Data → raw facts

Information → processed data with meaningful interpretation

Database : structured collection of interrelated data in form of tables ex: Banking, Management Systems, Airline Reservation

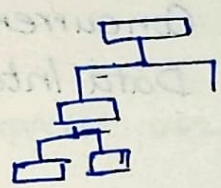
File System : structured form used for management of data used by an OS

CONS

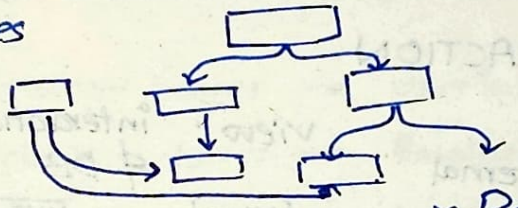
Data Redundancy
Poor Memory Utilization
Inconsistency
Not secure

Types of Databases :

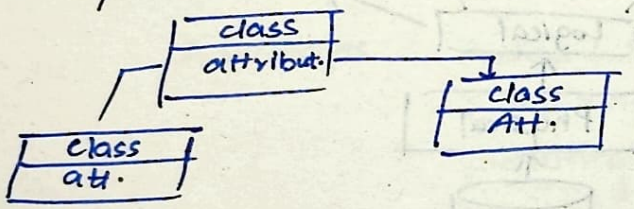
* Hierarchial



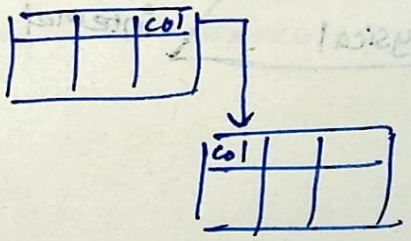
* Network Databases



* Object Oriented



* Relational



* Cloud Databases
  - BigQuery
  - Azure SQL

* Centralized
* Personal
* NoSQL
  - unstructured data
  - Ex: MongoDB

WHY?

Efficient Management
No Redundant data
Consistency
Data Security
Data Independence
Scalable
Abstraction

WHY NOT?

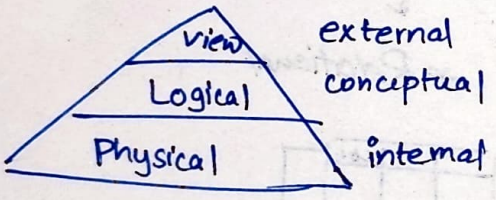
Cost
Not suitable for smaller projects
Vendor Lockin

APPLICATIONS

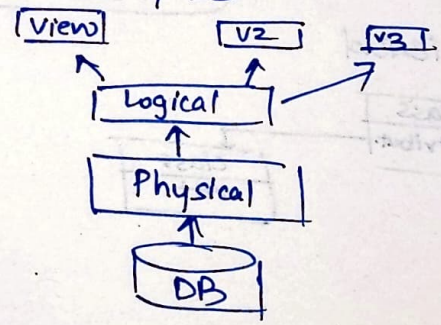
Organisation
Search and Retrieval
Access Control
Concurrency control
Data Integrity

Management systems
E-comm.

LEVELS OF ABSTRACTION



View : interaction layer btw users & DB



Physical

upto the DBA
deals with actual storage org, allocation

Logical

simpler commands are used to perform ops
usually used by programmers

External → user view

ex: customers

Database Architecture

Schema : logical container
defines structure

Types :

Physical Schema
Data storage in underlying hardware
High risk and efforts

Logical Schema
No concern to physical storage

↳ Conceptual schema
Overall view HL structure

↳ External Schema
External user specific view

3 Tier Architecture

1. Presentation Layer → user interface
2. Application Layer → Business Logic
3. Data Layer → Manage storage and processing

Advantages : Reduced Redundancy, Inconsistency
Simple Access
Security
Concurrency control

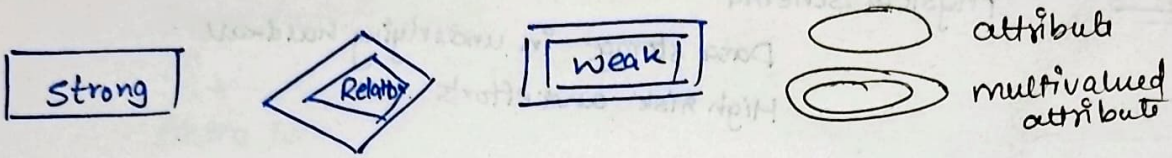
DATA MODELS : Abstract way to represent the schema

