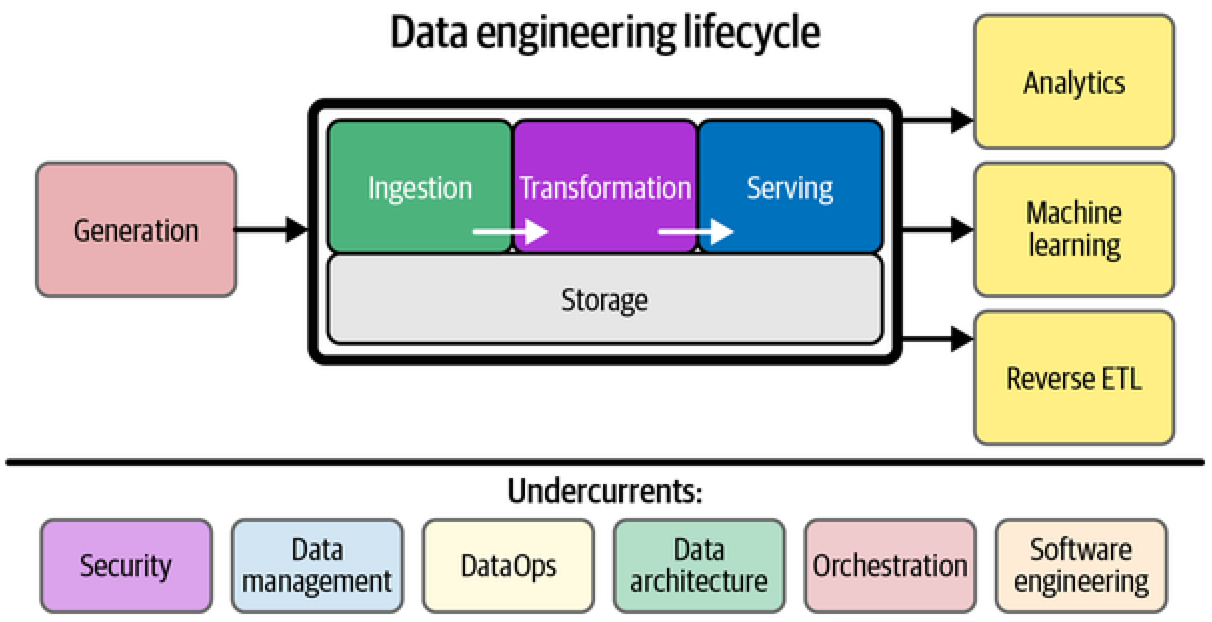
Fundamentals of Data Engineering Chapter 6

Favorite	☐
Archived	☐
Fleeting	☐
Area/Resource	
Project	
Type	
Review Date	
Image	
URL	
Created	@April 11, 2023 11:30 AM
Updated	@April 14, 2023 12:05 PM
Root Area	
Project Area	
Updated (short)	04/14/2023
Pulls	
Resource Pulls	
Project Archived	
URL Base	
🍳 Recipe Divider	🥗🥗🥗 RECIPE BOOK PROPERTIES 🥗🥗🥗
🍳 Recipe Tags	
📚 Book Divider	📚📚📚 BOOK TRACKER PROPERTIES 📚📚📚
📚 Author	

📚 Date Started	
📚 Date Finished	
📚 Book Status	
📚 Rating	

💡 As data gets stored multiple times in the Data Engineering lifecycle, it is important to understand the use cases and particularities of available storage solutions in choosing your data architecture.



Raw ingredients of storage

💡 To understand data storage solutions we will focus on the raw ingredients of storage. These are the lower levels of abstraction that will persist across the different use cases and solutions, and compose storage systems. It is essential to understand the basic characteristics of physical storage technologies to assess the trade-offs inherent in any storage architecture.

- HDD

- SSD
- RAM
- Networking
- Serialization
- Compression
- CPU

Storage systems

💡 In practice, we don’t directly access system memory or hard drives. Storage systems abstracts away the complexity of accessing them by providing mechanisms and access paradigms.

- HDFS
- Cache/Memory based storage
- RDBMS
- Object Storage
- Streaming storage

Storage abstractions

💡 These are the highest level abstractions, which include components and mechanisms involved in the data architecture of these systems. Each system will have their own business use cases and complexities.

- Data Lake
- Data platform
- Data lakehouse
- Cloud data warehouse

Raw ingredients of Data Storage

🧠 Data engineers need to be aware of the complexities of underlying components essential characteristics’, performance considerations, durability and costs.

Magnetic disk drive HDDs

How they work

⚙️ Magnetic disks utilize spinning platters coated of a ferromagnetic film. The film is magnetized by a read/write head during write operations to physically encode binary data. During read operations the head detects the magnetic field and outputs a bitstream, which represents digital data that is then used by the computer.

Performance

💡 Most data is still stored on HDDs as they are significantly cheaper than SSDs. These disks have also seen large improvements in performance, storage density and cost. There are new technologies that are being developed that will further improve HDDs. However there are still some physical limitations to the HDDs.

Limitations

💡 Disk transfer speed, which is the rate at which data can be read and written does not scale in proportion with disk capacity. Disk capacity scales with areal density (GB stored per square inch), whereas transfer speed scales with linear density (bits per inch). This leads to HDDs having huge storage capacity but relatively slow transfer times to read the data.

💡 Seek time is also limited. Seek time is the time taken for the heads to locate the appropriate track on the disk. Thirdly, in order for a particular piece of data to be found on the disk, the disk controller must wait for that data to rotate under the read/write heads. This leads to rotational latency. Finally, the I/O per second (IOPS) is limited which is critical for OLTP systems.

Comparison to SSDs

💡 While there are tricks to increase latency and transfer speed such as using a higher rotational speed, or limiting the radius of the disk platter, however none of these techniques make it remotely comparable to SSDs for random access lookups. This is due to SSDs having no physical rotating disks or magnetic heads to wait for.

✏️ Random access lookup refers to the ability to access a piece of data in a storage structure in constant time regardless of the size of the data.

✏️ Constant time is the property of an algorithm taking the same time to complete regardless of the size of the input. In Big O notation it is referred to as O(1) time complexity.

Parallelism

💡 HDDs while limited as a single unit can achieve massive transfer rates through parallelism. This is the critical idea behind cloud object storage, when data is distributed across thousands of disks in clusters, data transfer rates can increase dramatically as you are no longer limited by the physics of a single HDD. This leads to the primary limitation being network components and CPUs in distributed data systems.

