A → C, A → D}

transitive rule ↩ ↪ pseudo transitive rule

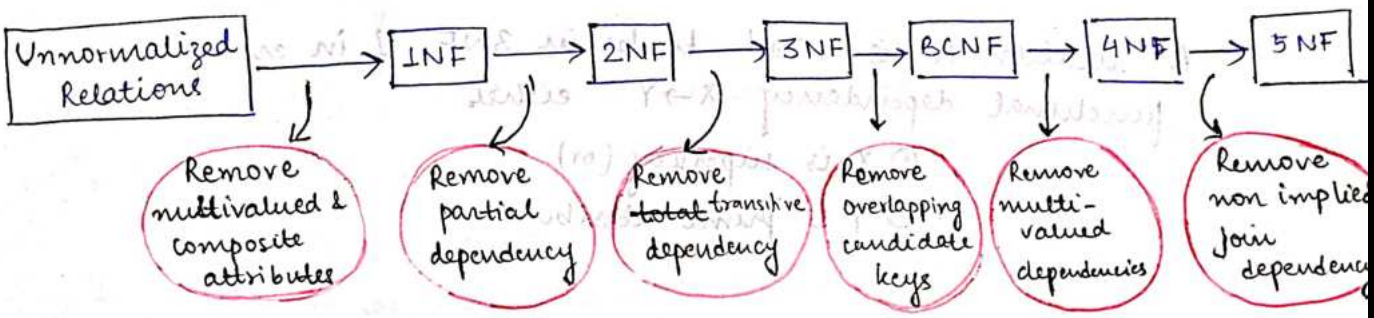
NORMALIZATION

Normalization is the process of minimizing redundancy from a relation or a set of relations

→ can cause insert, delete, update anomalies.

⊙ Process of grouping attributes into well structured relations, that contain minimum redundancy.

Process of normalization —



First Normal Form (1NF)

A relation R is said to be in 1NF if there is no any multivalued & composite attribute

Second Normal Form (2NF)

→ Relation is in 1NF

→ no non-prime attribute in R should be partially dependent on R prime attribute.

Partial dependency :-

$X \to Y$ is partial dependency if some attribute $A \in X$ can be removed and the dependency still holds.

→ If candidate key contains more than one attribute, then a part of candidate key should not derive anything else.

* No partial dependency if candidate keys contains only one prime attribute.

Third normal form (3NF)

→ Relation R should be in 2NF

→ No any non prime attribute in R should be transitively dependent on key of R.

A relation R is said to be in 3NF if in every functional dependency $X \to Y$ either

- ⊙ X is superkey (or)
- ⊙ Y is prime attribute

Boyce Codd Normal Form (BCNF)

→ Relation R should be in 3NF

→ for every functional dependency $\alpha \to \beta$, α should be superkey.

* Functional dependency may not be preserved

* Any relation with 2 attributes is always into BCNF

* Normal form considered adequate for normal database design

Ans → 3NF.

↳ bcoz dependency preserving lossless guarantees.

Lossy Decomposition

The decomposition of relation R into R1, R2, R3, ... Rn is lossy when the join of $R_1, R_2, \ldots R_n$ does not produce same relation as in R.

< No. of tuples ~~inc~~ are increased >

If no common attribute is present, then, the decomposition is lossless.

$R \rightarrow R1, R2, \ldots Rn$

$R = R_1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n$ — Lossless Decomposition

$R \subset R1 \bowtie R_2 \bowtie R_3 \ldots \bowtie R_n$ — Lossy Decomposition

★ ~~Dep~~ Decomposition is lossless when the common attribute is a key in one of the sub-relations.

★ Lossless decomposition is ~~guo~~ ~~go~~ guaranteed upto 3NF. In BCNF, ~~we~~ ~~to~~ sometimes, ~~we~~ it is possible to satisfy only ~~dec~~ lossless decomposition property or functional dependency preservation property.

We focus on making decomposition lossless.

∴ Lossless decomposition holds in BCNF also.

When table is split into >2 parts Chase algorithm is used to check if decomposition is lossless or not

Dependency Preserving Decomposition

A decomposition $D = \{R_1\ R_2\ R_3, \ldots Rn\}$ of R is dependency preserving with respect to a set F of functional dependency if

$$(F_1 \cup F_2 \cup F_3 \ldots \cup F_n)^+ = F^+$$

not ensured in BCNF

SQL : Structured Query Language

- Domain specific language
- used for managing data in RDBMS

→ SQL Data types –

CHAR (size) → a fixed length string (can contain letters, numbers, special characters)
0 < size < 255
Default → 1.

VARCHAR (size) → variable length string
Size defines the max. length of the string
0 < size < 65535

Bit (size) → a bit value type

BOOL → 0 is considered false, non zero value is tr

INT (size) → medium integer

BIGINT (size) → large integer

FLOAT (size, d) → floating point no.
d → no. of digits after decimal point.
size → total no. of digits.

FLOAT(p) → floating point no.
p is used to determine whether to use FLOAT or DO
0 < p < 24 → float
25 < p < 53 → double

DOUBLE (size, d)

DECIMAL (size, d) → an exact fixed point no.

★ SQL is not case sensitive

1. SELECT COMMAND

used to retreive data from one or more tables.

SELECT <column-name> FROM <table_name> ;

★ returns duplicates if present.

SELECT * FROM <table-name> → returns whole table.

SELECT DISTINCT ~~name~~ country FROM customers.

↳ only returns unique records.
always used only with SELECT

SELECT DISTINCT country, postal FROM customers

distinct combination of the 2 fields

2. WHERE Clause

- used with SELECT, UPDATE, DELETE & INSERT commands.
- used to filter specific rows from the table

SELECT * <column name> FROM <table_name>
WHERE <condition>

- Relational operators that can be used in WHERE clause —

  =, <>, <, <=, >, >=

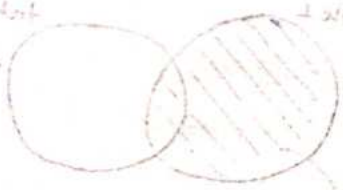
- Logical operators —

  AND, OR, NOT

- BETWEEN operator —

  used to filter the records in specific range.

  BETWEEN LB and UB (both inclusive)



SQL Joins

- needed to retreive records from more than one table collectively
- 2 tables must have one common column.

