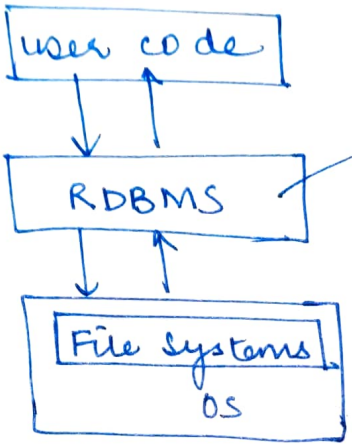
Database Management Systems

- Software for providing a quick way to access and modify data

RDBMS

↳ Data is stored with the help of relations & tables.



Software available that provide servers to read and write data.

→ Oracle, MySql, PostgreSQL.

Schema of table → basically the header of each column.
→ logical definition of a table
→ defines name of table, name & type of each column
(Blueprint)

Student
- st id
- name
- address
- class.

RDM **RDBMS** provides SQL (structured query language
↓
fourth gen. programming language.
↓
allows user code to access the data in convinient way.

Relational Database Management system
↓
also have administrative panel to define user role → codes to handle concurrency

RDBMS vs File system

- possibility of Redundancy.
- Concurrency issue
- Data inconsistent
- security issues

→ Focus only on business logic.

Disadvantages of RDBMS

- Has to be in proper structural form | Data has to follow the schema.
- Scalability issues

→ can be handled by NoSql

- MongoDB
- DynamoDB
- cassandra.

Entity Relationship Model

→ Design DB before implementation

ER model

- Entity set
- Relationship set
- Attributes

Attributes → columns of each row
name, id, address etc.

Relationship between entities → teaches

main items in DB → a particular row in table → entity set ||| table

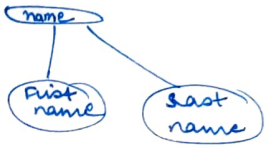
How to Represent these

ES → ▭ or ▭ (double) → weak ES.

RS → ◇ or ◇ (double) → weak

Attributes ⬭

- composite
  - multiple features
  - Eg:- name → First name, last name



Address → ~~school name~~ house no. street no. district state.

- Multiple values
  - → Multivalued

Eg:- Phone number ⬭

- Derived attribute
  - can be derived with the help of existing attributes

Eg:- age with the help of ~~dat~~ DOB

- Key attribute
  - that define a particular entity set

⬭

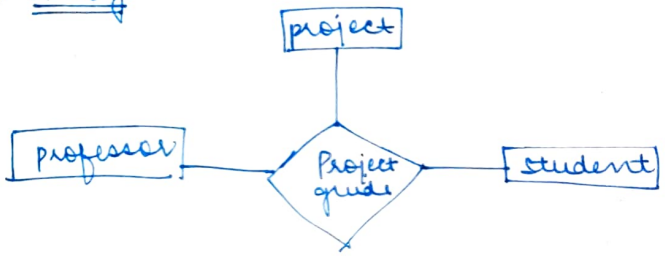
Relationships

Degree of relationship

1) unary → ✗ entity set related to itself
   - student is friend of student
   - employee is managed by employee
   - person is married to person.

2) Binary →
   - → student attends course
   - → supplier supplies items
   - → prof. teaches subject

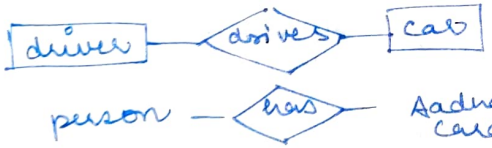
3) n-ary



Cardinality

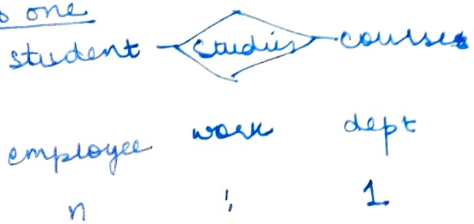
↳ how many entities of 1 side can participate in a relationship