Solid State Drives SSDs

How they work

⚙️ SSDs store data as charges in flash memory cells, but unlike RAM these charges are non volatile, and therefore can persist without constant charge. SSDs do not have any moving physical parts and therefore are not limited by the same constraints. The data is read by purely electrical means.

Performance

Due to the lack of physical constraints SSDs can scale up to massive data transfer speeds and IOPS by slicing storage into partitions with numerous storage controllers running in parallel. This performance makes SSDs popular for OLTP systems that require large IOPS.

Limitations

The cost of SDDs are much higher than HDDs for storage. Making HDDs the preferred solution for storage on large scale data lakes and warehouses.

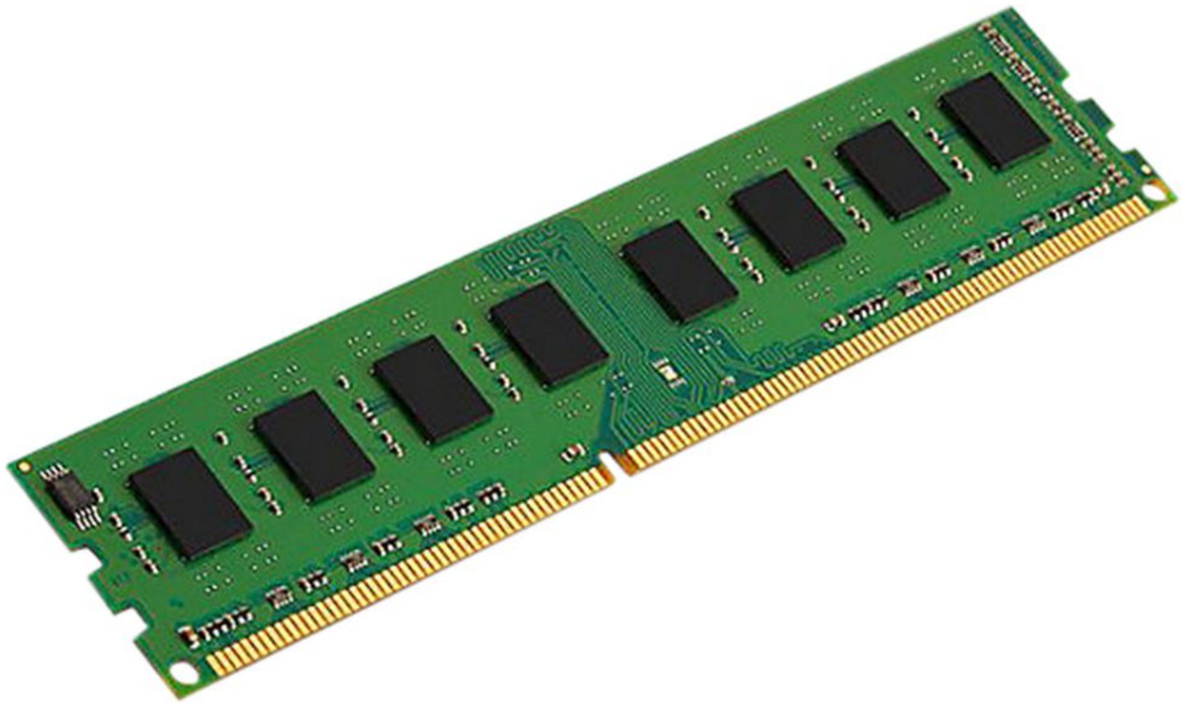
OLAP systems

SSDs can still play a role in OLAP systems as some leverage SSD caching to support high performance queries. As low latency OLAP becomes more popular expect SSD usage to increase in these systems.

Random Access Memory

How they work

RAM is also known as memory and it is a volatile memory (meaning it will not persist without constant charge). RAM is primarily used in conjunction with CPUs where it will access program instructions and data actively used by the CPU. Data is stored in memory cells as a charges, which makes it volatile due to it’s need of a charge.



Characteristics

💡 RAM is attached to the CPU and mapped into the CPU address space. Due to the need of the CPU to access the RAM for executing programs, the CPU is directly attached to the RAM instead of having to access it through I/O subsystems. This makes it extremely effective for random access.

- Ram stores the code that CPUs execute and the data that this code directly processes
- Offer significantly higher transfer speeds and faster retrieval times than SSD storage.
- Much more expensive than SSD

Limitations

💡 RAM is limited by the amount of RAM attached to an individual CPU and memory controller. It is also significantly slower than CPU cache, which is a type of storage directly located in the CPU. Since data in RAM are stored as charges in capacitors, and these leak over time, the data must be frequently refreshed to prevent data loss. However, data engineers must simply worry about bandwidth and retrieval latency.

Networking and CPU

🧠 Increasingly data storage systems are distributed to enhance performance, durability and availability. A cluster of disks can offer massive performance improvements offer individual magnetic disks.

