- Maven
    - a project management tool
    - can import dependencies into the project by importing repository in pom.xml
        - local: sits inside local machine, default folder location is ~/.m2
        - central: repository from the official maven community
        - remote: other artifacts deployed on the other domains
- Git
    - version control tool
- Basic Data Types
    - primitive types: int, long, char, double, float, long, byte, boolean
- String/StringBuilder/StringBuffer
    - a string object is immutable, while objects of StringBuilder and StringBuffer is mutable
    - StringBuilder vs StringBuffer: StringBuffer is thread-safe while StringBuilder is not, due to thread safety, StringBuffer has performance overhead.
- equals() and hashcode()
    - to ensure equals and hashcode contract, in a class we should override both methods when we want to define custom equals method
    - if two objects have the same content (equals return true) then their hashcode must also return the same value. However, if two objects have the same hashcode, their content might not be the same
- Java Collection
    - LinkedList vs ArrayList:
        - Add operation: Both O(1)
        - Remove Operation: ArrayList is O(n) while LinkedList is O(1)
        - Get Operation: ArrayList is O(1) while LinkedList is O(n)
    - list vs set
        - list is ordered while set is not
        - set does not allow duplicate element
        - remove complexity: list is O(n) set is O(n)/O(logn)
    - HashMap and HashSet:
        - HashSet is implemented using HashMap, each element is a key with null value

- Comparator vs Comparable
    - comparator can be used to create many sorting sequences while comparable is used to create a single sorting sequence (compare() vs compareTo())
- JVM
    - a specification that provides runtime environment in which java bytecode can be executed
    - consists of classloader, memory data area and execution engine
- Java ClassLoader
    - responsible for loading classes to JVM dynamically during runtime

        - bootstrap class loader: loading JDK internal classes, parent of all other classloader instances
        - extension class loader: child of bootstrap class loader, takes care of loading extensions of the standard core java classes
        - system class loader: child of extension class loader, loads files found in the classpath environment variable, -cp command line option
- Garbage Collector
    - regularly cleans up unreferenced objects in heap, default GC is parallel GC
    - Heap structure is divided into generations: young (eden, survivor 0, survivor 1), old, permanent. Minor GC happens in young generation, major GC happens in old generations
- Keywords
- OOP
    - JAVA is an OOP language. It centers around 4 concepts
        - Abstraction: only essential characteristics of an object are presented to users
        - Encapsulation: wrapping the implementation details in a unit
        - Polymorphism: achieved by overloading and overriding
            - method overload: methods with same name but different parameters and return types
            - method override: happens when child classes want to give a new implementation of the the same method from the parent class
        - Inheritance: A class can act as an child class by extending another (abstract) class, in doing so the child class has access to parent class methods and properties
- Exception
    - customize exception: create a class extending Exception class
    - checked vs unchecked: checked happens at compile time, unchecked happens at runtime
    - throw vs throws: throw is a keyword used in a statement, while throws used at the method definition body
- Generics
    - ensure type corrections at the compile time (type erasure)
    - use <> to specify parameter types
    - bounded generics and wildcards:
        - <T extends E> T class is a subtype of E
        - <? extends E> allows class E or sub classes of E
        - <? super T> allows class T and upper class of class T
- IO Stream
    - java.io package. stream is a continuous flow of data, usually connected to a data source, like a file or database connection. There are two kinds of streams, input and output.
- Serialization and Deserialization
    - done by implements Serializable

    - converts objects into a byte stream, deserialization is the opposite.
    - transient keyword: prevents a field to be serialized
- Java 8 features
    - Lambda expression − Adds functional processing capability to Java.
    - Method references − Referencing functions by their names instead of invoking them directly. Using functions as parameter.
    - Default method − Interface to have default method implementation.
    - Stream API − New stream API to facilitate pipeline processing.
    - Optional − Emphasis on best practices to handle null values properly.
- Java Multi-threading
    - lifecycle of a thread:
        - New − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
        - Runnable − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
        - Waiting − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
        - Timed Waiting − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
        - Terminated (Dead) − A runnable thread enters the terminated state when it completes its task or otherwise terminates.
    - Two ways to create a thread:
        - extends Thread class
        - implements Runnable
    - Runnable vs Callable
        - Runnable: overrides run()
        - Callable: overrides call()
    - ThreadPoolExecutor
        - customized thread pool
            - corepoolsize: minimum number of workers to keep alive
            - maximumpoolsize: maximum threads to be created
            - keepalivetime: when thread count exceeds corepoolsize, the alive time to keep the idle thread running
            - threadFactory: to create a

- an extensible thread pool implementation with lots of parameters and hooks for fine-tuning.

