Choosing a Database

Background

There are a lot of different options for databases, and each of them has a unique implementation! How can we decide which to choose for our systems design interview?

Review: Indexes

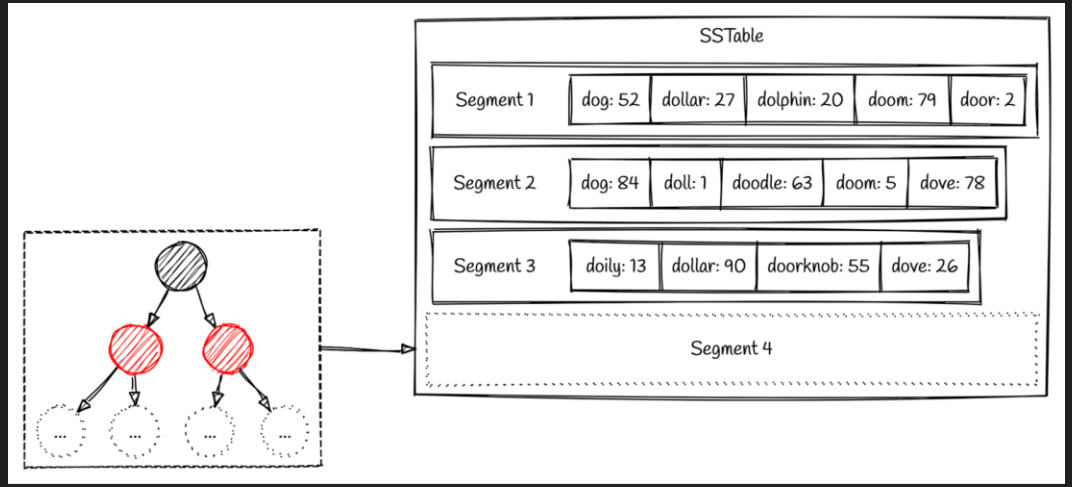
A database index is used for the purpose of speeding up reads conditioned on the value of a specific key! Be careful to not overuse indexes, as they slow down database writes.

Two main types:

- LSM Trees + SSTables
- B-Trees

Review: LSM Trees and SSTables

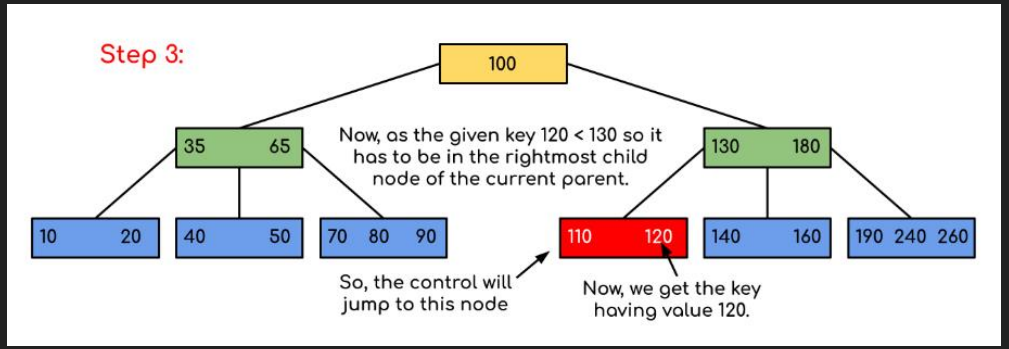
- Writes first go to a balanced binary search tree in memory
- Tree flushed to a sorted table on disk when it gets too big
- Can binary search SSTables for the value of a key
- If there are too many SSTables they can be merged together (old values of keys will be discarded)



Pro: Fast writes to memory!
Con: May have to search many SSTables for value of key.

Review: B-Trees

- A binary tree using pointers on disk
- Writes iterate through the binary tree and either overwrite the existing key value or create a new page on disk and modify the parent pointer to the new page



Pro: Faster reads, know exactly where key is located!
Con: Slow writes to disk instead of memory.

Review: Replication

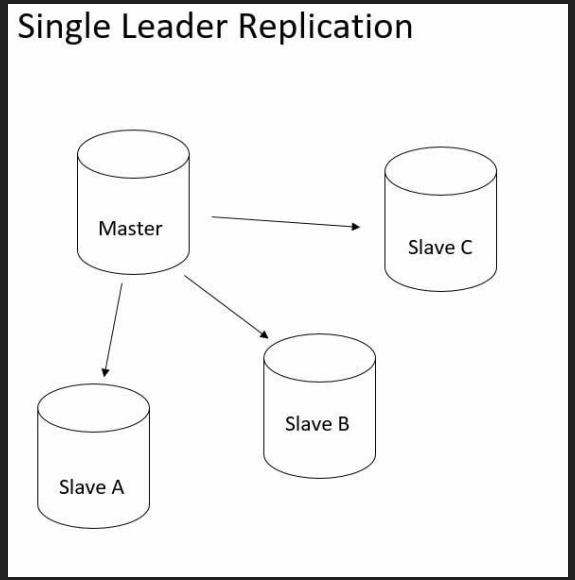
Replication is the process of having multiple copies of data in order to make sure that if a database goes down the data isn’t lost!