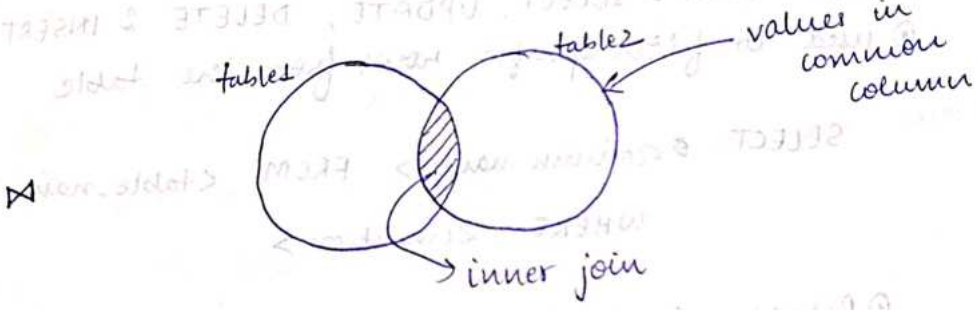
Types of joins —

- Full Join (Full outer join)
- Left Join (Left outer join)
- Right Join (Right outer join)
- Inner Join (Equi Join or Simple Join)
- Self Join

No. of attributes in result = Deg (table1) + Deg (table2)

1. Inner join

SELECT * FROM <table1> INNER JOIN <table 2> ON <table1>.<column> = <table2>.<column>



$\bowtie$

equivalent to

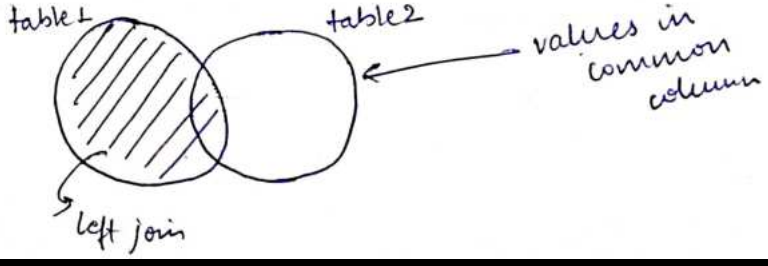
SELECT * FROM <table1>, <table2> WHERE <table1><column> = <table2><column>

2. Left Join

SELECT * FROM <table1> LEFT JOIN <table2> ON <table1>.<column> = <table2>.column.

$\bowtie$

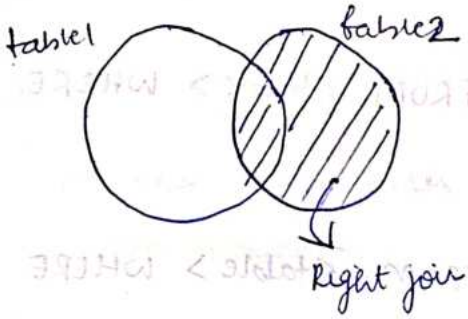
common tuples + all tuples from table1



3. Right join

SELECT * FROM <table1> ~~<column>~~ RIGHT JOIN <table2> ON <table1>.<column> = <table2>.<column>

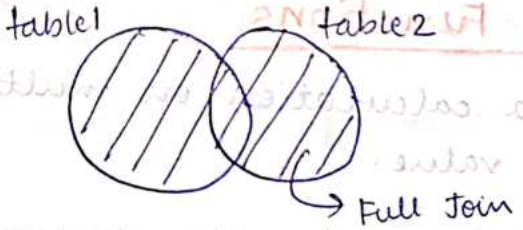
common tuples + all tuples from table 2.



4. Full join

SELECT * FROM <table1> FULL JOIN <table2>
ON <table1>.<column> = <table2>.<column>

common tuples + tuples from table1 + tuples from table2



5. Self join

only 1 table is involved in this join

self join is not keyword

∴ it is implemented using alias & WHERE clause

SELECT * FROM <table1> AS <t1> , <table1> AS <t2>
WHERE ~~<table1>~~ <t1>.<column1> = <t2>.<column2>

LIMIT clause

used to limit the number of fetched records from a huge database table.

SELECT * FROM <table> WHERE <condition> LIMIT <no>

no. of records needed.

imp

* SELECT * FROM <table> WHERE <condition> LIMIT m, n

- The query returns max ^n records after leaving first 'm' records
- leave first m records
- return n records

Aggregate Functions

- Performs a calculation on multiple values and returns a single value.
- * can only be used with SELECT clause

→ MIN(<column>) — returns min value in column
→ MAX(<column>) — returns max. value in column
→ SUM(<column>) — returns sum of values in column
→ COUNT(<column>) → returns no. of records ~~least~~
→ AVG(<column>) → returns average of values in column

all aggregate functions ~~give~~ ignore NULL values except count(*)

A	B
1	N
2	N
N	N
3	N

count(*) = 4
Count(A) = 3
count(B) = 0

SUM(A) = 6
AVG(A) = 2
MAX(A) = 3
MIN(A) = 1

SUM(B) = NULL
AVG(B) = NULL
MAX(B) = NULL
MIN(B) = NULL

IN operator

- The IN operator allows you to specify multiple values in a WHERE clause
- IN operator is shorthand for ~~must~~ multiple OR conditions

```
SELECT * FROM <table> WHERE <column> IN (value1, value2...
```

- IN operators is ~~also~~ used with subqueries.

ALIAS

- used to give temporary name to a table, or a column of a table.
- only exists during duration of that query.
- created using AS.

[space is not allowed in column name of a table.
To print space, ALIAS can be used]

```
SELECT CustomerName AS [Customer Name], FROM Customers.
```

GROUP BY Clause

- used in collaboration with SELECT statement to arrange identical data into groups.
- all attributes used in GROUP BY clause must be present in the SELECT clause.
- attributes not present in GROUP BY clause can appear in aggregate functions.

```
SELECT DNo., AVG(salary) FROM Employees
GROUP BY DNo;
```

HAVING clause

similar to WHERE clause.

can only be used with GROUP BY clause.

SELECT DNO, AVG(salary) FROM Employees
GROUP BY DNO HAVING ~~Salary~~ AVG(salary) > 50.

Order of execution → WHERE ⟶ GROUP BY ⟶ HAVING

WHERE	V/S	HAVING
checks the condition on individual rows		checks the condition on a group of rows.
cannot be used with aggregate function		can be used with aggregate functions
executed before GROUP BY clause		executed after GROUP BY clause.

Set operators

Set operators are used to combine results of two or more SELECT statements

no. of columns returned from the individual queries must be equal.

(SELECT * FROM <table1>) <set operator> (SELECT * FROM <table2>)

- UNION (duplicate rows are eliminated)
- UNION ALL (duplicate rows are retained)
- INTERSECT (selects only common rows)
- MINUS / EXCEPT (set difference)

ORDER BY

- ORDER BY clause is used to sort the result set in ascending or descending order.

SELECT * FROM <table_name> ORDER BY <column_name> ASC/DESC
(default)

- default order of sorting → ascending order
- ORDER BY clause is executed in the very end.

Subquery

- also called inner query, nested query
- query within a SQL query that is embedded within the WHERE clause

2 types of subquery
- Single Row subquery
- Multiple Row subquery.

~~1. Single Row subquery~~

* Product with second highest price

```
SELECT MAX(price) FROM ~~Emp~~ Products
WHERE price < (SELECT max(price) FROM Products)
```

* Product with third highest price

```
SELECT MAX(price) FROM Products
WHERE price < (SELECT max(price) FROM products
WHERE price < (SELECT MAX(price) FROM products));
```

Relational operators can be used in single row subqueries.

Multiple Row subquery

Operators used in Multiple Row subquery

1. IN operator

- allows to specify multiple values in WHERE clause.
- shorthand for logical OR.

```
SELECT * FROM Customers WHERE country IN
(SELECT country FROM suppliers);
```

* Find all customers who have placed more than 2 orders.