- one to one

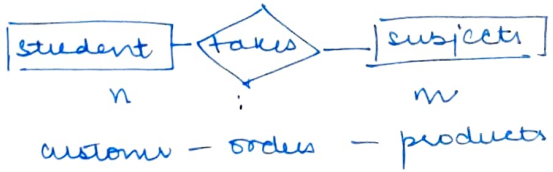


- one to many

- many to one



many to many



customer — orders — products

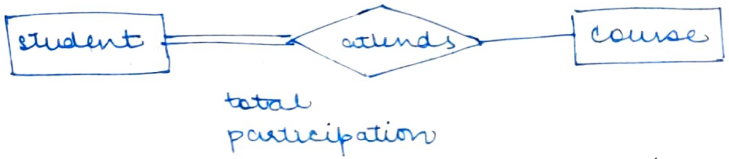
ER diagram

participation

entity set → set of entities

participation of entities

total → every entity of one side participate in the relationship



weak entity set

→ do not have their own primary key.

it always has total participation

Key → attribute or set of attributes that define a an entity uniquely.

candidate key
↳ minimum set of attributes of identify uniquely.

primary key is one of candidate key.

super key ≥ candidate key

Candidate keys → Primary keys + Alternate keys

Ex:- Student

Primary — Roll no
Aadhar card } — candidate keys
Pan card

Alternate keys

→ candidate key has to be minimal.

Super key → any combination of keys so that a row can be uniquely identified

Armstrong axioms

- Reflexing (A derives A)
- & Transitivity (A → C, C → D ⇒ A → D).
- Augmentation (X → Y, XZ → YZ)

1) $R_1(A, B, C, D)$

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

candidate key → A derives everything

2) $R(A, B, C, D)$

$AB \to CD$

→ candidate key — AB

3) $R_3(A, B, C, D)$

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

→ CK → B

Foreign key

→ Data has to be divided into different tables to reduce redundancy.

Order table — Foreign key (customer id) — customer table

Referential Integrity

→ Foreign key must not point to NULL set.
→ caused by removing data from 1 table

Normalization

- To reduce data redundancy (having same data at multiple data)
- ~~DB~~ data Integrity
  - No erroneous data

Objectives of good database design

- No updation, deletion, insertion anomaly.
- Easy Extendible
- good performance for all query sets.
- more informative

(a) Update anomaly

- some database is update while other didn't because of server crash / other failure leading to inconsistent data.

(b) ~~I~~ Insertion anomaly

no able to insert data because of some required fields → can be resolved by splitting tables.

(c.) Deletion anomaly → other data is also deleted when the row del. which needs to be prevented

Functional dependency

$A \rightarrow B$ B is functionally dependent on A

use → if we lookup A, we can definitely reach B.

FD
- Trivial
  - $AB \rightarrow A$
  - $ABC \rightarrow A$
- Non-trivial } — gives some information
  - $A \rightarrow B$
  - $BC \rightarrow D$

First Normal Form

every attribute contains only single values.

cust ID	Name	Phone no
101	ABC	999, 923
102	XYZ	456, 178, 984 ✗

} many value

create different columns for mobile no.

cust ID	name	Ph-1	Ph2	Ph3
101	ABC	999	923	—
102	567	456	178	984
m	m	236	-	-

↓ lot of null values

creating different entity for same customer

custID	Name	Ph
101	ABC	999
101	ABC	923
101	ABC	956
102		

or can create a separate table

cust ID | name → cust ID | ph.no.

Second normal form

non-prime attribute should depend on partial candidate key.

user_id(A)	course_id(B)	course_fee(C)
1	CS101	5000
2	CS102	2000
1	CS102	2000
3	CS101	5000
4	CS102	2000
2	CS103	7000.

course_id → course_ID → (Non-prime attr).

Candidate key → (user_id, course_id)

↓ part of candidate key.

$AB \rightarrow C$

$B \rightarrow C$

- $A \rightarrow B$
- $B \rightarrow C$

Third normal form

non-prime → non-prime (Not allowed)

$R(A, B, C, D)$

$A \rightarrow (B, C, D)$

$C \rightarrow D$ ← violates.

BCNF

Prime / non prime → prime attribute ✗

Indexing in Database

→ to reduce lookup time

Clustered index

data is stored in increasing order on the basis of primary key.

- Extra cost in deleting & insertion

Prime index → primary key is used as index.