Types:

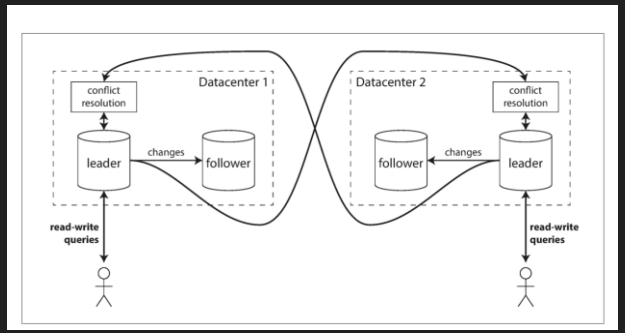
- Single leader replication
    - All writes go to one database, reads come from any database
- Multi leader replication
    - Writes can go to a small subset of leader databases reads can come from any database
- Leaderless replication
    - Writes go to all databases, reads come from all databases

Review: Replication Continued

Single leader replication is useful to ensure that there are no data conflicts, all writes will go to one node



Leaderless and multileader replication is useful for increasing write throughput beyond just one database node (at the cost of potential write conflicts)



Let’s Get Started!

SQL Databases

Key Features:

- Relational/Normalized data - changes to one table may require changes to others
  - E.g. adding an author and their books to different tables on different nodes
  - May require two phase commit! (Expensive)
- Have transactional (ACID guarantees)
  - Excessively slow if you don’t need them (due to two phase locking)
- Typically use B-trees
  - Better for reads than writes in theory

Conclusion: Use SQL when correctness is of more importance than speed

- See banking applications, job scheduling

MongoDb

Key Features:

- Document data model (NoSQL)
  - Data is written in large nested documents, better data locality (if you choose to organize your data in a way that takes advantage of this) - but denormalized
- B-Trees and Transactions supported

Conclusion: Rarely makes sense to use in a systems design interview since nothing is “special” about it, but good if you want SQL like guarantees on data with more flexibility via the document model

Apache Cassandra

Key Features:

- Wide column data store (NoSQL), has a shard key and a sort key
  - Allows for flexible schemas, ease of partitioning
- Multileader/Leaderless replication (configurable)
  - Super fast writes, albeit uses last write wins for conflict resolution
  - May clobber existing writes if they were not the winner of LWW
- Index based off of LSM tree and SSTables
  - Fast writes

Conclusion: Great for applications with high write volume, consistency is not as important, all writes and reads go to the same shard (no transactions)

- See chat application for a good example of when to use

Riak

Key Features:

- Wide column data store (NoSQL), has a shard key and a sort key
  - Allows for flexible schemas, ease of partitioning
- Multileader/Leaderless replication (configurable)
  - Super fast writes, supports CRDTs (conflict free replicated data types)
  - Allows for implementing things like counters and sets in a conflict free way, custom code to handle conflicts
- Index based off of LSM tree and SSTables
  - Fast writes

Conclusion: Great for applications with high write volume, consistency is not as important, all writes and reads go to the same shard (no transactions)

- See chat application for a good example of when to use

Apache HBase

Key Features:

- Wide column data store (NoSQL), has a shard key and a sort key
  - Allows for flexible schemas, ease of partitioning
- Single leader replication
  - Built on top of hadoop, ensures data consistency and durability
  - Slower than leaderless replication
- Index based off of LSM tree and SSTables
  - Fast writes
- Column oriented storage
  - Column compression and increased data locality within columns of data

Conclusion: Great for applications that need fast column reads

- Multiple thumbnails of a youtube video, sensor readings

Memcached and Redis

Key Features:

- Key value stores implemented in memory (Redis a bit more feature rich)
  - Uses a hashmap under the hood

Conclusion: Useful for data that needs to be written and retrieved extremely quickly, memory is expensive so good for small datasets

- Good for caches, certain essential app features (see geo spatial index for Lyft)

Neo4j

Key Features:

- Graph database!
  - As opposed to just using a SQL database under the hood with relations to represent nodes and edges, actually has pointers from one address on disk to another for quicker lookups
  - The former is bad because reads become slower proportional to the size of the index (O log n to binary search), but using direct pointers is O(1)

Conclusion: Only useful for data naturally represented in graph formats

- Map data, friends on social media

TimeScaleDB/Apache Druid

Key Features:

- Time series database!
  - Use LSM trees for fast ingestion, but break table into many small indexes by both ingestion source and timestamp
  - Allows for placing the whole index in CPU cache for better performance, quick deletes of whole index when no longer relevant (as opposed to typical tombstone method)

Conclusion: Very niche but serve their purpose very well

- Great for sensor data, metrics, logs, where you want to read by the ingestor and range of timestamp

New SQL

- VoltDb
  - SQL but completely in memory, single threaded execution for no locking
  - Expensive and only allows for small datasets
- Spanner
  - SQL, uses GPS clocks in data center to avoid locking by using timestamps to determine order of writes
  - Very expensive

Cool discussions to have in an interview, but probably pretty impractical to ever actually suggest!