Types :
* Hierarchial
* Network
* Relational
* Entity Relationship
* OO Data Model
* NoSQL
  - document (MongoDB)
  - key value (Redis)
  - Column (Cassandra)
  - Graph (Neo4j)

Entity

Strong: Non dependent unique

Weak: Dependent on some primary



Strong Relationship ⟹ No independent existence of entities

weak → inter-related entities.

* Specialization : Top down
* Generalization : BottomUp
* Aggregation : combination

Relational Data base Management

Structured data in form of rows & columns.

Intension : schema of a database, access, indexes etc. are defined.

Extension : data stored at any given time. (dynamic)
includes insertion, deletion, updation

* Data Definition Language (DDL) : create, alter, drop, remame, truncate
* Data Manipulation Language (DML): Insert, update, delete, select.
* Data control Language (DCL) : Grant, Revoke.
* Transaction Control Language (TCL) : commit, Rollback, Savepoint

DATABASE NORMALIZATION

↳ To improve efficiency
consistency
accuracy

WHY?

Remove Anomalies → Insertion
Improve Efficiency ↳ Updation
Remove Redundant data ↳ Deletion

Normal Forms

1NF ⇒ single valued attribute
i.e one cell one value

WHY? To remove duplicates

2NF ⇒ Each non key attribute is dependent on key

WHY? To ensure consistency

3NF ⇒ All non key attributes are independent of each other

~~Boyo~~ WHY?

Boyce Codd NF ⇒ Be in 3NF
+
X should be a superkey for every functional dependency $X \rightarrow Y$

4NF ⇒ No multivalued dependency

5NF ⇒ No join dependency, No further lossless decomposition

Functional Dependency : one set determines the other

$X \rightarrow Y \iff$ Dependent

Determinant

- Full functional Dependency : Removing any of the the attributes of X the dependency fail.

eg: (IFSC + Acc.No) ⟶ Account details Removing either will result in failure

- Partial Dependency : depends on a part of the composite key.

eg: (Amount + AccNo.) + Name) → Deposit Removing Name bares no issues.

- Trivial functional Dependency : dependent is part of the determinant

{Rollno, Name} ⟶ {Name}

- Transitive Dependency : Indirect dependency.

Withdrawal ⟶ Bank Balance ⟶ Bank

Withdrawal ⟶ Bank

- Multivalued Dependency A ⟶ {B, C}

* DENORMALISATION

combining normalised tables

Reasons :
1. Performance
2. Simplification of queries
3. Reducing Joins
4. Enhancing Retrieval of data

CONS :
- ~~Reduce~~.
- High Maintenance
- Inc Redundancy
- ↑ Inconsistency.

Techniques :
1. Add Redundant data
2. Add precomputed tables.
3. Store derived values.

RELATIONAL DATA MODEL

Structured storage of data in form of tables

Tables = Relations

Relational Schema ⇒ Table structure — Relation, attributes, constraints, keys

Relational Instance ⇒ set of data at a particular time in a relationship

Attribute ⇒ Columns of a table / properties of an relation

Domain of an attribute : Possible set of values for an attribute

Tuple ⇒ Each row for a relation

Cardinality → No. of distinct values in an column

Degree ⇒ No. of attributes also called Arity

Relational Algebra : Procedural language

1. Union : Avoids duplicates takes common values as well as non common
2. Intersection : Takes only common values
3. Difference : (A-B) → values in A that are not in B
4. Cartesian Product (A * B) → m*n relation
5. Selection (σ)

KEYS

ensure organization, accuracy and accessibility

1. **Super Key** : set of one or more attributes that can uniquely identify a tuple.