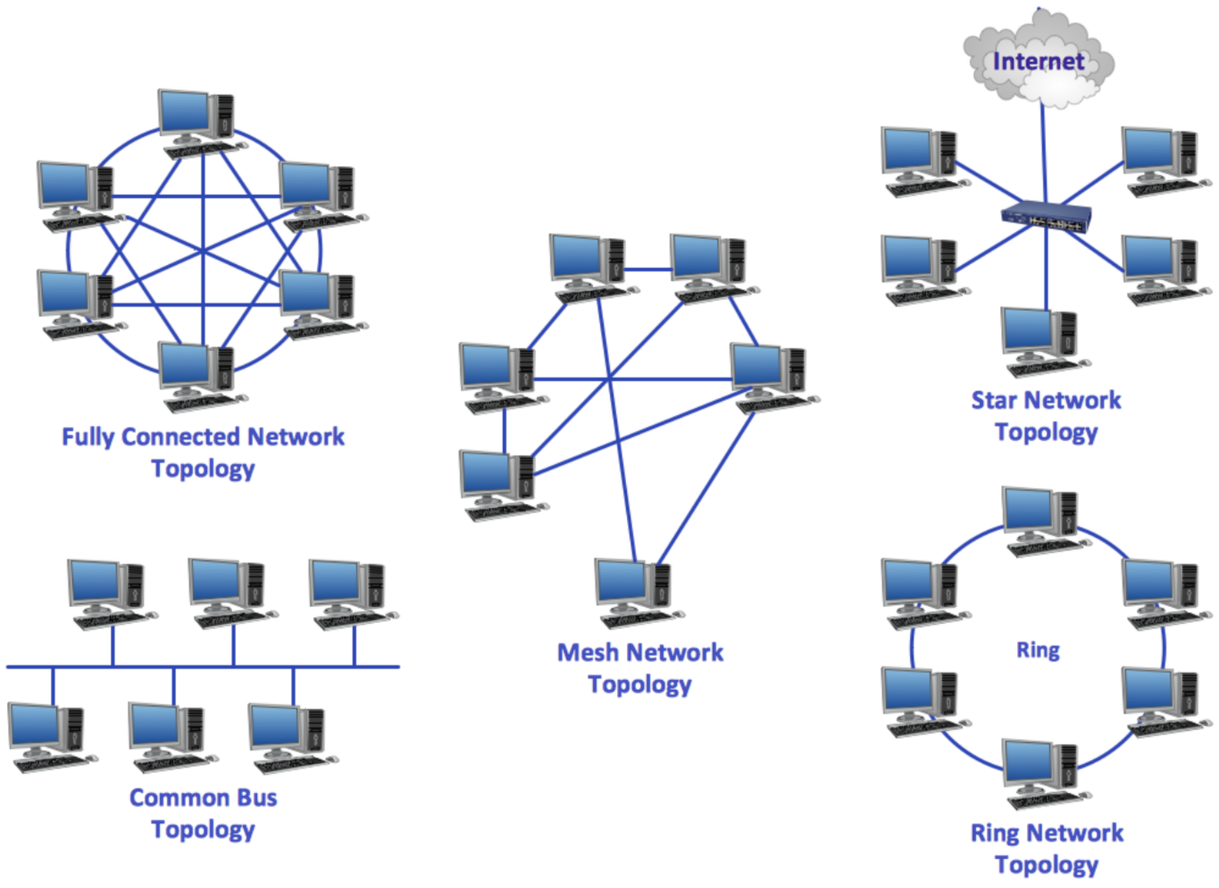
Availability zones

✏️ Availability zones are a standard cloud construct consisting of compute environments with independent power, water, and other resources. Multizonal storage enhances availability and durability of data.

CPUs

💡 CPUs handle the details of servicing requests, aggregating reads and distributing writes. Storage becomes a web app with an API, backend service components, and load balancing. Network device performance and network topology are key factors in realizing high performance.

✏️ Network topology refers to the physical or logical layout of network devices within a network. It defines the way in which devices are connected and the path that data travels over the internet.



🧠 Data engineers need to understand how networking will affect the systems they build and use. Engineers constantly balance the durability and availability achieved by spreading out data geographically versus the performance and costs benefits of keeping storage in a small geographic area.

Serialization

💡 Serialization is a critical element of database design. The decisions around serialization will inform how well queries perform across a network, CPU overhead, query latency and more. Designing a data lake, involves choosing a base storage system and standards for serialization that balance interoperability with performance considerations.

What is serialization?

Serialization refers to the process of flattening and packing data into a standard format that a reader will be able to decode. This is because not all data stored in system memory by a software is in a format suitable for disk storage or transmission over a network.

Serialization formats provide a standard of data exchange. Encoding data and then passing it to a user to decode with a standard library allows for standardized exchanges. A serialization algorithm has logic for handling types, imposes rules on data structure, and allows exchange between programming languages and CPUs. It also has rules of handling exceptions. Each serialization algorithm comes with a set of trade-offs and data engineers can tune these choices to optimize performance and requirements.

Data engineers should become familiar with common serialization practices and formats such as Parquet, hybrid serialization (Hubrid) and in-memory serialization (Arrow).

Compression

Compression makes data smaller, but compression algorithms interact with other details of storage systems in complex ways.

Advantages

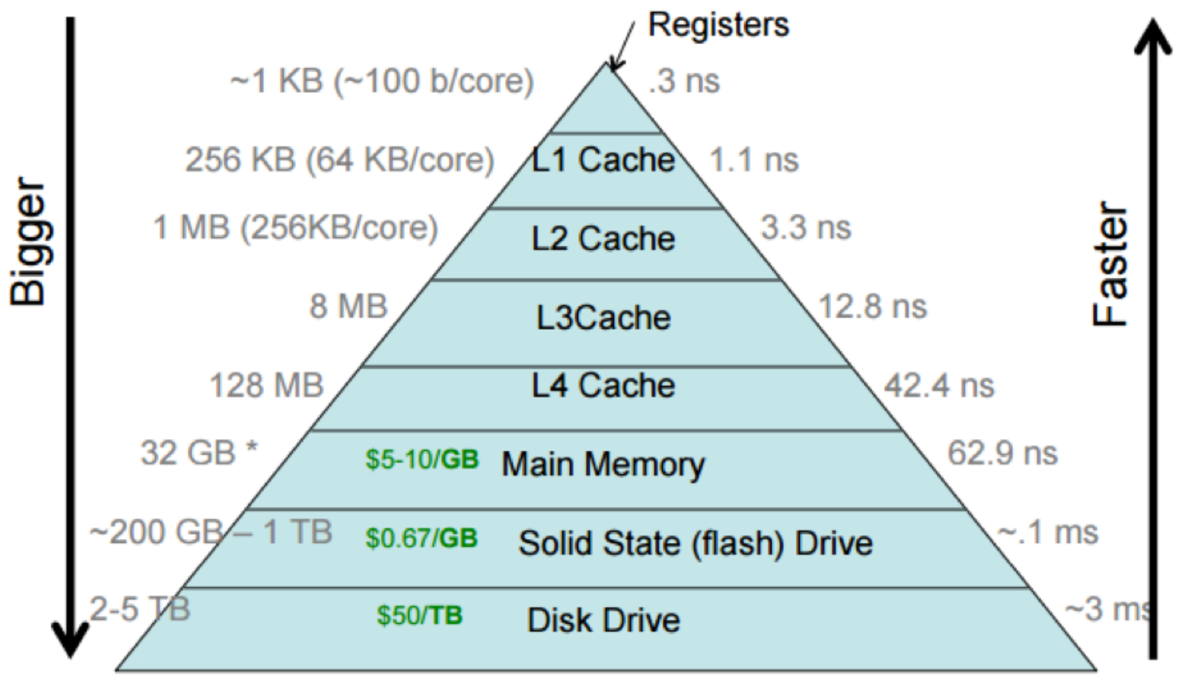
With smaller data taking up less disk space, it can increase the use cases of HDDs as it can increase the scan speeds of disk, and takes up less space. This will also have an impact on network performance.