2 tables Customers, Orders

```
SELECT * FROM Customers WHERE customerID IN
(SELECT customerID FROM orders
GROUP BY customerID HAVING count(customerID)>2);
```

2. ANY operator

```
SELECT <column_name(s)>
FROM <table_name>
WHERE <column_name> <operator> ANY
( subquery );
```

<operator> → any standard comparison operator (= <> <= >= < >).

3. ALL operator

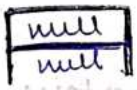
```
SELECT <column_name(s)>
FROM <table_name>
WHERE <column_name> <operator> ALL
( subquery ).
```

4. EXISTS operator

- used to test for the existance of any record in a subquery.
- returns true if subquery returns one or more records.

```
SELECT <column_name(s)>
FROM <table_name>
WHERE ~~<column_name>~~ EXISTS
(subquery)
```

* 'SELECT NULL' returns 1 record with column & row Nu



RELATIONAL ALGEBRA

- Relational algebra is a procedural query language which takes instances of relations as i/p & yeilds instances of relations as o/p
- provides theoretical foundation for relational databases

Basic operators —

- Select (σ)
- Project (Π)
- Union ($\cup$)
- Set Difference ($-$)
- Intersection ($\cap$)
- Cartesian Product ($\times$)
- Rename (ρ)
- Division ($\div$)

1. Selection operator

→ fetches tuples which satisfy a condition

$$\sigma_{<conditions>}(<Relation\_name>)$$

→ similar to WHERE clause of SQL.

⊙ Selection operator is commutative.

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

$$= \sigma_{<cond1> \wedge <cond2>}(R).$$

2. Projection operator

→ used to select only certain columns of the table.

$$\pi_{<column\_name(s)>}(R)$$

* eliminates duplicates in result.

→ similar to ~~SELECT~~ ~~statement~~ SELECT DISTINCT statement.

3. Set operations

⊙ The 2 relations on which set operations are applied should necessarily have similar data types of tuples.

⊙ they should be union compatible

* duplicates are not present.

Commutative & associative:
- i) Union operator (∪)
- ii) Intersection operator (∩)
- iii) Set Difference (-)

* $\pi_x(R_1 \cup R_2) = \pi_x(R_1) \cup \pi_x(R_2)$
* $\pi_x(R_1 \cap R_2) \neq \pi_x(R_1) \cap \pi_x(R_2)$

4. Cartesian Product

cross product

$|R| = m$
$|S| = n$

$|R \times S| = mn$

$Deg(R \times S) = Deg(R) + Deg(S)$

→ Joins

i) Conditional join

$R_1 \bowtie_{<condition>} R_2$

ii) Equi join

$R_1 \bowtie_{R.c = S.c} S$

iii) Natural join

Both relations should have atleast one common column name

$R \bowtie S$

iv) Left outer join

$R \text{⟕} S$

v) Right outer join

$R \text{⟖} S$

vi) Full outer join

$R \text{⟗} S$

5. Rename operator

- similar to aliasing in SQL.
- used to give temporary name to relation / attribute

$\rho_{<newname>}(<Relation>)$ → Renames relation

$\rho_{R(A,B,C)}(R)$ → Renames columns to A, B, C

6. Division operator

⊙ R1 ÷ R2

⊙ Attribute set of $R_2 \subseteq$ Attribute set of R1.

$$\text{Result set} = (\text{Attribute set of } R_1) - (\text{Attribute set of } R_2)$$

R_1

A	B	C

R_2

A	B

$R_1 \div R_2 \rightarrow$ Those values of C which are associated with every value of A, B in R_2

<used in those queries which contain 'for all' >

TRANSACTION

Logic unit of database which includes one or more database operations

↳ Commit - command used to make the changes permanant in a database. (operation successful)

↳ Rollback - command used to revert back to the original state of database (operation unsuccessful)

Database state becomes same as it was on last commit

ACID Property of Transaction

1. Atomicity:- All or none. (either complete transaction or never start it)

2. Consistency:- the database state must be consistent

3. Isolation:- transactions should execute such that it does not affect nor is affected by any other concurrent transaction

4. Durability:- changes made should be reflected back into database (permanantly)

Schedule-

Execution order of multiple transactions on a same database concurrently is called schedule.

Why concurrency?

1. Improved throughput (No. of lines executed per unit time)
2. Resource utilization.
3. ~~Increase in~~ Reduced waiting time.

Problems with concurrency -

1. Recoverability problems
2. Deadlock
3. Serializability issues.

Serializability problems

- ↳ Lost update problem
- ↳ Dirty Read problem
- ↳ Phantom tuple
- ↳ Unrepeatable Read problem
- ↳ Incorrect summary problem.

Good schedule - concurrent schedules which provide consistent result.

Bad schedule - These schedules do not give consistent result (suffer from one or more of the above problems)

Serial schedules

- schedules which do not have interleaving
- always give consistent results
- For n transactions, no. of serial schedules = $n!$

Non serial schedules

- schedules which have interleaving are called non serial schedule.

$T_1 \rightarrow n_1$ steps
$T_2 \rightarrow n_2$ steps
⋮
$T_n \rightarrow n_n$ steps

No. of schedules possible $= \dfrac{(n_1+n_2+n_3+\ldots n_n)!}{n_1!\, n_2!\, n_3! \ldots n_n!}$

Serializable schedule

- A concurrent schedule which provides result as a serial schedule is called serializable schedule.

Serializability
- Conflict Serializability
- View Serializability

1. Conflict Serializability

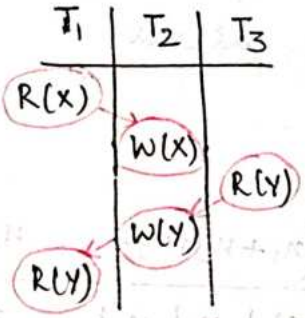
→ 2 DB statements have conflict if:
- Both are in different transactions
- Both are accessing same data item
- One of them is write operation

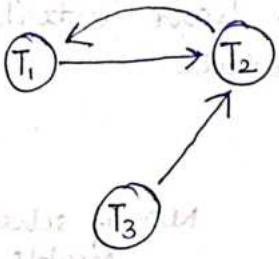
→ 2 transactions are conflict equivalent if the conflicting statements appear in the same order in both of them.

→ If one of the 2 transactions is serial schedule, the other one is called conflict serializable schedule.

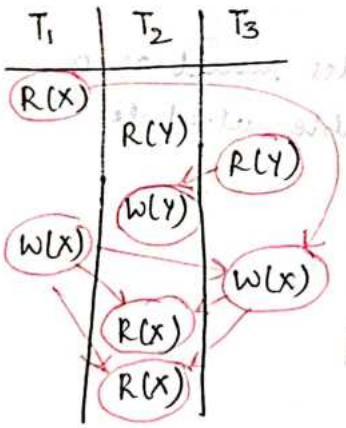
To prove if a given schedule is conflict serializable, we use **precedance graph**.

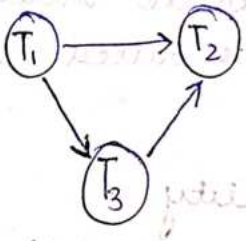
↳ vertices → transactions
↳ edges → conflicts
↳ directed graph.