ArrayList vs LinkedList
- ArrayList is implemented based on a dynamically resizable array; linked list is based on a linear data structure in which elements are not stored in a contiguous memory location, each element is linked using pointers.
- for get operation, ArrayList is faster than LinkedList, because we can access element by index, linked list however, need to be traversed by pointers to access an element
- for add and remove operations, LinkedList is faster than ArrayList because ArrayList might need to ensure the array capacity and copy the array internally,
ArrayList -> add: O(1) at best, O(n) at worst; remove(): O(n)
LinkedList -> add(): O(1) add(index, elem): O(n); remove():O(n)

HashMap
Hashmap is a data structure that stores key-value mapping pairs, internally it converts each key to a hashcode which is stored inside a bucket of an array with default size of 16.
HashMap is internally implemented using array + linked list + redblacktree (jdk1.8 added redblacktree). When linked list size is at TREEIFY_THRESHOLD (>8), it would convert the linked list to a redblacktree, when nodes of redblacktree is less than UNTREEIFY_THRESHOLD (<6), it would convert back to linkedlist.

HashMap vs HashTable
They both implement the Map interface.
- HashMap allows one null key and multiple null values whereas Hashtable doesn't allow any null key or value.
- HashTable is thread-safe, while HashMap is not.(synchronized vs unsynchronized)

HashTable vs ConcurrentHashMap
Instead of locking the whole map like HashTable, ConcurrentHashMap allows concurrent read and thread-safe update operations. To perform read operation thread won't require any lock but to perform update operation thread require a lock, but it is the lock of a bucket, by default ConcurrentHashMap maintain 16 locks for a map.

LinkedHashMap
LinkedHashMap is a map data structure whose keys are preserved in insertion order, internally implemented by the doubly linked list, which means each node entry has two more pointers which point to the previous node and the next node to track the insertion order.

TreeMap
TreeMap is a map data structure whose keys are ordered by natural ordering or a comparator. It has O(logn) lookup and insertion time. Internally it is implemented by a red black tree.

HashSet Internal Working

Hashset is implemented based on HashMap, elements in the hash are actually keys in the HashMap, where each mapping value is by default null.

ArrayDeque
ArrayDeque is a special queue data structure that can be inserted and removed at both ends of the queue. Internally it is implemented by a dynamically resizable array

fail-fast vs fail-safe iterators

Iterators in java are used to iterate objects in java collections. Fail-fast means that java would immediately throw concurrentmodificationexception if there is another thread modifying the collection while the current thread is iterating. Fail-safe means that java would first copy the content of the collection then iterate on the copy, this way would not trigger the concurrentmodificationexception, but at the same time, iterator cannot access the content after modifications at the time of iterating.

SQL vs NoSQL

- SQL applies to relational databases. Relational database is a collection of data with predefined relationships within, the data items are organized as a table with rows and columns. Non Relational databases however, can have many different structures and handle unstructured data; they can be graph-based, document-based,key-value pairs, or wide-column stores.
- Predefined schema means that the structure of data is static, while dynamic schema can change the structure while adding items into the NoSQL database.
- Most SQL databases can be scaled vertically, by increasing the processing power of existing hardware. NoSQL databases use a master-slave architecture which scales better horizontally, with additional servers or nodes. These are useful generalizations, but it’s important to note:
- SQL databases can be scaled horizontally as well, though sharding or partitioning logic is often the user’s onus and not well supported.
- NoSQL technologies are diverse and while many rely on the master-slave architecture, options for scaling vertically also exist.
- Savings made using more efficient data structures can overwhelm differences in scalability; most important is to understand the use case and plan accordingly.
- ACID vs CAP
    - ACID addresses an individual node's data consistency
    - CAP addresses cluster-wide data consistency
    - ACID means Atomicity, Consistency, Isolation and Durability
        - Atomicity:guarantee that either all of the transaction succeeds or none of it does.
        - Consistency: guarantee that all data will be consistent. All data will be valid according to all defined rules, including any constraints, cascades, and triggers that have been applied on the database.