Disadvantages

💡 Compressing and decompressing data entails extra time and resource consumption to read and write data.

Caching

💡 The core idea of caching is to store frequently or recently accessed data in a fast access layer. The faster the cache, the higher the cost and the less storage space available. Less frequently access data is stored in slower and cheaper storage. Caches are critical for data serving, processing and transformation.



Archive storage

This can be thought of as reverse cache. As it provides inferior access characteristics for lower costs.

Data storage systems

Storage systems are the level of abstraction above the raw ingredients. For example magnetic disks are a raw storage ingredient while HDFS makes use of magnetic disks.

Single Machine versus Distributed Storage

Distributed storage which involves storing data on multiple servers becomes necessary as data storage and access patterns become more complex. Distributed storage coordinates the activities of multiple serves to store, retrieve and process data faster and at a larger scale, all while enabling redundancy and scalability. This is common when you want built-in redundancy and scalability for large amounts of data.

Data engineers should be aware of the consistency paradigms of distributed systems.

Eventual versus Strong consistency

Consistency

Data consistency is the concept of data in a system reflecting the true state of the objects or entities it represents. Meaning that the data is valid, and up-to-date based on previous changes of the state.

⚠️ In distributed systems, where data is partitioned or replicated a challenge arises with consistency as servers storing the data are located in different geographic areas and rely on networking to communicate. For example, if we have a database representing products, and someone buys one, a highly consistent system would reflect the decrease in the inventory of that product and prevent another person from buying it. While a less consistent system may not reflect that change across every node immediately.

Consistency patterns (ACID)

- Atomic: Several transactions are treated as a single transaction. The transaction will fail if each sub transaction does not succeed.
- Consistency: Ensures that the database remains consistent.
- Isolation: Each transaction is isolated from one another. Each transaction is executed as if it was the only transaction being made at the time.
- Durability: Ensures that a transaction once completed will survive power outages or system shutdowns.

Consistency patterns (BASE)

💡 This concept is the opposite of ACID and is the basis of eventual consistency.

- Basically Available: Consistency is not guaranteed, but database reads and writes are made on a best-effort basis, meaning consistent data is available most of the time.
- Soft-state: The state of the transaction is fuzzy and it's uncertain whether the transaction is committed or uncommitted.
- Eventual Consistency: At some point, reading data will return consistent values.

Eventual consistency

✏️ **Eventual consistency** is a consistency paradigm which prioritizes availability and partition tolerance over strict consistency. The data is allowed to be temporarily inconsistent across nodes over time, and is eventually brought into a consistent state.

✏️ **Partition tolerance** is the ability of a distributed system to continue operating in the case of a network outage or failure that prevent communication between nodes. For example, if a distributed global system has two regions which can no longer communicate, then both regions can act independently until communication is restored.

💡 Eventual consistency is a common trade-off in large-scale, distributed systems. If you want to scale horizontally to process data in high volumes, then eventually, consistency is often the price you'll pay. This paradigm allows you to retrieve data quickly without verifying that you have the latest version across all nodes.

Strong consistency

💡 ACID transactions ensure strong consistency, and ensures that in a distributed database writes to any node are first distributed with a consensus and that any reads against the database return consistent values. This comes with the drawback of higher latency but will be used when you require correct data every time you read from the database.

Consistency decisions

There are three places where DEs make decisions about consistency:

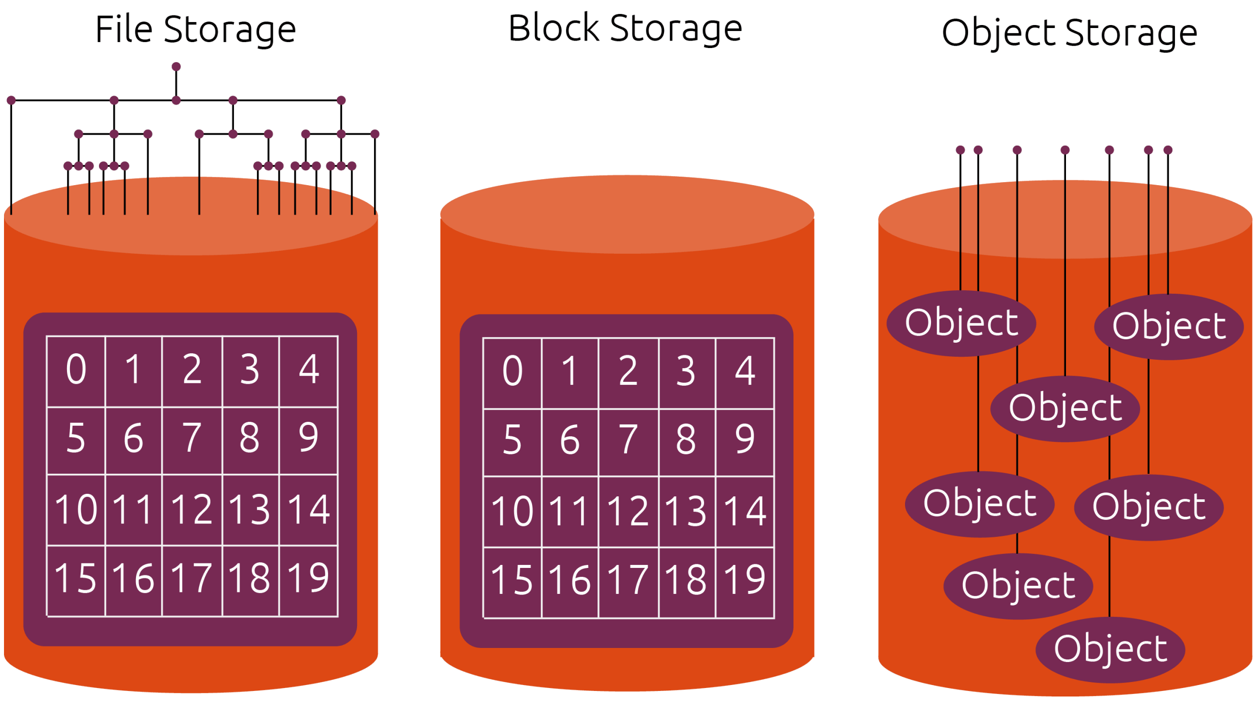
1. The database technology itself
2. Configuration parameters for the database

3. Individual query level: Some databases support configuration support at this level of granularity

🧠 Data engineers should understand how a database handles consistency.