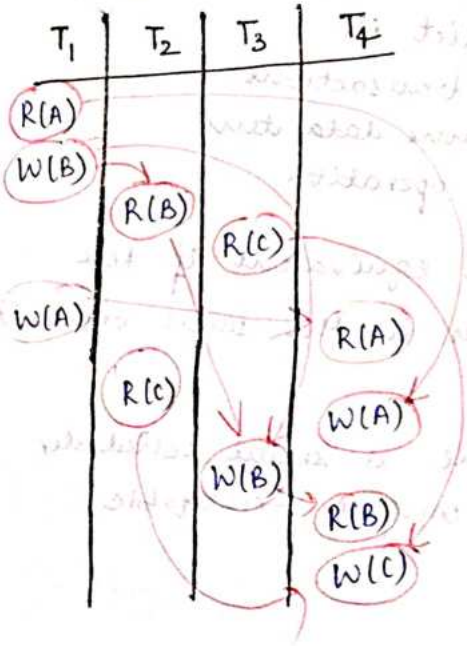
∵ cycle is present in precedance graph,
∴ given schedule is not conflict serializable.

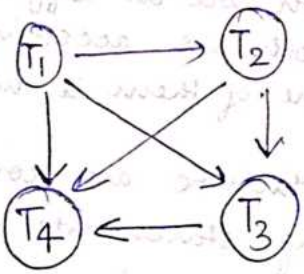




no cycle
∴ conflict serializable schedule

eg. serial schedule = T_1 T_3 T_2





conflict serializable schedule

~~eg conflict serial s~~

eg. serial schedule —

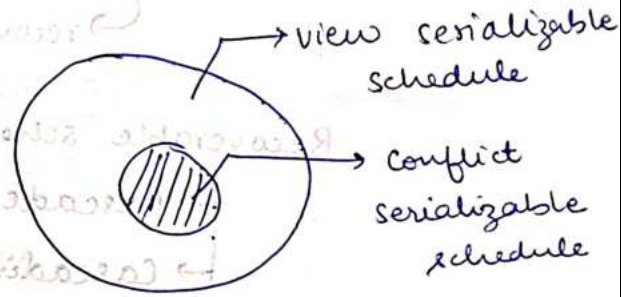
T_1 T_3 T_4 T_2
T_1 T_3 T_2 T_4
T_1 T_2 T_3 T_4
T_2 T_1 T_3 T_4

2. View Serializability

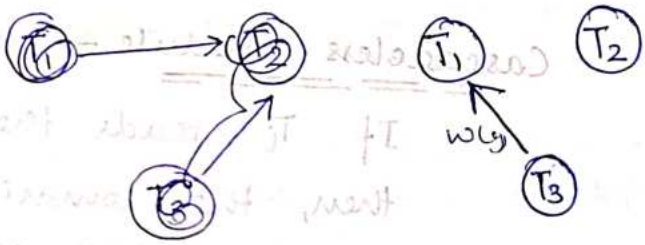
3 points to remember
- ↦ who reads from database
- ↦ who reads from other's written value
- ↦ who writes last

⊙ 2 schedules S_1 & S_2 are view equivalent if both are following similar sequence / rules of view serializability.

⊙ All conflict serializable schedules are view serializable.



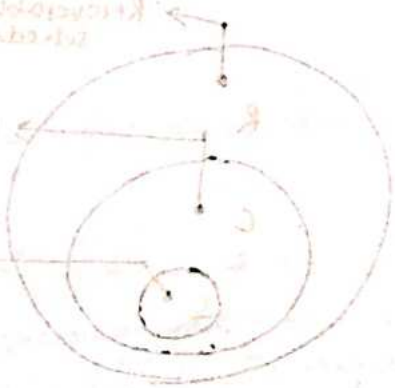
T_1	T_2	T_3
R(X)		
	R(Y)	
		W(Y)
W(Y)		
	W(X)	



not view serializable.

T_3 T_1 T_2
T_3 T_2 T_1
T_2 T_3 T_1

} 3 possible serial schedules



Recoverability

⊙ When no any committed transaction needs to roll back, it is called recoverable schedule.

⊙ If a transaction T_i reads dirty value written by T_j ~~and~~ then, the commit operation of T_i should be after the transaction T_j commits.

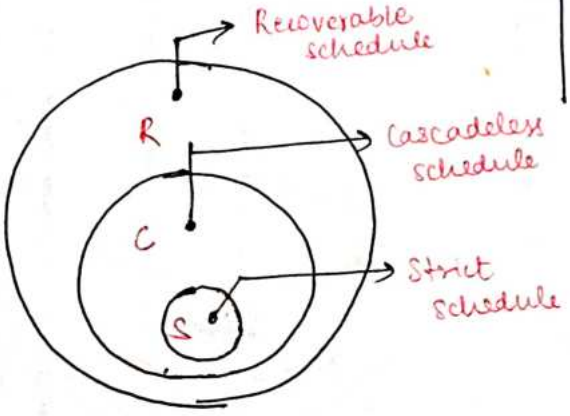
↳ recoverable schedule.

Recoverable schedule
- ↳ Cascadeless Recoverable ~~schedule~~ Rollback
- ↳ Cascading Recoverable ~~schedule~~ Rollback

Cascadeless schedule –

If T_i reads the data item written by T_j then, the commit operation of T_j should appear before read operation of T_i

T_j	T_i
R(x)	
X = X+2	
W(x)	
commit	
	R(x)
	X = X+3
	W(x)
	commit



Obtaining serializability (view serializable & conflict serializable) is ~~also~~ not practically possible
∴ locking protocols are used.

Locking Protocols

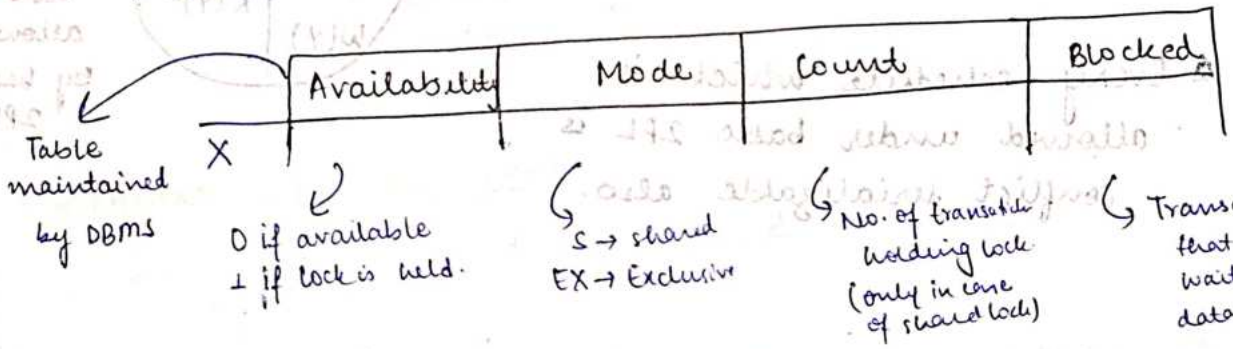
Lock → Shared lock (other transactions can share) taken for read
Lock → Exclusive lock (no other transaction can read/write) taken for write.