2. **Candidate key** : set of attributes that can uniquely identify a relation record
   It can be composite or simple

3. **Primary key** : Candidate key choosen as primary

4. **Alternate key** : keys that can be primary but were not choosen

5. **Foreign Key** : An attribute that points to the primary key of another table

6. **Composite Key** : Two or more attributes grouped together

7. **Unique Key** : Contains unique value with one NULL value

Consistent

ACID – A transaction is a single logical unit of work

Transaction manager — **Atomicity** : Entire transaction either takes place or doesn't. its 0 or 1

Application programmer — **Consistency** : Valid states before and after any transaction

Concurrency Controler — **Isolation** : Multiple transactions independent of each other concurrently

Recovery Manager — **Durability** : Persistance of changes even after system failure

High Availability

BASE ⇒ flexibility and fluidity (NOSQL)

Ex: Cassandra, MongoDB

Basically Available : Availability of data over consistency

Soft State : Change in data overtime

Eventually Consistent : will be consistent eventually it takes time to converge.

CRUD → Create Read Update Delete

Truncate ⇒ Removes all data from the table

Delete → Can delete specific rows

Drop ⇒ delete tables & databases.

OPERATORS

1. Arithmatic (+, -, *, /)
2. Comparison (=, <, >, <)
3. Logical (AND, OR, NOT)

AGGREGATE

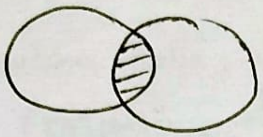
1. count
2. Sum
3. Avg
4. Max
5. Min

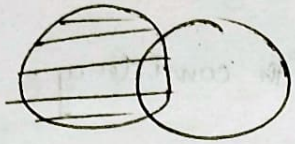
CLAUSES

1. WHERE → used to filter
2. GROUP BY → often used with aggregate functions
3. HAVING → filter the rows created by GroupBy

   GROUPBY → HAVING
4. ORDER BY → sort DESC, AESC
5. LIMIT : Limits the results

JOINS ⋈

1. INNER :  Common value

2. LEFT JOIN:  All records from left table & matched records in from right

3. RIGHT  All records from right & matched records from left

4. ~~CROSS~~ FULL JOIN  All records from both tables.

5. CROSS JOIN : Carterian product of both tables.

6. SELF JOIN: Join a table with itself.

UNIONS combine result of two or more SQL queries. (UNIQUE result) including ALL gets duplicate data too

VIEWS virtual table to display any retrieval.
they can be updated

Inline views : subqueries inside the FROM clause.

Materialized View: saves the view physically

PROS:
* Abstraction
* Data security
* Limiting access

CONS:
performance overhead.

Scheduling : organizing and executing transactions

* Serial Schedules : No interleaving transactions

* Non serial schedules : has concurrency with consistency

a) Serializable schedules :
checks if a database is consistent and the interleaved execution is equivalent to serial execution.

(1) Conflict serializable
if it can be transformed into a serial schedule by swapping non-conflicting operations.

~~An~~ operations are conflicting if
- Belong to diff transactions
- Operate on the same item
- At least one operation is write.

(2) View Serializable : view is equal to a serial schedule (No overlapping transactions)

b) Non serializable

I) Recoverable schedule :
Transactions commit only after all transactions they read commit.

