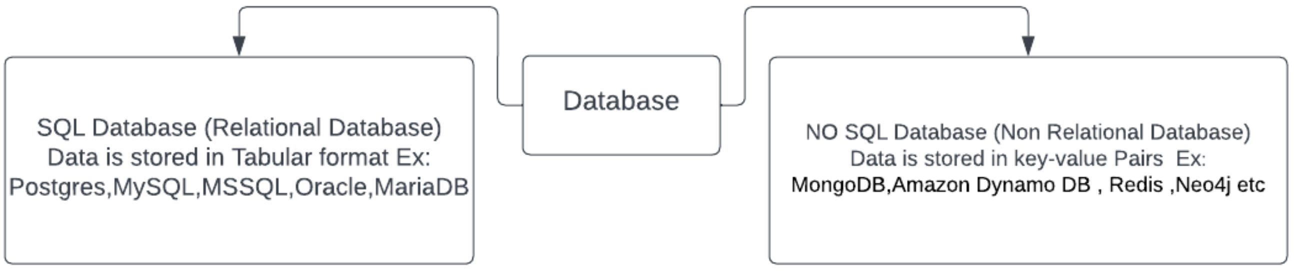
Database

In System Design Interview How good the design is and how well it can scale depends very mush on the choice of database
There are mainly two types of Databases SQL Database and No-SQL Database
Youtube:1) Link 2) Link 3) Link 4)LInk 5) Link



Database
SQL Database (Relational Database) Data is stored in Tabular format Ex: Postgres,MySQL,MSSQL,Oracle,MariaDB
NO SQL Database (Non Relational Database) Data is stored in key-value Pairs Ex: MongoDB,Amazon Dynamo DB , Redis ,Neo4j etc

Centralized Database: A centralized database is a type of database architecture in which all data is stored in a single location, and all access to the data is controlled by a central server. In this architecture, the database server is responsible for managing and controlling access to the data, and clients access the data through the server.

Distributed Database: A distributed database is a type of database architecture in which data is distributed across multiple nodes or servers in a network. In a distributed database, each node stores a portion of the data, and the nodes work together to provide a unified view of the data to users and applications. It is used in Ecommerce, Social networks , Banking , Healthcare

Cloud Database: A cloud database is a type of database that is hosted on a cloud computing platform, such as Amazon Web Services, Microsoft Azure, or Google Cloud Platform. In a cloud database, data is stored in the cloud rather than on a local server, and users can access the data from anywhere with an internet connection.

Object Oriented Database: An object-oriented database is a type of database that is based on the principles of object-oriented programming. In an object-oriented database, data is represented as objects, which are instances of classes that contain data and methods for manipulating that data.

Hierarchical Database: A hierarchical database is a type of database that organizes data into a tree-like structure, where each record has a parent-child relationship with other records in the database. In a hierarchical database, the data is organized into levels, with each level representing a different type of record.

SQL Database:

- SQL database is used in case of Structured Data
- SQL databases are optimized for transactions and provide ACID (Atomicity, Consistency, Isolation, and Durability) compliance
- SQL databases provide powerful querying capabilities that allow for ad-hoc queries and complex joins. This makes them a good choice for applications that require advanced data analytics and reporting.
- Scalability: SQL databases are easy to scale vertically by adding more powerful hardware, making them a good choice for applications that require high performance and scalability.

When not to use SQL databases:

- Unstructured data: If your application deals with large volumes of unstructured data.

- High write throughput: SQL databases can be slower when it comes to high write throughput or real-time data processing or high concurrency.
- Horizontal scalability: SQL databases can be more difficult to scale horizontally by adding more nodes to a cluster. NoSQL databases may be more suitable for applications that require horizontal scalability and fault tolerance.
- Cost: SQL databases can be expensive to license and require significant hardware resources. NoSQL databases may be a more cost-effective option for applications with large amounts of data.