Non clustered indexing

secondary indexing

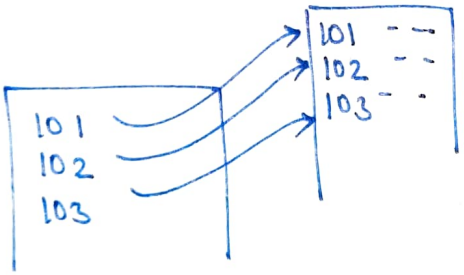
data is not ordered on the basis of key in HDD

---

clustered Index

search key	Pointer

→ stores reference to disk block containing given key



Ordered Index file
Hashing file org^n

Dense

every key has a entry in index table

sparse

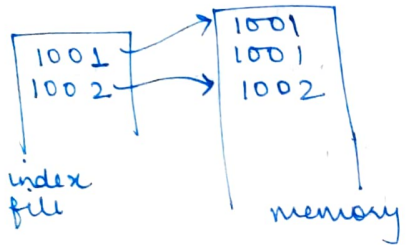
↓

some keys are skip

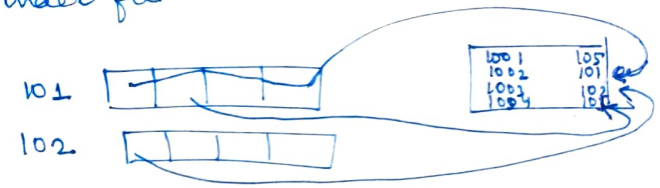
101
101
121
→ search akey 106
↓
go to 101 and move linearly

Dense → Every entry has to be present exactly once, not every row

order-ID	order date	cost	cust-id
101	-	-	1001
102	-	-	1002
103	-	-	1001



Non-clustered index.



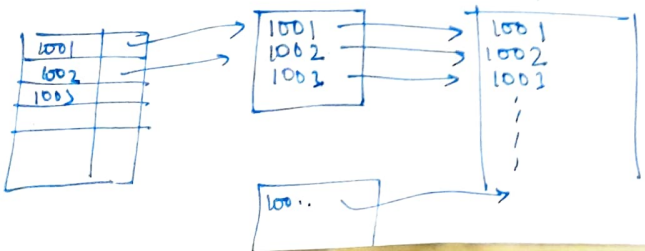
All the attributes are stored in increasing order in index file ~~bbb~~

Sparse is not possible.

there are many fields ~~on the basis of~~ corresponds to single index

Multilevel indexing

→ we have big index file, then we create one more index file for the index file



we use B trees for ensuring scalability.

↳ Balanced Binary search trees which grow & shrink accordingly

B and B+ trees

→ n-ary search trees

→ If branching factor is b, then every node has $\lfloor b/2 \rfloor$ to b children.

→ In B+ tree leaves contain all the data present in internal nodes so that sequential access to data is possible.

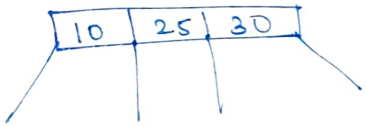
↳ height never goes beyond $\log n$ if there are n cells

b = 4

min = 2 children

max. = 4 "

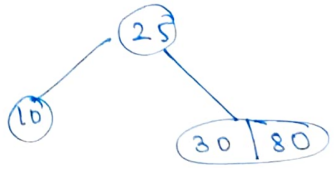
Key → children



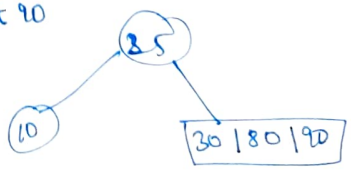
if we want to insert 80 → not possible as it will have 5 children