        - Isolation: all transactions will occur in isolation. No transaction will be affected by any other transaction.
        - Durability: once a transaction is committed, it will remain in the system – even if there's a system crash immediately following the transaction.
    - CAP means Consistency, Availability, and Partition-Tolerance
        - Consistency: All Nodes Have Same Data via Eventual Consistency
        - Availability: system continues to operate despite arbitrary message loss or failure of part of the system
        - Partition-Tolerance: System continues to operate despite arbitrary message loss or failure of part of the system
- The difference between relational and hierarchical databases lies in the data structures. While the hierarchical database architecture is tree-like, data in a relational database is stored in tables with a unique identifier for each record. A relational database structure facilitates easy identification and access of data in relation to other data points in the database. The tables are separate from physical storage structures, which enables database administrators to alter physical data storage without reorganizing the database tables themselves.
    - Hierarchical data store advantages and disadvantages:
        - The key advantage of a hierarchical database is its ease of use. The one-to-many organization of data makes traversing the database simple and fast, which is ideal for use cases such as website drop-down menus or computer folders in systems like Microsoft Windows OS.
        - The major disadvantage of hierarchical databases is their inflexible nature. The one-to-many structure is not ideal for complex structures as it cannot describe relationships in which each child node has multiple parents nodes. Also the tree-like organization of data requires top-to-bottom sequential searching, which is time consuming

Short Description of mongoDB:

mongoDB is a nonrelational document-based database. Documents consist of key-value pairs, with values being various kinds of data.
Due to mongoDB's being nonrelational database, it has a dynamic schema that allows users to add additional data to the key.

- **CAP:**
    - Consistency simply means, when you write a piece of data in a system/distributed system, the same data you should get when you read it from any node of the system.

- Availability means the system should always be available for read/write operation.
- Partition-tolerance means, if there is a partition between nodes or the parts of the cluster in a distributed system are not able to talk to each other, the system should still be functioning.
- We generally categorize RDBMS in **CA**. Because Relational databases are a single node system and hence we do not need to worry about partition tolerance and hence if RDBMS server is up and running, it will always respond success for any read/write operation.

mongoDB with CAP:

Not with A:

MongoDB supports a “single master” model. This means you have a master node and a number of slave nodes. In case the master goes down, one of the slaves is elected as master. This process happens automatically but it takes time, usually 10-40 seconds. During this time of new leader election, your replica set is down and cannot take writes.

MongoDB can always be Consistent based on how you configure your client and the way you write data(using write options) and can be always available for reads but you can never make write always available, there will always be downtime when:
a) the new leader is getting elected
b) the client driver disconnects from the leader

Range partitioning with mongoDB:

MongoDB uses the shard key associated to the collection to partition the data into chunks. A chunk consists of a subset of sharded data. Each chunk has a inclusive lower and exclusive upper range based on the shard key.

MongoDB splits chunks when they grow beyond the configured chunk size. Both inserts and updates can trigger a chunk split.

Does mongoDB support ACID?

MongoDB 4.0 introduces multi-document transactions for replica sets and can be used across multiple operations, collections, and documents. The multi-document transactions provide a globally consistent view of data, and enforce all-or-nothing execution to maintain data integrity. The transactions will have following properties

- When a transaction is committed, all data changes made in the transaction are saved.
- If any operation in the transaction fails, the transaction aborts
- When a transaction aborts/aborted, all data changes made in the transaction are discarded.
- Until a transaction is committed, no write operations in the transaction are visible outside the transaction.

In short, Yes for Atomicity.
In MongoDB, Transactions (also called multi-documents transactions) are associated with a session. That is, you start a transaction for a session. At any given time, you can have at most one open transaction for a session.
You can not lock the entire collection for writes. You may want to create multiple transactions to ensure that writes are not interlacing/overriding between your processes. MongoDB uses

Optimistic Locking

- Optimistic Locking
    - is a concurrency control
        - begin: record a timestamp of transaction
        - modify: read data, tentatively write changes
        - Validate: Check whether other transactions have modified data that this transaction has used (read or written). This includes transactions that completed after this transaction's start time, and optionally, transactions that are still active at validation time.
        - Commit/Rollback: If there is no conflict, make all changes take effect. If there is a conflict, resolve it, typically by aborting the transaction

Memcached vs Redis

- Both memcached and Redis are open-source in-memory key-value data store, both belong to the NoSQL family
    - in-memory data store: database that keeps the whole dataset in RAM, each time we query a database, we don't need to access the disk, thus vastly decreasing the response time. But at the same time in-memory databases also risk losing the data when a process or server is down or rebooted, we can minimize the risks by persisting write operations in disks using snapshots or taking logs.
        - For non-change operations (retrieve data), in-memory databases do not use the disks.
        - For change operations (update, add or delete), in-memory databases write to main memory and also persist on disks.
            - Writing to disks: Transactions are applied to the transaction log in an append-only way. (When addressed in this append-only manner, disks are pretty fast. 100mb/sec)

- Data Partitioning
    - Both Redis and Memcached are able to distribute the data across multiple nodes
- Value data structures
    - In addition to strings, Redis supports lists, sets, sorted sets, hashes, bit arrays, and hyperloglogs.
- Multithreading
    - Memcached supports multithreading, and it can scale up the computing usage.
- Snapshots
    - Redis supports snapshots, in case when the server/process is down or rebooted.
- Replication
    - Redis lets you create multiple replicas of a Redis primary. This allows you to scale database reads and to have highly available clusters.
- Transactions
    - Redis supports transactions which let you execute a group of commands as an isolated and atomic operation.