Demanded / currently held ↓	Shared	Exclusive
Shared	✓	×
Exclusive	×	×

* A transaction may suffer from starvation if it demands exclusive lock while shared lock is being held by multiple transactions.

* If a transaction is blocked for exclusive lock because other transactions have shared lock, then multiple shared locks are not allowed.

How to avoid starvation?



→ Simple Locking mechanism can lead to unrepeatable read problem.

T_1	T_2
Lock_S(x)	
R(x)	
Unlock(x)	
	Lock_X(x)
	W(x)
	Unlock(x)
Lock_S(x)	
R(x)	

∴ To solve unrepeatable read problem, several protocols / ~~need~~ rules are developed.

2 Phase Locking Protocols (~~2PL~~ Basic 2PL)

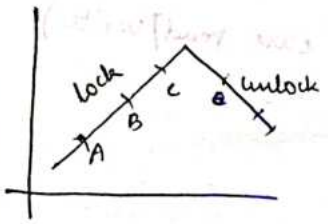
→ Systematic Locking mechanism.

→ Once unlock is done, a transaction is not allowed to lock any other data item.

Each transaction has 2 phases —

↳ Growing phase – acquiring of locks

↳ Shrinking phase – releasing of locks.



T_1	T_2
LOCK_S(X)	
R(X)	
LOCK_EX(Y)	
Unlock(X)	
	LOCK_EX(X)
	W(X)
W(Y)	
Unlock(Y)	
	LOCK_S(Y)
	R(Y)
	Unlock(X)
	Unlock(Y)

∴ Only 2 possible schedules are there —

T_1	T_2
R(X)	
	W(X)
W(Y)	
	R(Y)

OR

T_1	T_2
R(X)	
W(Y)	
	W(X)
	W(Y)

T_1	T_2
R(X)	
	W(X)
	R(Y)
W(Y)	

→ schedule not allowed by basic 2PL.

▲ Every schedule which is allowed under basic 2PL is conflict serializable also.

Question asked – Tell whether the given schedule is allowed under 2PL or not.

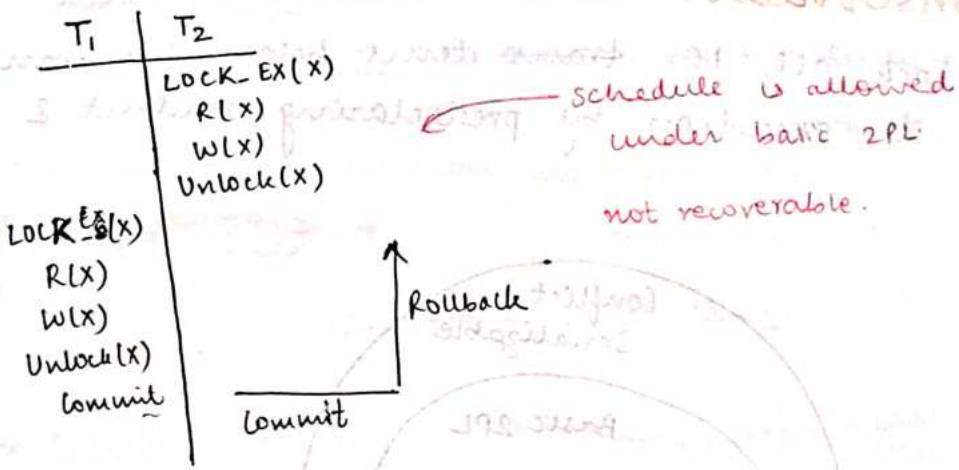
T_1	T_2	T_3	T_4
W(X)			
	W(Y)		
		W(Z)	
			R(X)
			R(Y)
			R(Z)

★ 2PL suffers from starvation.

★ 2PL suffers from deadlock

T_1	T_2
Lock_S(X)	
R(X)	
	LOCK_S(Y)
	R(Y)
LOCK_EX(Y)	
W(Y)	
	LOCK_EX(X)
	W(X)

★ 2PL can allow non recoverable schedules



Strict Schedule – Every write operation should be ended with commit before another transaction performs read/write on the same database item.

Strict 2PL Protocol

Exclusive locks should be released only after commit.

★ Strict 2PL allows only strict schedules.

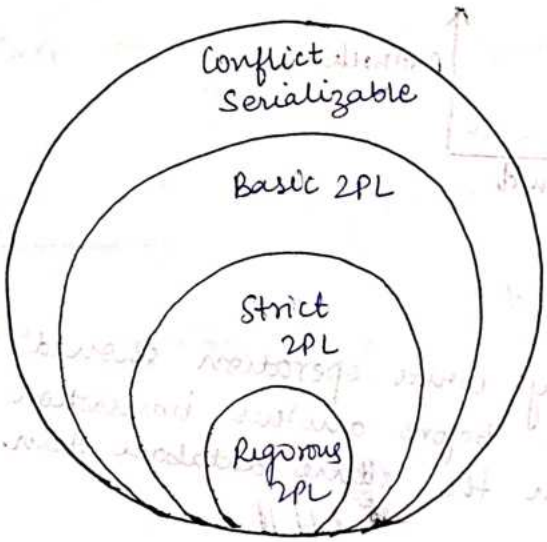
Rigorous 2PL Protocol

Every lock should be released after commit.

- ⊙ Strict & Rigorous 2PL do not suffer from dirty read problem.
- ⊙ Basic 2PL may allow non recoverable schedule.
- ⊙ Strict & Rigorous 2PL have only recoverable schedules (cascadeless)

Conservative 2PL Protocol

Lock all the ~~trans~~ items before the transaction begins its execution by predeclaring read-set & write-set



Timestamp Based Protocols

Timestamp:- unique identifier created by DBMS to identify the relative starting time of a transaction

$TS(T_1) < TS(T_2) \rightarrow T_1$ starts before T_2

T_1 is older than T_2

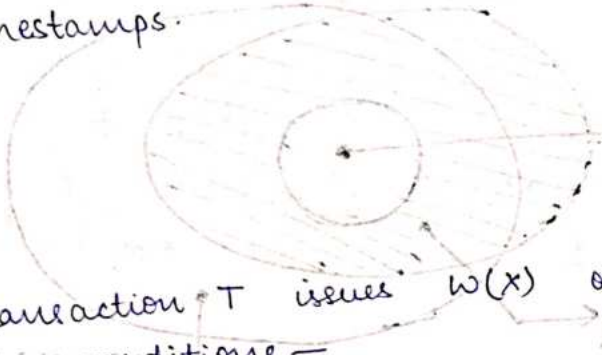
Read Timestamp(A) → Youngest transaction who reads A

Write Timestamp(A) → Youngest transaction who writes A

T_1	T_2	T_3	
R(A)			→ R_TS(A) = 1
R(A)			→ R_TS(A) = 1
		R(A)	→ R_TS(A) = 3
R(A)			→ R_TS(A) = 3 (because T_3 is youngest)

*youngest ≠ latest

- Basic Timestamp algorithm is used to control concurrency by allowing transactions to run in their relative ordering of ~~timest~~ timestamps.