B tree has data in internal nodes also

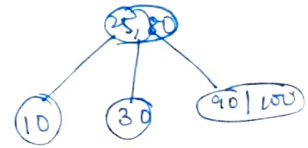
B+ trees has data on leaves only



insert 90

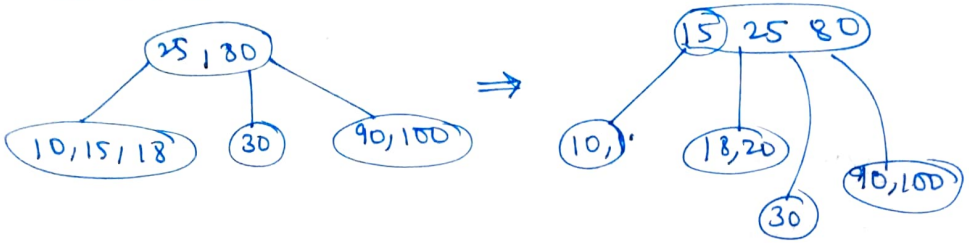


insert 100

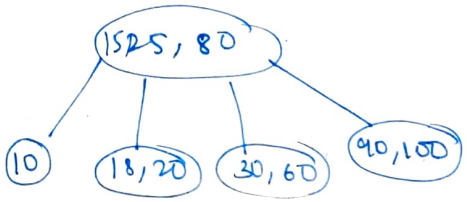


split by centre

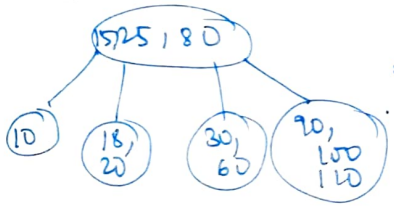
insert 15, 18



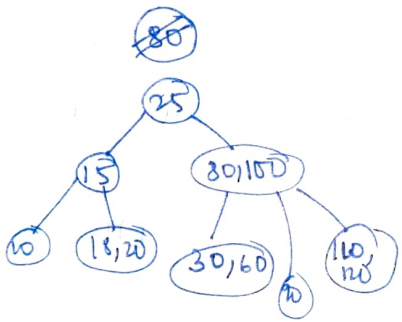
insert 60



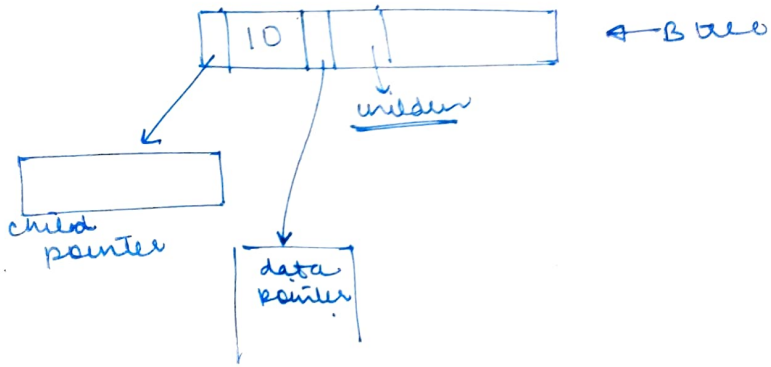
insert 110



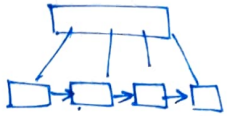
insert 120



in deletion → shrink



B+ trees → only leaf nodes contain data pointer and leaf nodes are connected



~~index~~ internal nodes has pointer to index blocks, leaf node has pointer to data blocks.

→ every node has only one pointer & sequential access.

Transaction & concurrency control

↓

set of instructions that has to be executed together

Atomicity → either whole transaction is completed or every instruction executed must be reverted

Consistency → Database should be in consistent after a transaction happened.

Isolation → no transaction must not be interfered by other transaction

Durability → changes should be permanent.

Conflicts & scalability.

ii) Serializability

You have an interleaved schedule now check if it is view equivalent of T_1T_2 or T_2T_1

Suppose if we have 6 transactions then we have to check if our trans. is view equivalent to any of 6! permutations

Serializability

↳ check if transactions schedules is serializable. A serializable schedule is one that always leaves the DB in consistent state.

It is required only when interleaving is done.

We need to check these 3 rules for every item

Initial Read
Update Read
Final write

~~tets~~ Lets do T_1T_2

data item X
→ initial read is done by T_1

data item Y
~~tt~~ initial read is done by T_2 ✗
↓
because in T_1T_2 Y must be read by T_1 first
X.

check for T_2T_1

data item X
↳ T_2 is not reading X and prev.

data item Y
↳ T_2 reads first.

T_1	T_2
Read (X)	
X = X − 10	
Write (X)	
	Read (Y)
	Y = Y − 10
	Write (Y)
Read (Y)	
Y = Y + 10	
Write (Y)	
	Read (Z)
	Z = Z + 10
	Write (Z)

② Update Read.

sequence of read write should be same in both schedules.