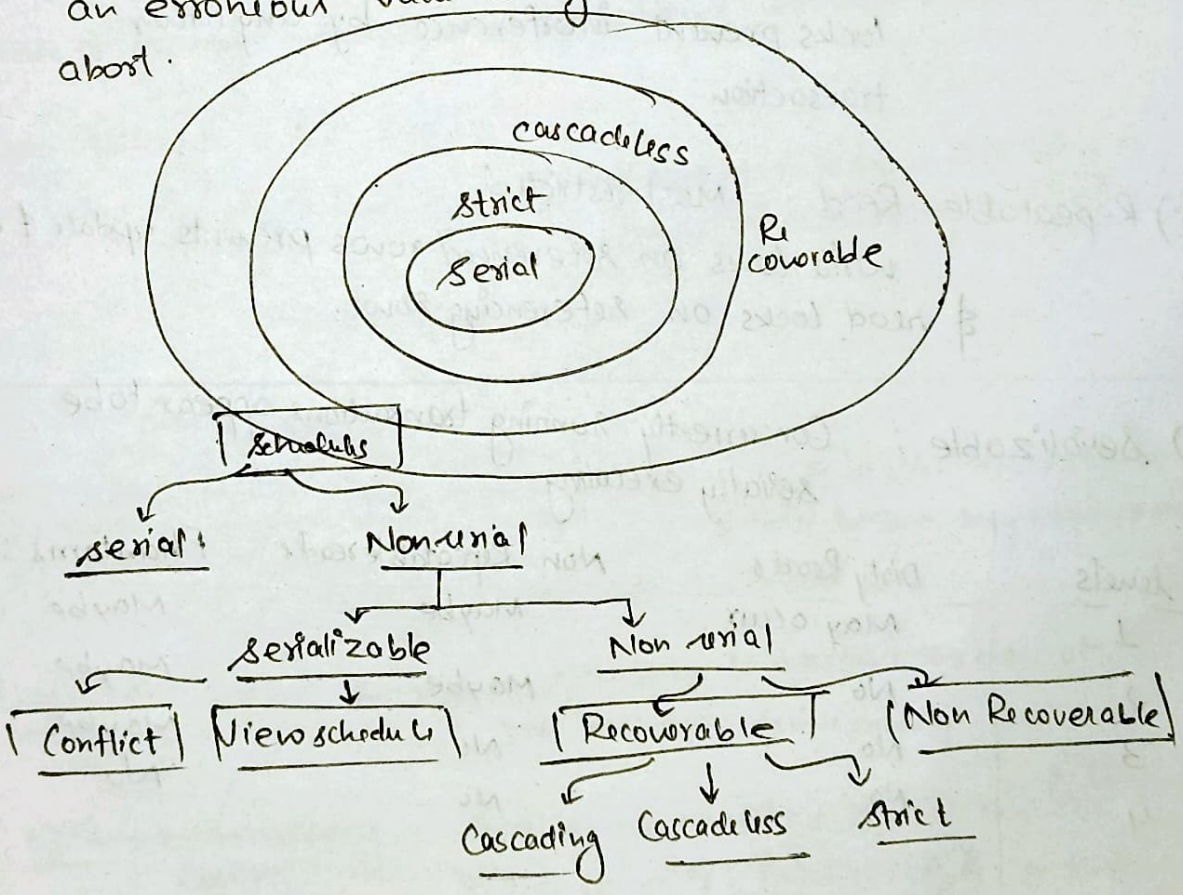
eg: if T_A reads from T_B the only after T_B commits
T_A commits

Recoverable → i) Cascading Schedule
also called Avoids cascading Aborts (ACA)
if one transaction fails all the depending are either rolled back or aborted. (cascading abort/rollback)

ii) Cascadeless Schedule : Transactions only read after all transactions whose changes are to be read are committed.
why? ⇒ Avoids the cascading aborts and rollbacks

iii) Strict Schedule :
A can read or write updated or written value of B only after B commits / aborts.

II) Non Recoverable Schedule
Value is read before commit / abort leading to an erroneous value being read in case of future abort.



TRANSACTION ISOLATION

Dirty Read : A transaction reads data that has not been committed

Non Repeatable read : A reads B twice and gets diff values both times are C has updated B

Phantom Read → two reads of the same query return diff results, by two diff transactions.

4 Levels of Isolation :

i) Read Uncommitted : Reading uncommitted data. i.e a dirty read is allowed.

No isolation at this level.

ii) Read Committed : Any read data is committed. (transactions hold locks)
locks prevent interference by any other transaction.

iii) Repeatable Read : Most restricted
write locks on referenced rows prevents update & delete & read locks on referencing rows.

iv) Serializable : Concurrently running transactions appear to be serially executing.

levels	Dirty Reads	Non Repeatable reads	Phantoms
1	May occur	Maybe	Maybe
2	No	Maybe	Maybe
3	No	No	No
4	No	No	

Additional Levels/Features

Snapshot Isolation : Allows transactions to work with a snapshot of the database at any specific time.

Multi-Version Concurrency Control: Maintains multiple versions of data for better concurrency.