Storage formatting

💡 Storage formatting is a way of storing data on disk in a way that allows us to easily access it. There are three types: file storage, block storage and object storage.



File storage

✏️ A file is a data entity with specific read, write and reference characteristics used by software and OS. They have the follow characteristics:

1. Finite length: A file is a finite-length stream of bytes
2. Append operations: We can append bytes to the file up to the limit of the host system storage
3. Random access: We can read from any location in the file or write updates to any location

💡 Object storage acts much like file storage, but with key differences.

💡 File storage systems organize files into a directory tree: /Users/alexg/output.txt. The file stores each directory as metadata about the files and directories that it contains. This metadata consists of the name of each entity, relevant permission details and a pointer to the actual entity. To find files on disk, the OS looks at the metadata at each hierarchy level and follows the pointer to the next sub-directory until it is eventually found.

Local disk storage

💡 The most familiar type of file storage is an OS managed file system on a local disk partition. Local file systems generally support full read after write consistency; reading immediately after a write will return the written data (You will always get back the up to date file you just wrote, full consistency). OS also employs locking strategies to manage concurrent writing attempts to a file.

Network attached storage NAS

✏️ NAS systems provide a file storage system to clients over a network, enabling users to access a centralized and shared storage space for their files (Dropbox). NAS systems make use of NAS devices which are servers that come in with built-in dedicated NAS interface hardware.

💡 While there are performance penalties to accessing the file system over a network, significant advantages to storage virtualization also exist, including redundancy, reliability, fine-grained control of resources, storage pooling across multiple disks for large virtual volumes and file sharing across multiple machines.

Cloud file system services

💡 Cloud file system services provide a fully managed file system for use with multiple cloud VMs and applications. Cloud file systems should not be confused with standard storage attached to VMs. Generally, cloud systems behave much like NAS solutions but abstracts away the complexity of networking, managing disk clusters, failures and configurations. An example would be the Amazon Elastic File system (EFS). EFS is highly consistent and offers read-after-write consistency.

Block storage

💡 Fundamentally, block storage is the type of raw storage provided by SSDs and HDDs. In the cloud, virtualized block storage is the standard for VMs. These block storage abstractions allow fine control of storage size, scalability and data durability.

✏️ A block is the smallest addressable unit of data supported by a disk. This used to be 512 bytes of usable data but is now closer to 4,096. Blocks typically contain extra bits for error detection/correction and other metadata.

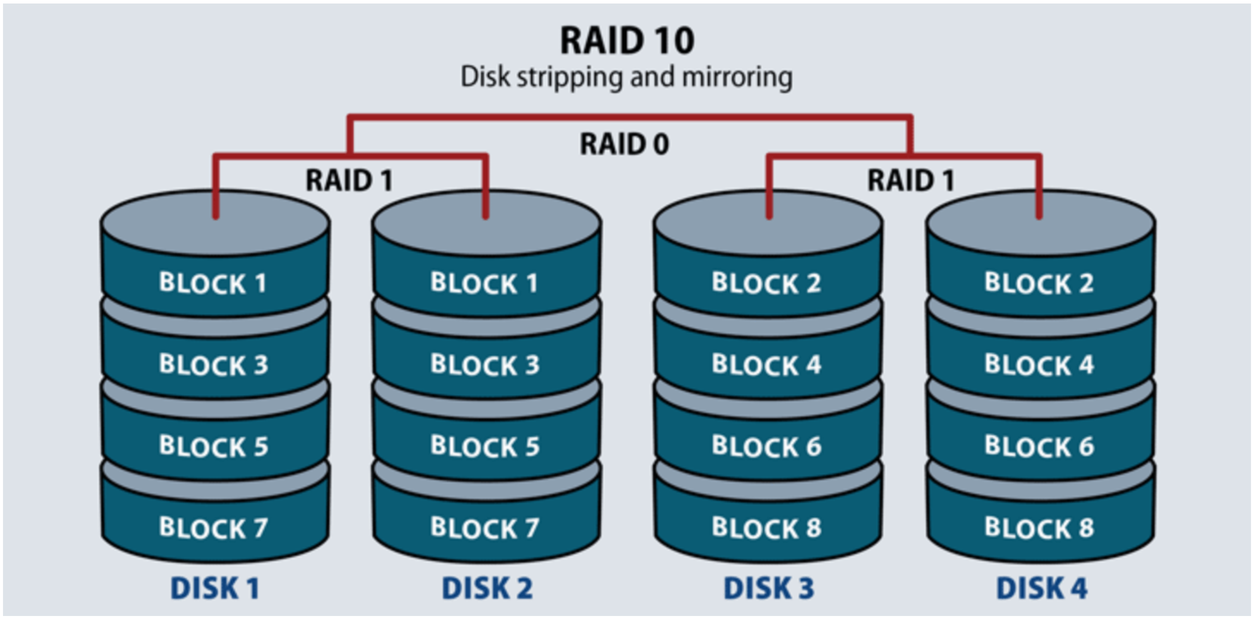
💡 Blocks on magnetic disks are geometrically arranged on a physical platter. Two blocks on the same track can be read without moving the head, while reading two blocks on separate tracks requires a seek.

⚙️ Blocks are logical units as opposed to physical when they are stored on disk. There is the concept of sectors in HDD and pages in SSD which comprise of the smallest physical unit of data making up a certain amount of bytes, but blocks can make up several sectors or pages.

Block storage applications

💡 Transactional databases generally access disks at a block level to lay out data for optimal performance. Block storage also remains the default option for OS boot disks on cloud VMs.

