BST

- If the node belongs to the left subtree, it is less than the parent
- If the node belongs to the right subtree, it is greater than the parent

Traversal of Nodes in BST

1. Inorder: left subtree, current node, right subtree
2. Preorder: current node, left subtree, right subtree
3. Postorder: left subtree, right subtree, current node

Storage/Space

- CPU, Registers, L1 Cache, L2 Cache, RAM, SDD/HDD
- L1 Cache and L2 Cache are faster than RAM because they are closer to the processor
- RAM measured at nanosecond speed level
- SDD/HDD: lots of storage, persistent, can survive power cycle, but SLOW
- Database systems minimize HDD/SDD accesses
- 64 bit integer → 8 bytes
    - To get this value, need to go through 2048 byte block size
- TREE
    - 4 x 8 = 32 bytes
    - If you want to store 7 nodes on the same block, it's 7 x 32 bytes
    - If you store on separate blocks, it's 7 x 2048 bytes
    - In a perfect balanced tree, you never have to read more than 3 blocks of memory

B+ Tree

- In just 2 levels of a B+ tree, you can store 258 nodes and 256 keys
- Nodes in a B+ tree are always going to be at least half full
- Indexing data structures allow us to access data that is stored on disk faster than scanning disk linearly

Hash Tables (like Python dictionary)

- Gamma = load factor = n / m, where n is the number of inserted values and m is the table size
- Hash function: h(k) = k mod m (hash function always mods table size)
    - This ensures that the indices possible are within the size of the table
- Constant amount of work for any k
- Putting things in the table (as long as there are slots), requires constant time work
- Faster than AVL tree (in most occasions)
- Linear search
- Keeping insert as constant time as possible
- Good dispersion = spread has values out
- Each location can be a python list, don't make the whole hash table a python list
- Worst version of dispersion is when you don't spread keys across the table

Benefits of Relational Model (RDBMs)

- Standard data model and query language
- ACID compliance
- Works well with highly structured data
- Handles large amounts of data
- Well understood, lots of tooling/experience
- Increases efficiency with:
    - Indexing
    - Directly controlling storage
    - Column oriented storage versus row oriented storage
    - Query optimization
    - Caching: stores frequently used data in fast memory location
    - Prefetching: predicts/fetches data that might be needed soon
    - Materialized views: database object that stores results of a query
    - Precompiled stored procedures
    - Data replication and partitioning

Transaction Processing

- Transaction: sequence of 1 or more of the CRUD operations performed as a single, logical unit of work
- Entire sequence succeeds → COMMIT
- Entire sequence fails → ROLLBACK or ABORT
- Helps ensure:
    - Data integrity
    - Error recovery
    - Concurrency control
    - Reliable data storage
    - Simplified error handling
- Dirty Read: transaction T1 is able to read a row that has been modified by transaction T2 that hasn't yet executed a COMMIT
- Non-Repeatable Read: two queries in a single transaction T1 execute SELECT but get different values because another transaction T2 has changed data and committed
- Phantom Reads: transaction T1 is running and transaction T2 adds or deletes rows from set T1

Distributed System

- A collection of independent computers that appear to its users as 1 computer
- Computers operate concurrently
- Computers fail independently
- No shared global clock
- Single main node, and can either perform replication or sharding
- Sharding: splitting large amounts of data into smaller chunks (shards) and storing across multiple servers
- Data stored on > 1 node, typically replicated (each block of data available on N nodes)

- Distributed databases can be relational or non-relational
- Network partitioning (process of dividing a network into smaller subnetworks) is inevitable
    - Network failures, system failures
    - System needs to be partition tolerant
    - Partition tolerant: system keeps running even with network partitions

CAP Theorem

- Impossible for a distributed data store to simultaneously provide more than 2 out of 3 guarantees:
    - Consistency: every read receives most recent write/error thrown
    - Availability: every request receives non-error response, no guarantee that response contains most recent write
    - Partition Tolerance: system can continue to operate despite arbitrary network issues
- RDBMs, PostgreSQL, MySQL are Consistent and Available
- MongoDB, HBase, Redis are Consistent and Partition Tolerant
- CouchDB, Cassandra, DynamoDB are Available and Partition Tolerant
- A consistent and available system is a system that always responds with the latest data and every request gets a response, but may not be able to deal with network partitions
- A consistent and partition tolerant system is when the system responds with data from the distribution system, it is always the latest, else data request is dropped.
- An available and partition tolerant system is a system that always sends and responds based on distributed stores, but may not be the absolute latest data.

Vertical Scaling

- Vertical scaling in shared memory architecture has a geographically centralized server and some fault tolerance (via hot-swappable components)
- In shared memory architecture, there is one shared memory and multiple CPUs. To vertically scale, you would add more CPUs.
- Vertical scaling in shared disk architecture has machines connected via a fast network, contention and overhead of locking limit scalability for high-write volumes. However, it is OK for data warehouse applications (high read volumes)
- In shared disk architecture, there is one disk/storage and multiple servers with their own CPU and memory. To vertically scale, you would add more servers.

Replication Lag

- Time it takes for writes on leader to be reflected on all followers
- In synchronous replication, lag causes writes to be slower and system to be more brittle as the number of followers increases
- In asynchronous replication, we maintain availability at the cost of delayed or eventual consistency (inconsistency window)