SERIALIZABILITY AND CONCURRENCY CONTROL

↳ ensures the results are same as if the queries were performed serially instead of concurrent execution.

types: i) Conflict : can be converted into serial schedule

ii) View : preserves the same view as serial

CONS: Performance Overhead, Complexity

Methods to manage simultaneous operations without any conflicts

Primary focus: to avoid:

i) Lost update : if two transactions are trying to update simultaneously a single value.

ii) Dirty Read

iii) Non Repeatable Read.

Techniques : i) Lock Based Protocol: A transaction must aquire a lock before any ~~tran~~ operation

Locks can be shared or exclusive.

ii) Timestamp-Based : Assigns timestamp to determine the order of execution. This maintain serializability

iii) Optimistic Concurrency Control : Assumes no conflicts and checks only before commits. Rollsback if conflicts are found.

4. Multiversion Concurrency Control: versions of data are maintained.
(MVCC)

PROS
- Data integrity
- Inc. throughput
- Improved User exp.

APPLICATIONS
- Banking
- Ecom
- Social Media
- Reservations

LOCKING PROTOCOLS

LOCKING PROTOCOLS

To control the access of data to ensure consistency and accuracy.

1. Shared Locks 2. Exclusive lock

Types of locks

Lock is a simple variable associated with the data accessibility at any given time.

SHARED LOCK (S-LOCK): also called the read only lock
Multiple transactions can read at the single instance.
ensures no write operations occur on the given data.

EXCLUSIVE LOCK: (X-LOCK)
only a single transaction can read & write the data
thus single transaction holds the data

Lock Based PROTOCOLS

i) Simplistic Lock Protocol : acquire lock before read/write access
uses single lock type.

PROS: data consistency is maintained through no concurrent access
simple

CONS: Performance bottlenecks due to limited concurrency
No deadlock prevention

ii) Pre-Claiming Lock Protocol : Need to declare all the locks and obtain them before access.

PROS: Avoids cyclic wait

CONS: Reduced concurrency
Inefficient due to unnecessary holding of locks.

iii) Two Phase Locking (2PL)

Transaction → Growing Phase : cannot release locks
→ Shrinking Phase : cannot aquire locks

Guarantees serializable transactions.

PROS: Consistency
Ensure no interference

CONS: Deadlocks
Performance overhead

iv) Strict Two Phase locking (S2PL)

Must hold all exclusive locks untill commit/abort.

Shared locks can be released

PROS: Avoids cascading aborts
simplifies recovery as uncommitted changes are not visible to other processes.

CONS: Deadlock
Reduced concurrency.

Database Recovery Management

Types: Transaction Failure → logical } software
System failure → OS }
Media failure → Physical
Natural Disaster →

Techniques:

i) Backup and Restore: restoring from backups
ii) Log-Based Backup: Redo completed transactions & undo failed transaction

3. Shadow Paging : a shadow copy is maintained of the database page.
In case of updates the other stays unchanged. acting as a point of restoration in case of failure.

4. Checkpointing : snapshots of the database are created at particular intervals.

5. Rollback & Rollforward
↓ undoes incomplete transactions
↳ Applies changes by completed transactions

Best Practices : Regular Backups
Offsite storage : 3:2:1 rule
Test recovery processes.
Automate processes.

QUERY OPTIMISATION

1. Indexing : pointers to speed up retrieval of rows
Have a overhead thus periodic check for unused indexes is essential.

2. Avoiding select *
3. using EXISTS instead of IN or COUNT
↳ searches for first occurence only

4. WHERE instead HAVING
↓ filters first
↳ groups first

5. USE JOINS over Subqueries
6. using LIMIT or TOP
7. Avoid wildcards at beg like patterns.

9. using proper datatypes

10. Avoid functions over indexed columns

~~#~~

INDEXING

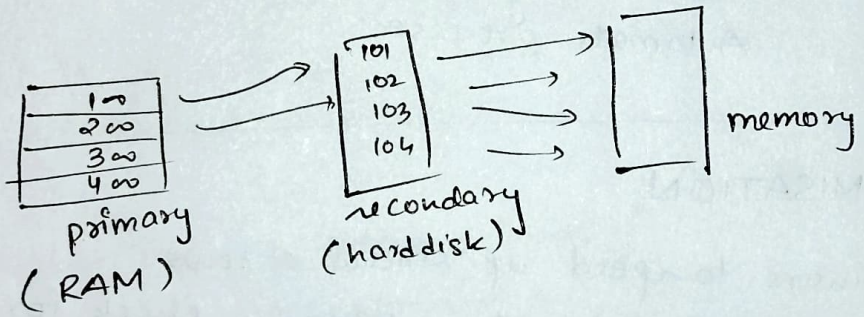
PROS : faster queries

CONS: storage & computation overheads. throughput hit

i) Ordered Indices
incares with sorted index

ii) Clustered Index : grouping

iii) Secondary Index : Additional indexing to reduce size of mapping in sparse indexing