RAID

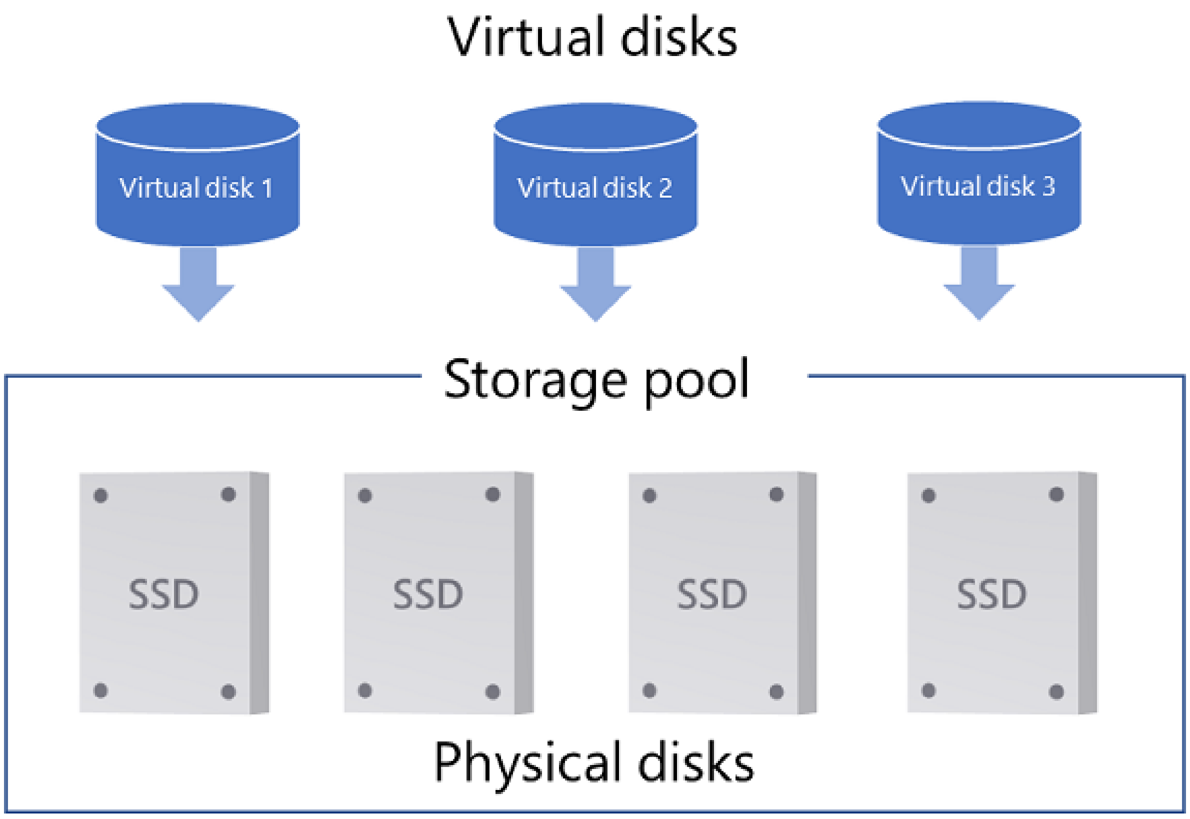


✏️ **Redundant arrays of independent disks (RAID)** parallelize on a single server. It involves using several magnetic disks to work together as a single logical disk that can act a single disk to the computer. It can increase redundancy and increase performance.

💡 While storage solutions such as RAID act on a single server, cloud object storage acts on a much larger scale often with disks distributed across a network and even multiple data centers and availability zones.

Storage access network SAN

💡 SAN systems provide virtualized block storage devices over a network, typically from a storage pool. SAN abstraction allows fine-grained storage scaling and enhance performance, availability and durability. A SAN system knows the location of each block of data, within the different storage pools they can access and manage. SAN maintains a mapping between the logical addresses used by servers and applications to access the data, and the physical addresses used by the storage devices to store the data.



💡 SANs allow servers to connect to storage pools or hard drives as if they were directly attached to the server. Block storage is the lowest level of storage on these SSD and HDDs and RAID is a technique used to pool storage devices together to act as a single logical unit. These enable servers to access highly redundant and reliable storage systems.

Cloud virtualized block storage

💡 Cloud virtualized block storage are similar to SAN but free engineers from dealing with SAN clusters and networking details. Elastic Block Store (EBS) is a good example. EBS store data separate from the instance host server but in the same zone to support high performance and low latency. This allows EBS to persist when an EC2 instance shuts down, when it fails or is deleted. EBS is suitable for use cases such as databases where data persistence is important.

Metrics

EBS performance metrics include:

- IOPS
- Throughput

Replication and Snapshots

💡 EBS replicates all data to at least two separate host machines, protecting data if a disk fails. EBS volumes also allow point-in-time snapshots while the drive is used.

Local instance volumes

Cloud providers offer block storage volumes that are physically attached to the host server running a VM. These storage volumes are generally very low cost and included with the price of the VM. Instance volumes behave essentially like a disk physically attached to a server in a data center. A difference is that when the VM is deleted or shuts down the contents of the locally attached disk are lost. Local instances support none of the virtualization benefits offered by EBS.

Usefulness

Regardless locally attached disks are extremely useful, and are used as a local cache and don't need all the advanced virtualization features of a service like EBS. It is useful for ephemeral jobs such as consuming data from S3, processing it locally and storing it back in S3. It is recommended to think about local instance volumes in worst case scenarios: what happens during a local disk failure, a VM shutdown etc… if these scenarios are not catastrophic then local storage is a good solution.

Object storage

Object storage contains objects of all shapes and sizes. While object has several meanings in computer science, in this context we are referring to a specialized file-like construct. It could be any type of file - TXT, CSV, JSON, images, videos or audio. Object storage has grown in importance as S3 and GCS are widely used.

Many dara warehouse providers use object storage as their storage layer, and cloud data lakes generally sit on object stores.

Key value storage

💡 An object store is a key-value store for immutable data objects. Unlike file storage, objects don't allow append operations or random writes, instead they are written as a stream of bytes. Once written the data is immutable.

Stream of bytes

✏️ A stream of bytes refers to a sequence of data bytes that are being transferred and processed sequentially. The term stream does not imply that it is limited or limitless, but rather that the data is being processed one bit at a time as opposed to all at once. The distinction is made above to show how the data is handled, objects are written once as a stream of bytes and then cannot be changed.

Serving parallel distributed query engines

💡 Object stores can support extremely performant parallel stream writes and reads across many disks, and this parallelism is hidden from engineers. Read bandwidth can scale with the number of parallel requests, the number of virtual machines employed to read data and the number of CPU cores. These characteristics make object storage ideal for serving high volume web traffic or delivering data to highly parallel distributed query engines.

Parallelism

✏️ Parallelism refers to simultaneous execution of multiple data read and write operations. Instead of processing data sequentially (one operation at a time), multiple operations can be carried out concurrently. This is achieved as the data of the object store is distributed on several disks in which the disks can be accessed simultaneously by multiple read and write operations, making use of their combined resources.

Why object data can handle parallelism

1. Flat address space: Object stores have a unique identifier across the entire network, which does not rely on a hierarchical structure of folder and files. This simplifies access as it does not force you to navigate through folders and files to find the object, which means you can locate it directly.