When a transaction T issues W(X) operation, check the following conditions —

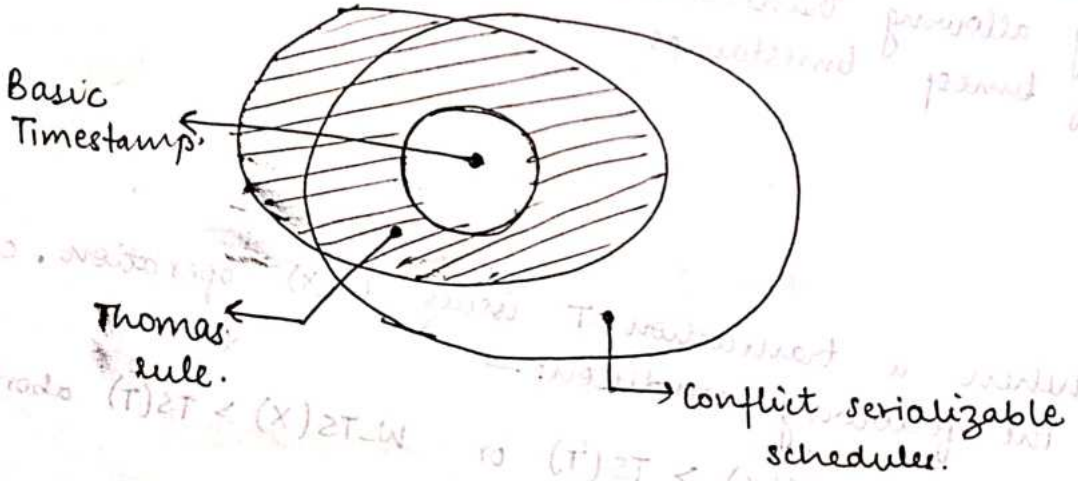
- If R_TS(X) > TS(T) or W_TS(X) > TS(T) abort the transaction and rollback T.
- else, execute W(X) and update W_TS(X) to TS(T).

When transaction T issues R(X) operation, check the following conditions -

- If W_TS(X) > TS(T) then abort and reject the operation.
- else execute R(X) and set R_TS(X) to max(TS(T), R_TS(X))

Thomas Write Rule

- Read is same as basic timestamp rules.
- W(A) in transaction T:
  - → If R_TS(A) > TS(T), abort T, rollback
  - → else if W_TS(A) > TS(T), skip the write operat
  - → else perform W(A) and update W_TS(A)= TS(T)



- Some schedules which follow Thomas write rule are not conflict serializable

Deadlock Prevention

~~1. Wait - Die~~

Let T_i = older transaction T_j = ~~new~~ younger transaction

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

1. Wait Die –

→ If T_i tries to acquire lock currently held by T_j, then, T_i waits till the lock is available.

→ If T_j tries to acquire lock currently held by T_i, then, T_j is aborted and restarted with same time stamp.

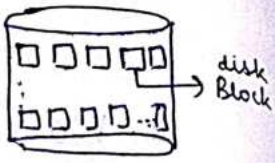
2. Wait Wound –

→ If T_i tries to acquire lock currently held by T_j, then, T_j is pre-empted by aborting it.

→ If T_j tries to acquire lock currently held by T_i, then, T_j waits till the lock is available.

Disk Block -

- ⊙ logical unit of storage created by file system.
- ⊙ A combination of 1-2 sectors.
- ⊙ size = 512B to 2KB



Record storage -

Suppose, 1 disk block = 64 bytes = 2^6 bytes

Student

Name	ROLLNO.	....

→ 1 record = 4 bytes

∴ No. of records in 1 disk block = $\frac{2^6}{2^2} = 2^4 = 16$

If the table contains 512 records,

no. of disk blocks needed = $\frac{512}{16}$ = 32 blocks.

Fixed v/s Variable Length Records -

Student table

Roll NO.	Name	Photo (max. 2MB)	Sign (max. 200kB)

max. size of 1 record ≈ 2.2 MB

If a student uploads pic of size 1 MB, sign 200 kB,

size of record = 1.2 MB.

In GATE syllabus →

In fixed length record, size of record is fixed = 2.2MB (extra space is wasted)

In variable length record, size of record is not fixed (space ~~saved~~ utilized)

Spanned Mapping –

record of a file is stored inside the block even if it can only be stored partially, hence, record is spanned over multiple blocks.

Block size = 32B
Record size = 5B
6 records in block 1
2B of 7th record in b1
3B of 7th record in b2

Adv :- no wastage of memory (internal fragmentation)

Disadv :- memory access time is larger.

Unspanned Mapping

a record of file is stored inside a block only if it can be completely stored inside it.

Block size = 32B
Record size = 5B
6 records in B1
7th records in B2
2B in B1 wasted

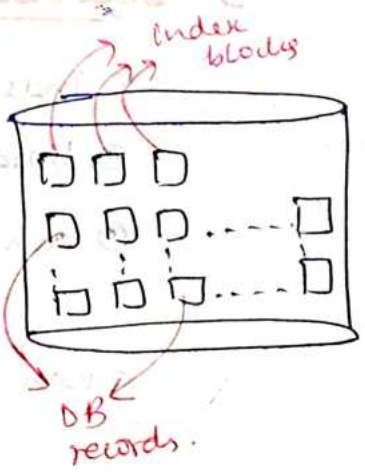
Adv :- access time of records is less

Disadv :- wastage of memory (internal fragmentation)
<space wasted per block>

Indexing

index file is created to store information of which records are present in which disk block.

Index file/table is also stored in disk blocks.



↦ Dense index:
Index record is for each database record.

↦ Sparse Index:
Index record is for a few records only.

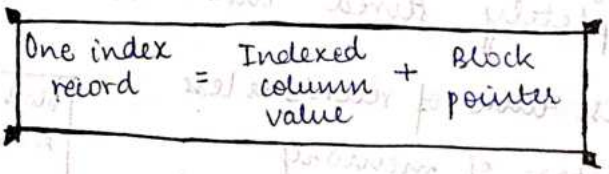
Indexing Techniques

- Primary indexing
- clustering indexing
- Secondary key indexing
- Secondary non key indexing

1. Primary indexing

- Indexing is done on ~~key~~ primary key or super key.
- Data must be ordered on index.
- Its always sparse index.

जिस order में index है, उसी order में data भी होना चाहिए



2. Clustering indexing

- Indexing is done on non key field. (duplicate values may be present)
- Data must be ordered on index field

→ Indexing is done for unique value of non-key

- sparse indexing

3. Secondary key Indexing

- Indexing done on primary key or any super key
- Data must not be ordered on index field
- can be dense or sparse

< non key > → similar to key indexing except that indexing is ~~done~~ done on non key field.

B Tree

- ⊙ Tree based indexing technique.
- ⊙ Dynamic indexing technique.
- ⊙ Self balancing ~~tree~~ search tree

→ grows horizontally as well as vertically.