iv) Primary Index : Made on the primary key of the table

Sparse index : records for only some of the values
ex: for 1000 records at every 100

Dense Index : index for every search key value.

B - Trees

All leaf nodes must be at the same level

Balanced data structure

Each node has m children where m is the tree order
(m-1) keys

Non root and Non leaf nodes have atleast ceil(m/2) children

Nodes are only added at leaf nodes.

B+ Trees

Internal nodes act as guides while data is stored at leaf nodes.

Leaf nodes are linked | self balanced

PROS

Fast access and efficient
Support for large data sets
Self balancing
Dynamic size Handling

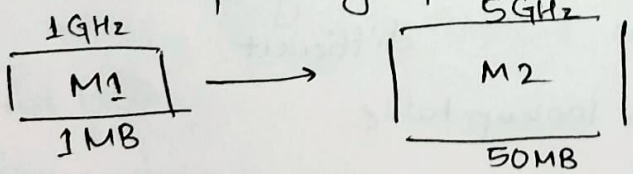
APPLICATIONS

file systems
DB
Search engines
Telcom Networks
Geospatial DB

Vertical and Horizontal Scaling

changing system size in regards to needs.

VERTICAL : Adding additional resources to the system { scale up/down }
i.e powering up the current machines



PROS

* Single Node
* Less time & expertise
* Easier to maintain

CONS :

* Higher risk of data loss in case of failure
* Downtime
* Limited upgradability

HORIZONTAL SCALING

Changing the number of nodes (scaling in/out)

Distributed workload

Low downtime

Load balancing

increased complexity

Expensive, more efforts to maintain

SHARDING : Dividing a large database into a smaller chunk that more manageable.

The shards can be hosted seperately

Components : i) Shards : Logical partitions of data

ii) Shard key : Column that determines the distribution

iii) Shard Nothing Architecture :

Independent op. of shards & unaware of others.

Methods

i) Range Based : specified Ranges

* Easy to implement
* Can lead to uneven data distribution

ii) Hashed Sharding : using a hash function & key

* Even distribution
* Reassignment is difficult.

iii) Directory sharding : uses a lookup table

iv) Geo Sharding : partition based on geo loc.

Optimizing sharding

factors: Cardinality : unique value ↑ ↑shards
Frequency : Avoid high freq. to reduce hotspots
Monotonic change : Rate of change of keys

ROLE BASED ACCESS CONTROL

components : 1) Role : could be a job function within an org.
2) Permissions : access rights & privileges.
3) Users : individuals

User Assignment , Role Permissions , Role Hierarchies

PROS : simplified Administration
security
Scalability
Compliance

CONS: Complexity
Inflexible
Delegation
Granularity

DATA MASKING : obfuscating data (Hiding)

i) Static Data Masking
changing words when data is at rest.
usually before transfers commonly during development and testing

ii) Dynamic Data Masking
Both for full / partial Masking
could be done during data transfers.

iii) Deterministic Data Masking
value is replaced by similar value in the same row

iv) On the fly:
Similar to dynamic data M. but done one value at a time.

TECHNIQUES

i) Substitution : replace with a fake yet realistic value
↑ Makes realistic data
↓ Not applicable to large data that is unrelated

ii) Averaging : can be used with numeric data
iii) Shuffling
iv) Encryption
v) Null out / Deletion
vi) Redaction : replacing with generic values.
vii) Data Aging

DATA ENCRYPTION

plaintext → ciphertext

i) Symmetric key : same key on both sides (private encryption)
eg: AES
ii) Asymmetric key : diff keys public & private
they are mathematically linked
eg: RSA, DSA