Databases

- Problem Statement: can we build software w/out using dbs?
  - WhatsApp
    - ↳ need place to store info: list of people's names, & messages associated w/ them
      - need this storage to be permanent
    - ↳ could use simple file storage, but it has limits

- Limitations of File Storage
  - We can't offer concurrent management to seperate users accessing the storage files from diff locations
  - We can't grant different access rights to diff users
  - How will the system scale for 1000s of entries?
  - How will we search content for diff users in a short time?

- Solution : Databases
  - Organized collection of data that can be managed & accessed easily
  - DBs are created to make it easier to store, retrieve, modify, & delete data in connection w/ diff data processing procedures
  - SQL (relational dbs)
    - ↳ organized & have predetermined schemas: phone books w/ contact numbers & addresses
  - NoSQL (non-relational dbs)
    - ↳ file directories that store anything from contact info to shopping preferences: unstructured, scattered, dynamic schema

- Advantages
  - Manage large data
  - Retrieve accurate data
  - Easy to update (using data manipulation language)
  - Security: only authorized users can access the DB
  - Data Integrity: diff constraints on data
  - Availability: DBs can be replicated on diff servers, & can be concurrently updated. Replicas ensure availability
  - Scalability: DBs are divided (using partitioning) to manage the load on a single node

- Relational DBs (RDBs)
  - adhere to particular schemas before storing data
  - data stored in RDBs has prior structure
  - This model organizes data into one or more relations (also called tables) w/ a unique key for each tuple (instance)
  - Instances
    ↳ stored in rows
    ↳ attributes of instances stored in columns
    ↳ each tuple has a unique key — a tuple in one table can be linked to a tuple in other tables by storing the primary keys in other tables (known as foreign keys)
  - SQL used for manipulating data on the DB
  - RDBs are simple, robust, flexible, & can scale well while managing generic data well
  - RDBs provide ACID:
    ↳ Atomicity: A transaction is considered an atomic unit, therefore, either all the statements within a transaction will successfully execute, or none will. If a statement fails, it should be aborted & rolled back
    ↳ consistency: DB should be in a consistent state, & should remain in a consistent state after every transaction

- ex:) If multiple users are viewing a record from a DB, it should return a similar result each time
  - ↳ Isolation: Multiple transactions happening concurrently shouldn't affect each other
    - Final DB state should be the same as the transactions were executed sequentially
  - ↳ Durability: System should guarantee that completed transactions will survive permanently in the DB even in sys failure events
- DB management systems: MySQL, Oracle DB, Microsoft SQL Server, IBM DB2, Postgres, SQLite

• Why relational DBs?
- RDBs are the go to — for structured data
- flexibility: In the context of SQL, data definition language (DDL) gives us flexibility to mod the DB
- Reduced Redundancy: info related to a specific entity appears in one table while relevant data to that specific entity appears in other tables linked thru foreign keys
  - ↳ this process: normalization, additional benefit of removing an inconsistent dependency.
- Concurrency: Important factor while designing an enterprise DB
  - ↳ data is read & written by many users @ the same time
  - ↳ we need to coordinate these actions to avoid inconsistencies in data — for ex:) double booking of hotel rooms
  - ↳ This concurrency concept is handled thru transactional access to the data
  - ↳ Transactions are atomic ops: so it works in

error handling to either roll back or commit a transaction on successful execution.
- Integration: process of aggregating data from multiple sources
  ↳ commonly used: shared DB where multiple apps store data
  ↳ This way, all apps can easily access each other's data while the concurrency control measures handle the access of multiple apps
- Backup & Disaster Recovery
  ↳ Export & Import ops make backup & restoration easy
  ↳ cloud based DBs perform continuous mirroring to avoid data loss

• Drawback: Impedence Mismatch
- What is it? Diff b/t the relational model & in-memory data structures
- The relational model organizes data into a tabular structure w/ relations & tuples
- SQL ops on this structured data yields relations aligned w/ relational algebra
- It has limitations:
  ↳ values in a table take simple values that can't be a structure or list
  ↳ In-memory, complex data structures can be stored
  ↳ to make complex structures compatible w/ the relations, we would need a translation of the data in light of relational algebra
  ↳ so: Impedence mismatch requires translation b/t 2 representations

- Non-relational databases (noSQL)
  - used in apps that require semi/un-structured data
  - Low latency & flexible data models
  - Simple Design:
    ↳ don't deal w/impedence mismatch
  - Horizontal Scaling:
    ↳ NoSQL is preferred due to its ability to run DBs on a large cluster
    ↳ If num users ↑ by alot, NoSQL makes it easy to scale since data is stored on a single document instead of multiple tables over nodes
    ↳ NoSQL DBs spread data across many nodes & balance queries automatically, node failure results in transparent replacement of the failed node without application disruption.
  - Availability: node replacement performed w/out app downtime
  - Support for unstructured & semistructured data
  - Cost: noSQL DBs are free

- Types of noSQL databases
  - Document DB: mongoDB, Google Cloud Filestore
    ↳ Tree Structure: can include maps, collections, scalars
    ↳ designed to work w/ XML, JSON, BSON, etc.
  - Graph DB: Neo4J, OrientDB, Infinite Graph
    ↳ Graph structure: nodes represent entities, edges rep. relationships b/t entities
    ↳ use case: social apps
  - Columner DB: Cassandra, HBase, HyperTable, Amazon Simple DB
    ↳ efficient for large num of aggregation & data analytics queries

↳ reduces amount of data required to load from the disk
- Key-Value DBs: Amazon DynamoDB, Redis, Memcached DB
  ↳ stores data in key-value pairs
  ↳ useful in session based apps (web apps, store user info in a DB during a session)
  ↳ unique ID for a user session for easy access

- Drawbacks of noSQL DBs
  - lack of standardization: no "relational algebra" like w/ RDBS.
  - Consistency
    ↳ we can't have strong data integrity (like primary & referential integrity in relational DBs)
    ↳ Data might not be strongly consistent but slowly converging using a weak model like eventual consistency

- Choosing the right DB

relational	non-relational
- Structured Data - ACID is required - Size of data is small & can be stored on a node	- Unstructured data - need to serialize & deserialize data - size of data is large