Order of B tree - max children of a node of B Tree (order is denoted by 'p')

An order p B-Tree —

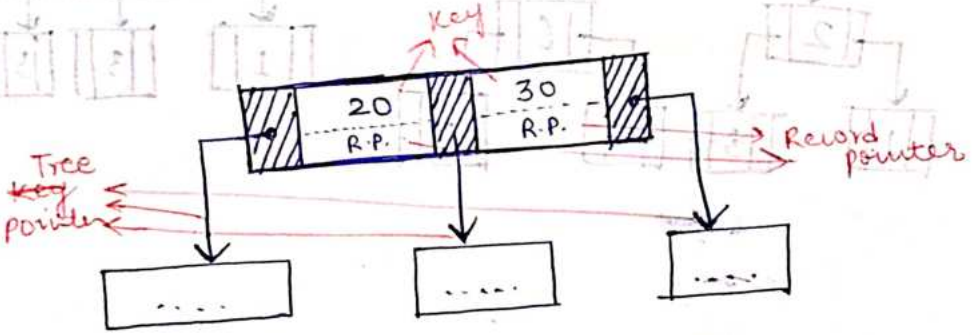
- ↦ every node has max p children (tree pointers)
- ↦ every node other than root should have atleast $\lceil \frac{p}{2} - 1 \rceil$ nodes.
- ↦ every node can have atmost (p-1) keys and p tree pointers
- ↦ Root can have minimum 1 ~~node~~ key
- ↦ all leaves are at same level.

B Tree node structure

Key - index pointer

Record pointer :- pointer to the disc block where ~~rec~~ key index is stored.

Tree pointer :- children.



Example –

Order 3 B Tree

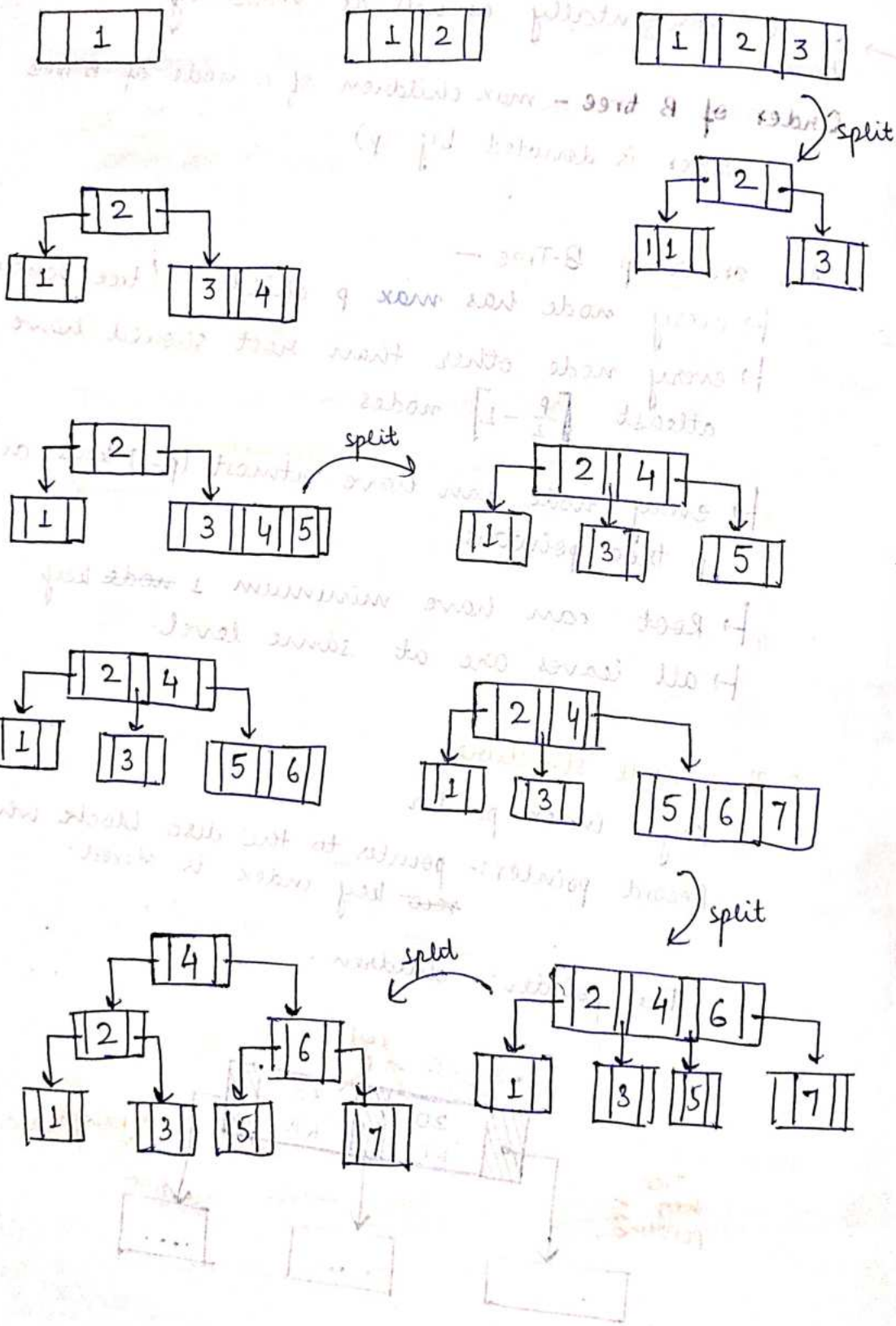
Max keys = 2

Max links (children) = 3

Min keys (except root) = $\lceil \frac{3}{2} \rceil - 1 = 1$

Min ~~Max~~ keys (except root) = $\lceil \frac{3}{2} \rceil = 2$

Insert keys 1, 2, 3, 4, 5, 6, 7



⊙ Each node of a B Tree is stored in a disk block.

If p is order of tree,

max No. of keys = p-1

max. No. of record pointers = p-1

max. No. of tree pointers = p.

$$\therefore (p-1)(\text{key size}) + (p-1)(\text{record pointer size}) + p(\text{tree pointer size}) \leq \text{Block size}$$

formula used to find max. value of 'p' if block size is given.

or a min block size if p is given.

Imp.-

p order B tree

Total nodes = n

H_{min} = min height of BTree = $\lceil \log_p (n+1) - 1 \rceil$

H_{max} = max height of BTree = $\left\lfloor \log_{\lceil \frac{p}{2} \rceil} \frac{n+1}{2} \right\rfloor$

B-Tree deletion in leaf node -

→ After deletion, if no violation of ~~node~~ minimum keys then no changes in tree.