View in SQL:
In SQL, a view is a virtual table based on the result-set of an SQL statement. Views do not store any data of their own but display data stored in other tables.
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

Syntax:
create a view
```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

update a view
```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

drop a view
```
DROP VIEW view_name;
```

Stored Procedure in SQL:
A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

Stored Procedure vs View

A view is mostly used when only a SELECT statement is needed. Views should be used to store commonly-used JOIN queries and specific columns to build virtual tables of an exact set of data we want to see. Stored procedures hold the more complex logic, such as INSERT, DELETE, and UPDATE statements to automate large SQL workflows.

Stored Procedure vs Trigger:

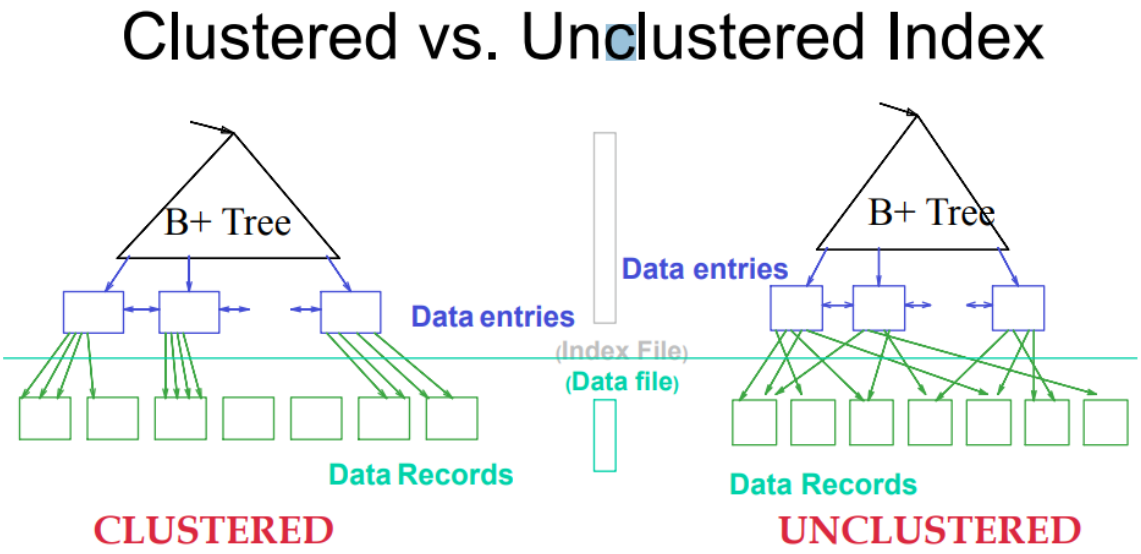
A trigger is a stored procedure that runs automatically when various events happen (eg update, insert, delete). A stored procedure is invoked by calling it explicitly.

View vs Materialized View:

Materialized views are **disk based** and are updated periodically based upon the query definition. They are similar to regular views, in that they are a logical view of your data (based on a select statement), however, the **underlying query result set has been saved to a table**. The upside of this is that when you query a materialized view, **you are querying a table**, which may also be indexed. The **downside though is that the data you get back from the materialized view is only as up to date as the last time the materialized view has been refreshed**.

Views are **virtual only** and run the query definition each time they are accessed. The upside of a view is that it will **always return the latest data to you**. The **downside of a view is that its performance** depends on how good a select statement the view is based on.

B+ tree vs B Tree



They are both self-balancing trees that sort the internal nodes in order.

In a B tree search keys and data are stored in internal or leaf nodes. But in a B+-tree data is stored only in leaf nodes.
Full scan of a B+ tree is very easy because all keys are found in leaf nodes. Full scan of a B tree requires a full traversal since keys might not be at the leaf level.
In a B tree, data may be found in leaf nodes or internal nodes. Deletion of internal nodes is very complicated. In a B+ tree, data is only found in leaf nodes. Deletion of leaf nodes is easy.
Insertion in B-tree is more complicated than B+ tree.
B+ trees store redundant search keys but B-tree has no redundant value.
In a B+ tree, leaf node data is ordered as a sequential linked list but in a B tree the leaf node cannot be stored using a linked list. Many database systems' implementations prefer the structural simplicity of a B+ tree.