Databases- overview

1. Hierarchical Model, Network/Graph Model, Relational Model, Object/Relational Model, Object-Oriented Model, Semi Structured Model, Associative Model, Entity-Attribute-Value data Model, Context Model
2. A relational DBMS is special system software that is used to manage the **organization, storage, access, security and integrity of data**. This specialized software allows application systems to focus on the user interface, data validation and screen navigation. When there is a need to **add, modify, delete or display data**, the application system simply makes a "call" to the RDBMS.
3. A table is a collection of related data held in a structured format within a database. It consists of fields (columns), and rows.
4. Primary/Foreign Keys
    - Primary Key - Every database table should have one or more columns designated as the primary key. The value this key holds should be unique for each record in the database.
    - Foreign Key - These keys are used to create relationships between tables.
5. RDBMS relationships
    - **One-to-one:** Both tables can have only one record on either side of the relationship.
    - **One-to-many:** The primary key table contains only one record that relates to none, one, or many records in the related table.
    - **Many-to-many:** Each record in both tables can relate to any number of records (or no records) in the other table.
6. The normalization of database schema usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database using the defined relationships.
7. Integrity constraints ensure data integrity in the database tables and enforce data rules which cannot be violated
8. Advantages/Disadvantages of using index
    - Advantages :
        - Low overhead (locks, speed) in sliding large amounts of data into separate table for truncation (our primary concern)
        - Increased performance.
    - Disadvantages:
        - Low overhead (locks, speed) in sliding large amounts of data into separate table for truncation (our primary concern)
        - Increased performance.
9. The Structured Query Language is a special-purpose programming language designed for managing (creation, modification, extraction, manipulation) data held in a relational database management system (RDBMS).
10. Transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a database must either complete in its entirety or have no effect whatsoever.
    - Examples:
        - 1) Verify account details.

        - 2) Accept withdrawal request
        - 3) Check balance
        - 4) Update balance
        - 4) Dispense money

11. It is no relational database. The database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. The data is stored as documents. The documents do not have a fixed structure. The document is a single record.
12. Non-Relational Data Models:
    - Document model - Set of documents, e.g. JSON strings
    - Key-value model - Set of key-value pairs
    - Hierarchical key-value - Hierarchy of key-value pairs
    - Wide-column model - Key-value model with schema
    - Object model - Set of OOP-style objects
13. NoSQL databases:
    - Redis
        - Redis is an in-memory store: all your data must fit in memory. RDBMS usually stores the data on disks, and cache part of the data in memory. With a RDBMS, you can manage more data than you have memory. With Redis, you cannot.
        - Redis is a data structure server. There is no query language (only commands) and no support for a relational algebra. You cannot submit ad-hoc queries (like you can using SQL on a RDBMS). All data accesses should be anticipated by the developer, and proper data access paths must be designed. A lot of flexibility is lost.
        - Redis offers 2 options for persistency: regular snapshotting and append-only files. None of them is as secure as a real transactional server providing redo/undo logging, block checksuming, point-in-time recovery, flashback capabilities, etc ...
        - Redis only offers basic security (in term of access rights) at the instance level. RDBMS all provide fine grained per-object access control lists (or role management).
        - A unique Redis instance is not scalable. It only runs on one CPU core in single-threaded mode. To get scalability, several Redis instances must be deployed and started. Distribution and sharding are done on client-side (i.e. the developer has to take care of them). If you compare them to a unique Redis instance, most RDBMS provide more scalability (typically providing parallelism at the connection level). They are multi-processed (Oracle, PostgreSQL, ...) or multi-threaded (MySQL, Microsoft SQL Server, ...), taking benefits of multi-cores machines.
    - MongoDB
        - Pros:
            1. Schema-less. If you have a flexible schema, this is ideal for a document store like MongoDB. This is difficult to implement in a performant manner in RDBMS
            2. Ease of scale-out. Scale reads by using replica sets. Scale writes by using sharding (auto balancing). Just fire up another machine and away you go.

Adding more machines = adding more RAM over which to distribute your working set.

3. Cost. Depends on which RDBMS of course, but MongoDB is free and can run on Linux, ideal for running on cheaper commodity kit.
4. You can choose what level of consistency you want depending on the value of the data (e.g. faster performance = fire and forget inserts to MongoDB, slower performance = wait til insert has been replicated to multiple nodes before returning)

- Cons:
    1. Data size in MongoDB is typically higher due to e.g. each document has field names stored it
    2. Less flexibity with querying (e.g. no JOINs)
    3. No support for transactions - certain atomic operations are supported, at a single document level
    4. At the moment Map/Reduce (e.g. to do aggregations/data analysis) is OK, but not blisteringly fast. So if that's required, something like Hadoop may need to be added into the mix
    5. Less up to date information available/fast evolving product

- CouchDB
    - Pros:
        1. Simplicity. You can store any JSON data, and each document can have any number of binary attachments.
        2. Thanks to map/reduce, querying data is somewhat separated from the data itself. This means that you can index deeply within your data, and on whether or not something exists, and across types, without paying a significant penalty. You just need to write your view functions to handle them.
    - Cons
        1. Arbitrary queries are expensive. To do a query that you haven’t created a view for, you need to create a temporary view. This can be solved to some extent by using Lucene.
        2. There is a bit of extra space overhead with CouchDB compared to most alternatives.