③ Final Write → final write reads on an item should be same.

Conflict serializable

conflict operations

- Two operations from two different transaction on same data with one of them being "Write".

check if given schedule is conflict equivalent to any of the schedules T_1T_2 or T_2T_1

T_1	T_2
Read (A)	
Write (A)	
	Read (A)
	Write (A)
Read (B)	
Write (B)	
	Read (B)
	Write (B)

we can swap ↕ operations and get a serial schedule
↳ Hence conflict serializable.

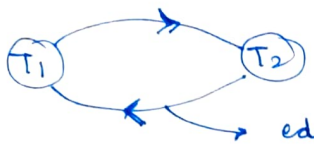
T_1	T_2
Read (A)	
	Read (A)
	Write (A)
Write (A)	
Read (B)	
Write (B)	
	Read (B)
	Write (B)

we can't swap these operations as these are conflict Reading & writing A

T_1	T_2
Read (A)	
Write (A)	
Read (B)	
Write (B)	
	Read (A)
	Write (A)
	Read (B)
	Write (B)

Simple graph based method (Precedance graph)

no of transaction → no of nodes



cycles shows that not conflict serializable

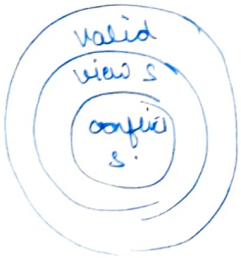
edge when there is a read followed by write

T_1	T_2
Read (X)	
	Read (Y)
	Write (X)
Write (Y)	

T_1	T_2
Read (A)	
Write (A)	
	Read (B)
	Write (B)
Read (B)	
Write (B)	
	Read (A)
	Write (A)

not conflict serializable

conflict serializability is more strict



Recoverable, Cascadeless and Strict

problem → suppose a transaction is rolled back in middle, the changes done by it is it's execution is used by another transaction that has committed already.

T_1	T_2
Read (A)	
Write (A)	
	Read A)
	Write (A)
rolled back ↑ Read(B)	↑ comimi
failed	

T_1 is rolled back and but T_2 has already committed which leads to inconsistent DB

↓

Irrecoverable

Rules to ensure recoverable schedule

* one problem is dirty read → reading the value changed by the transaction that has not committed yet.

If there is a dirty read then the transaction doing dirty read must commit after the transaction from which it's reading.

Cascadeless

↳ one rollback is causing other rolls back

to ensure cascadelessness

- we make some rules
  - dirty ready will wait until the transaction has committed already.

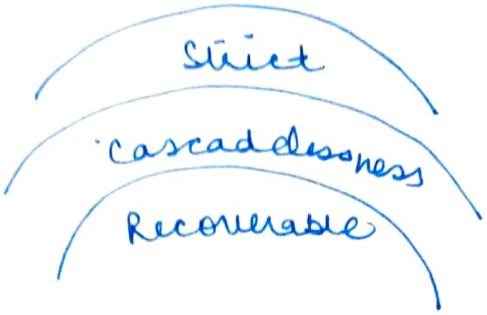
Strict schedule

→ BI handles blind writes

→ Write without Read

↓

even write operations should wait for commit



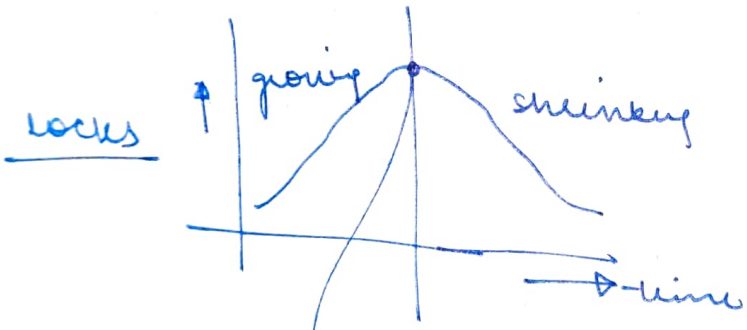
Two phase locking protocols

two types of locks → shared
→ Exclusive

two phases → growing
→ shrinking

growing → all locks are acquired here only

shrinking → all locks are released here only.



lock point → where all locks are acquired and no more locks are left.

	S	E
S	✓	X
E	X	X

→ deadlock
→ cascade