Rela^{n}al Databases

1) lotta dBs out there

2) 2 major categories depending on struc imposed on these dBs

(1) Rela^{n}al a.k.a SQL dB (eg:- POSTGRESQL)

(i) A type of dB that imposes on the data stored in it, a tabular like structure (data stored as table)

(ii) Rows (record) → instances of the entities that the resp. tables represent

Columns (attribute) → Attributes of the entities

eg:- table → payments record for a shop

row → each customer payment details such as date, amount, transaction medium

colum → the attr.

Cust. name	Date	Amt	Transac^{n} medium
A	17/9	10	Credit Card
B	15/9	45	Cash
C	14/9	3	Netbanking
D	13/9	87	Cash

(iii) Schema → Define the shape of the table i.e. it will basically specify what attr (col) each entity (row) will have

eg:- schema for a book of library

Each row → book name

For each row, cols → borrowed on, returned on, details of payment, condition of book

Date:
M T W T F S S

② Non rela'al dB a.k.a noSQL (eg:- mongo DB, Google Cloud Datastore)

(i) dBs don't impose tabular struc. on data stored in them. Sometimes they may impose some kinda other struc. but def'ly not table

(ii) Flexible, less rigorous

SQL → Structured Query Lang.

i) Most rela'al dB (rdB) support SQL

ii) SQL is used to perform powerful complex queries in rdBs

iii) Because rdBs support SQL, they come with powerful querying capabilities and this is also the reason why most people choose rDBs over non-rDBs for part of their sys.

iv) Why SQL > Python, JS

Cuz for Python, JS you gotta load data in mem. to perform these kinda queries

So when we talk abt large scale distri. sys, we got Tbs of data and we can't do this trivially

SQL dBs aka RDBs

1] Must use ACID transac"s → transac" supportin ACID props.

Atomicity Consistency Isola" Durability

i) Atomicity - If a transac" consists of multiple sub-opn then these sub-opn are gonna be considered as 1 unit so they'll either all succeed or all fail

eg:- ₹ funds transfer in banks

→ sub-opn 1 → Reduce money in person 1's acc

→ sub-opn 2 → Increase money in person 2's acc

ii) Consistency (aka Strong Consistency) → dB is never invalid and never stale

- (i) Any transac" in dB is gonna abide by all the rules in the dB → dB is never invalid
- (ii) Any future transac" in dB, is gonna consider any past transac"s in dB i.e. no 'stale' state' in dB where 1 transac" has exec.ed but another transac" doesn't know it has exec.ed → dB is never stale

iii) Isola" - Multip. transac"s can occur at the same time but under the hood, they will actually hv been exec.ed as if they had been done sequentially (i.e. put in a queue) one-by-one

iv) Durability - When you make a transac" in a dB, the effects of that transac" in the dB are permanent

i.e. data stored in the dB is effectively stored in disc (not mem.)

Database Index (dBI)

A dBI is a data struc that improves the speed of data retrieval ops on a dB at the cost of addi"al writes and storage space to maintain the index data struc

Consider this eg for bank passbook

Month	Date	Amount credited	Amt debited
January	28th	10000	—
Feb	7th	5000	—
March	15th	700	—
April	27th	2000	—

Say we wanna figure out on which day of the yr was the highest amt. credited. (Ans:- Jan 28th)

(i) Conventional Meth :- Traverse thru entire look thru entire dB to find that amt

O(N)T

(ii) Using dBI :- Auxiliary data struc created i.e. optimized for fast searching on a specific attr. (col.) in the table.

Say our dBI = table that has all credited amts stored in sorted order (O(NlogN)) and

each of these amts pts. to the relevant record (row) in the main dB table
Thus, we get our largest amt by simply goin to the end of the dBI table

simple way of lookin at dBI

- auxilliary data struc of main dB that speeds up gettin data frm main dB (read opns. faster)
- auxilliary data struc so extra space reqd. for storage
- whenev. you write data to main dB, you also gotta write to dBI (write opns. slower)
- In practice, we create an idx on 1 or multip cols. in our main dB

NOTE : Eventual Consistency

- A consistency model which is unlike Strong Consistency
- In this model, reads might return a stale view of the sys.
- An eventually consistent datastore will give guarantees that the state of the data will eventually reflect writes within a time period (eg:- 10 mins, 30 days)
- eg:- Google Cloud Datastore