    💡 File storage systems organize files into a directory tree: /Users/alexg/output.txt. The file stores each directory as metadata about the files and directories that it contains. This metadata consists of the name of each entity, relevant permission details and a pointer to the actual entity. To find files on disk, the OS looks at the metadata at each hierarchy level and follows the pointer to the next sub-directory until it is eventually found. While object storage may have a similar syntax as /bucket/file_name this is a naming convention and does not rely on the searching characteristics of file storage

2. Stateless design: Since object storage is immutable there is no need to maintain information about the clients accessing it, or locking and coordination to prevent mismatches or conflicts between data as it is accessed
3. Data partitioning and distribution: Object stores can be automatically partitioned and distributed across several disks allowing for the data to be accessed by parallel read and writes across all disks. This allows for horizontal scalability.

💡 The design of object storage enables high availability across several zones and high durability as it can persist among failures.

Separation of compute and storage

💡 Cloud object storage is a key ingredient in separating compute and storage, allowing engineers to process data with ephemeral clusters and scale these clusters up and down on demand. This enables an almost limitless amount of storage as data is only limited by the amount of disks, which in cloud environments is practically limitless.

- [ ] Why is the separation of compute and storage important

Object stores for data engineering applications

💡 Object storage provide excellent performance for large batch reads and batch writes. This corresponds well to the use case for massive OLAP systems. Object stores do not handle update operations as well as transactional databases, however. Object storage is an ideal repository for unstructured data in any format beyond these structured data applications. It can house any binary data with no constraints on type or structure and plays a role in ML pipelines for raw text, images, videos and audio.

Object lookup

💡 Due to the store being a key-value store, the key of the object must be unique across the entire namespace of the cloud provider. As mentioned, the semantic of the id: s3://alexgirardet/hello.txt is only the semantic of the id and does not represent true hierarchy. This is important because it means to run lookups based on prefixes or 'directory' level operations can be quite costly.

Object consistency and versioning

💡 As a rule object storage doesn't support in place update or appends. To update an object we replace the object with the same key. We must be aware of the consistency model these stores use. S3 was eventually consistent until recently, meaning if you update the object the key may sometimes return the old object. After enough time, the database will eventually be consistent across all disks.

Imposing consistency

💡 We can improve consistency by leveraging the metadata of the object store and using a strongly consistent database such as postgres.

1. Write the object
2. Write the returned metadata for the object version to the strongly consistent database
3. Uniquely identify the new objects using the id and the metadata by fetching the metadata from the strongly consistent db
    a. Query object metadata using the object query, and check if it's metadata matches the metadata from the strongly consistent db
    b. If it doesnt repeat the step until it does

💡 When we rewrite an object we are essentially writing a brand new object, setting references from the existing key to the object, and deleting the old object references. Updating all references across all clusters take time, hence the potential for stale reads. Eventually the storage cluster garbage collector deallocates the space dedicated to the dereferenced data, recycling disk capacity for use by new objects.

✏️ A stale read is when data is read from an object and it returns data that is no longer current or accurate.

Versioning

💡 With object versioning turned on, we add additional metadata to the object that stipulates a version. This also prevents the garbage cleaning process, as old references are listed and stored to point to previous objects. This means that when we reference an object with a version the consistency issue disappears. We must however, consider the cost of storage as old versions are not deleted. Engineers can deploy lifecycle policies that delete data based on their version number or age.

Storage classes and tiers

💡 Cloud vendors offer storage classes that discount storage pricing in exchange for reduced access or reduced durability. Storage is cheaper but retrieval is more expensive. Think of archive storage.

Object store-backed filesystems

💡 Object store synchronization solutions have become increasingly popular. Tools like s3fs and Amazon S3 File gateway allow users to mount an S3 bucket as local storage. Mounting object storage as a local file system works well for files that are updated infrequently. This is due to latency constraints and the lower consistency model of object storage relative to local file systems.

Cache and Memory-Based Storage Systems

💡 RAM as mentioned before offers excellent latency and transfer speeds, but is extremely vulnerable to data loss as it is volatile storage that relies on charges. This makes it useful however, for caching applications which need to present data for quick access and high bandwidth, which do not need persistent storage.

Memcached and lightweight object caching

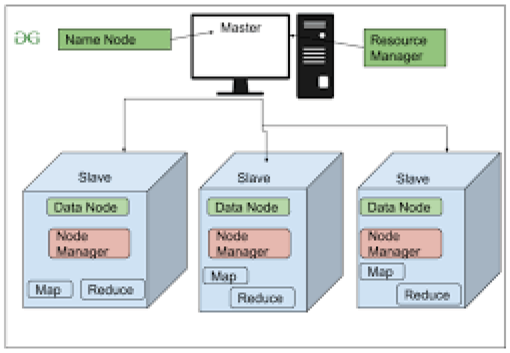
💡 Memcache is a key-value storage designed for caching database query results, API call responses and more. Memcached uses simple data structures, supporting either string or int types. It can deliver low latency results and reduce load on back-end systems.

Redis

💡 Redis is a key-value store but supports more complex data types such as lists or sets. It also builds in multiple persistence mechanisms, including snapshotting and journaling. It writes data every two seconds as default and is suitable for extremely high-performance applications that can tolerate a small amount of data loss.

The Hadoop Distributed file system

💡 Hadoop is based on the Google file system and was engineered to process data with the MapReduce programming model. It is similar to object storage but with a key difference, compute and storage are on the same node where object stores typically have limited support for internal processing. Hadoop still appears in legacy systems as the MapReduce modle has fallen by the wayside. Spark clusters still run on Hadoop sometimes.



⚙️ Hadoop breaks large files into blocks, chunks of data less than a few hundred megabytes. The system is managed by the NameNode, which maintains directories, file metadata and a detailed catalog describing the location of file blocks in the cluster. It has built in persistence as it distributes data across three nodes typically, and with it's built in support for processing using the MapReduce programming model it is powerful.

- [ ] Difference between MapReduce and Spark programming models

Streaming storage

💡 Streaming data has different storage requirements to non storage data. In the case of message queues, stored data is temporal and exepected to disappear after a certain duration. Kafka supports indefinite data retention by pushing old, infrequently accessed messages down to object storage.

Replay

💡 Replays allow a streaming system to return a range of historical stored data. Replay is the standard data-retrieval mechanism for streaming storage systems. It can be used to run batch queries over a time range.