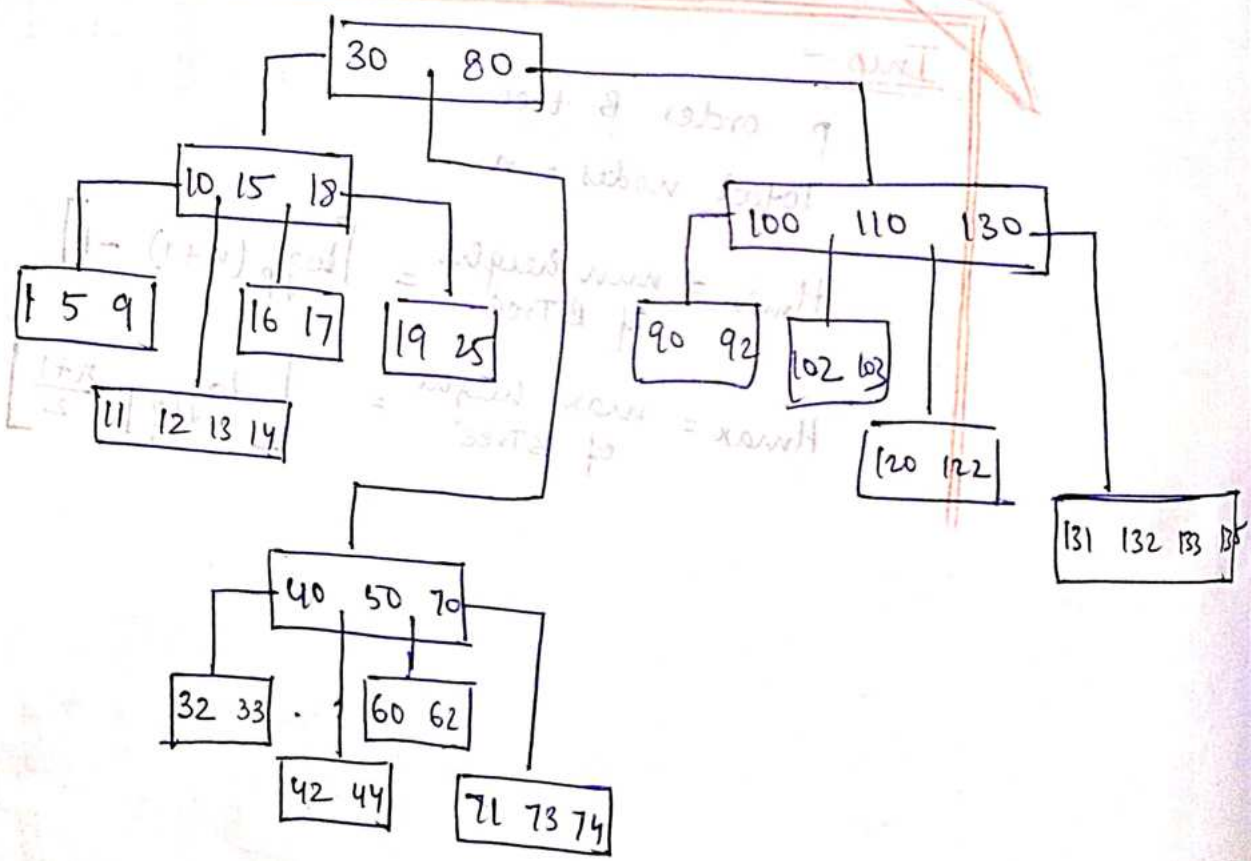
→ If violation of min keys, then, borrow key from sibling (rotation through parent)

→ If borrow through parent can't be possible then, merge the node with sibling & pull down the ~~act~~ anchor ~~node~~ key from parent.

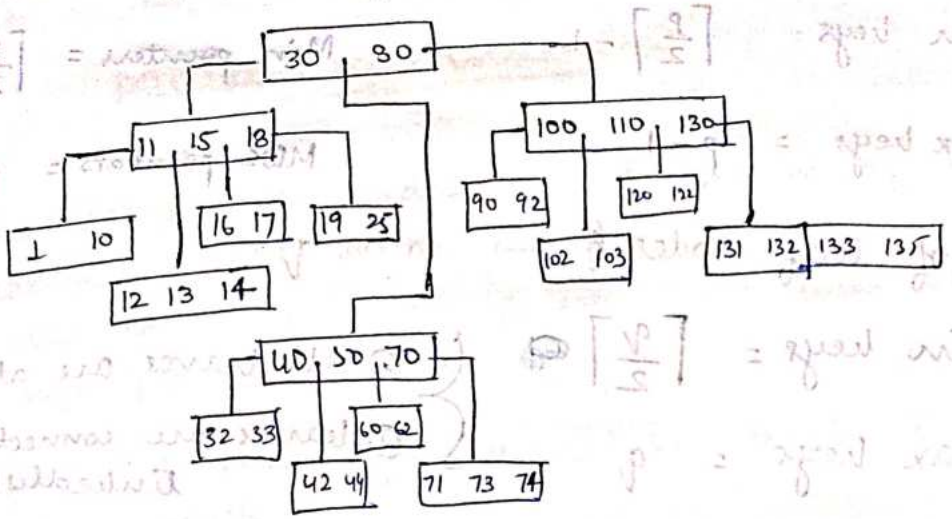
• B tree of order 5

max keys = 4

min keys = $\lceil \frac{5}{2} \rceil - 1 = 2$



Delete 5 → no change
Delete 9 → borrow from sibling



B tree deletion from internal node —

→ Replace the deleted value with inorder successor or inorder predecessor.

→ Now follow rules of deletion of key from leaf node.

B^+ Tree

B^+ Tree
- Internal nodes
  - Keys
  - Tree pointers
- Leaf nodes
  - Keys
  - Record pointers
  - Linked list pointer

} order of leaf node and internal node is different

① Leaf nodes are connected to each other ~~like~~ as linked list

② Range queries work faster in B^+ Trees.

Order of internal nodes (except root) → order p.

Min keys = $\lceil \frac{p}{2} \rceil - 1$ ... Min pointers = $\lceil \frac{p}{2} \rceil$

Max keys = $p - 1$ Max pointers = p

Order of leaf nodes → order q

Min keys = $\lceil \frac{q}{2} \rceil$

Max keys = q

① all leaves are at same level
② leaves are connected using linkedlist.

★ Splitting at internal node is same as that of B Tree
Splitting at leaf node — middle key is duplicated & is part of right subtree.

① One node of B+ tree must be present in one disk block.

Order of B+ tree = p.
Key = 8 bytes
Block pointer = 32 bytes
Record pointer = 48 bytes
Block size = 2048 bytes

For external node,
max (q-1) keys
(q-1) record pointers

$\therefore (q-1)(8) + (q-1)(48) \leq 2048$

$\Rightarrow (q-1)(56) \leq 2048$

$\Rightarrow q - 1 \leq \frac{2048}{56}$

$\Rightarrow q - 1 \leq 36.57 \Rightarrow q \leq 37.57$

$\therefore q_{max} = 37$ (max)

For internal node,
max (p-1) keys
p block pointers

$\therefore (p-1)8 + p(32) \leq 2048$

$\Rightarrow 8p - 8 + 32p \leq 2048$

$\Rightarrow 40p \leq 2048 + 8 \leq 2056$

$\Rightarrow p \leq \frac{2046}{40}$ 51.125

$\therefore p_{max} = 51$ (max)

max order of internal nodes B+ tree = 51

max order of leaf node of B+ tree = 37.

~~Multipl~~

Multi-level indexing

→ In secondary key indexing, if the no. of records in the database table is huge, then, the index file size will also be very huge. & searching will need to access multiple disk blocks.

↳ entire o index file cannot be stored in a single disk block.

→ ∴ Do indexing on the index file.

< Primary indexing >

No. of blocks accesses needed to search a record is minimized.