Indexes, Partinioning and Clustering

💡 Indexes provide a map of the table for particular fiels and allow extremely fast lookup of individual records. Without indexes entire scans of the table would be necessary to find data. In RDBMSs indexes are used for primary table keys, and foreign keys.

💡 We are witnessing an evolution away from indexes in analytics-oriented storage systems and some new developments for analytic use cases.

Evolution from rows to columns

💡 In the early days warehouses were typically build on RDBMS. Massively parallel processing (MPP) systems lead to a shift towards parallel processing for significant improvements in scan performance across large databases for analytic purposes.

Columnar serialization

💡 This enables scans on the database on only the required data for a particular query, reducing the amount of data read from disk. Columnar databases performed poorly on joins, which lead to engineers denormalizing the data, using wide schemas, arrays and nested data whenever possible but join performance has dramatically improved making this no longer a requirement. Joins requre finding and matching data from multiple rows which is inconsistent with the columnar format.

From indexes to partitions and clustering

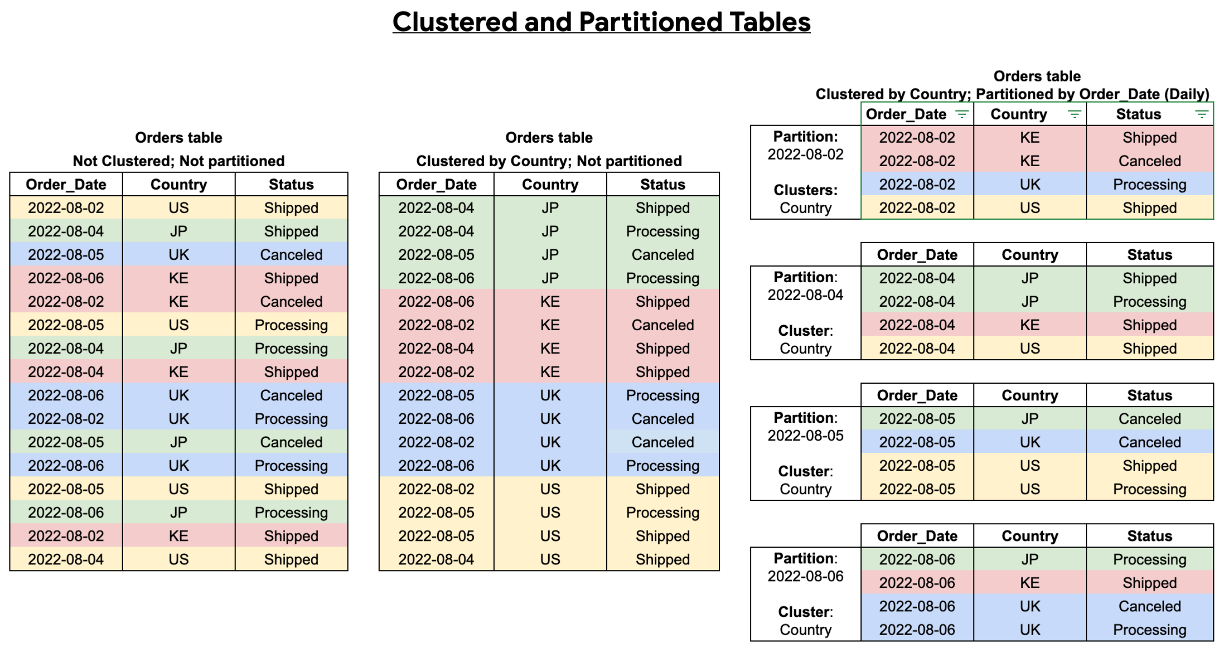
💡 Clusters and partitions allow for faster scan speeds as less data is needed to be scanned.

Clustered data

✏️ Clustering data refers to physically arranging similar data on disk in contiguous blocks, which can improve query speeds.

Partitioned data

✏️ Partitioning refers to the logical division of tables into smaller tables based on partitions.



Clustered and Partitioned Tables

Orders table
Not Clustered; Not partitioned

Order_Date	Country	Status
2022-08-02	US	Shipped
2022-08-04	JP	Shipped
2022-08-05	UK	Canceled
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-05	US	Processing
2022-08-04	JP	Processing
2022-08-04	KE	Shipped
2022-08-06	UK	Canceled
2022-08-02	UK	Processing
2022-08-05	JP	Canceled
2022-08-06	UK	Processing
2022-08-05	US	Shipped
2022-08-06	JP	Processing
2022-08-02	KE	Shipped
2022-08-04	US	Shipped

Orders table
Clustered by Country; Not partitioned

Order_Date	Country	Status
2022-08-04	JP	Shipped
2022-08-04	JP	Processing
2022-08-05	JP	Canceled
2022-08-06	JP	Processing
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-04	KE	Shipped
2022-08-02	KE	Shipped
2022-08-05	UK	Processing
2022-08-06	UK	Canceled
2022-08-02	UK	Canceled
2022-08-06	UK	Processing
2022-08-02	US	Shipped
2022-08-05	US	Processing
2022-08-05	US	Shipped
2022-08-04	US	Shipped

Orders table
Clustered by Country; Partitioned by Order_Date (Daily)

	Order_Date	Country	Status
Partition: 2022-08-02	2022-08-02	KE	Shipped
	2022-08-02	KE	Canceled
Clusters: Country	2022-08-02	UK	Processing
	2022-08-02	US	Shipped

	Order_Date	Country	Status
Partition: 2022-08-04	2022-08-04	JP	Shipped
	2022-08-04	JP	Processing
Cluster: Country	2022-08-04	KE	Shipped
	2022-08-04	US	Shipped

	Order_Date	Country	Status
Partition: 2022-08-05	2022-08-05	JP	Canceled
	2022-08-05	UK	Canceled
Cluster: Country	2022-08-05	US	Shipped
	2022-08-05	US	Processing

	Order_Date	Country	Status
Partition: 2022-08-06	2022-08-06	JP	Processing
	2022-08-06	KE	Shipped
Cluster: Country	2022-08-06	UK	Canceled
	2022-08-06	UK	Processing

Snowflake micro-partitioning

💡 Snowflake clusters together similar rows. This constrats the traditional approach to partitioning on a single designated field. This allows for aggressive pruning of queries based on predicates. The snowflake metadata database, stores a description of each micro-partition including the number of rows and value ranges for fields. When a query is made snowflake analyzes micro-partitions to determine which ones need to be scanned.