Ex:MySQL,Postgres,MSSQL,Oracle,MariaDB,CockroachDB
Relational Databases are the most popular ones and two factors which helps us to decide weather we should select relational database or not they are : **Schema and Acid Properties**
Schema: How our data is Going to be structured,so , if our data can be represented in the form of tables and rows while satisfying properties of relational DB then we should select relational DB, rdbs comes with schema constraints which tell some field can't be null so it make sure some garbage data can't come to the database

ACID(Atomicity Consistency Isolation Durability): LInk

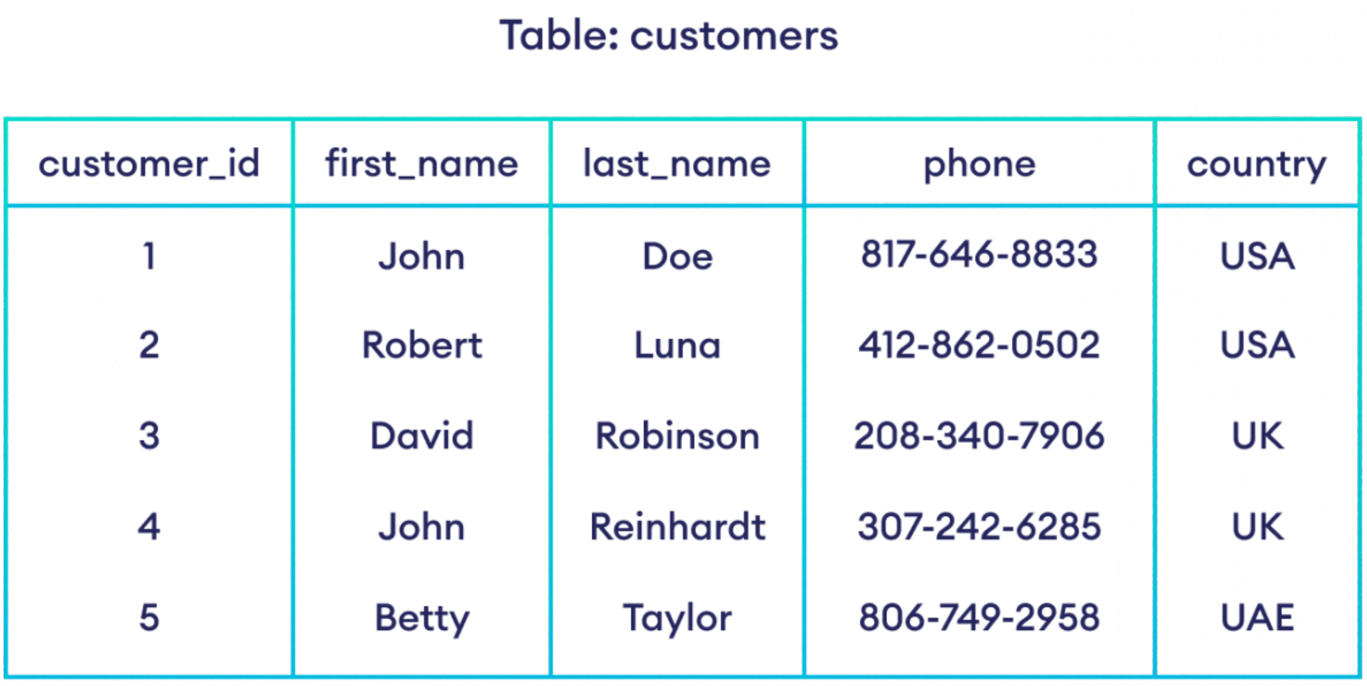


Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Taylor	806-749-2958	UAE

NoSQL Database:

NoSQL databases are becoming increasingly popular due to their ability to handle large, complex, and unstructured data.
When to use NoSQL databases:

- Large amounts of unstructured data
- Scalability: NoSQL databases are highly scalable and can easily accommodate increasing data.
- High availability: provide high availability and fault tolerance, making them a good choice for applications that require continuous availability.
- Flexibility: NoSQL databases are schemaless, which means they can accommodate changes to data structure without requiring changes to the database schema.

When not to use NoSQL databases:

- Transactions
- Complex data relationships: NoSQL databases do not support complex data relationships as well as traditional relational databases, so they may not be the best choice for applications with complex data models.

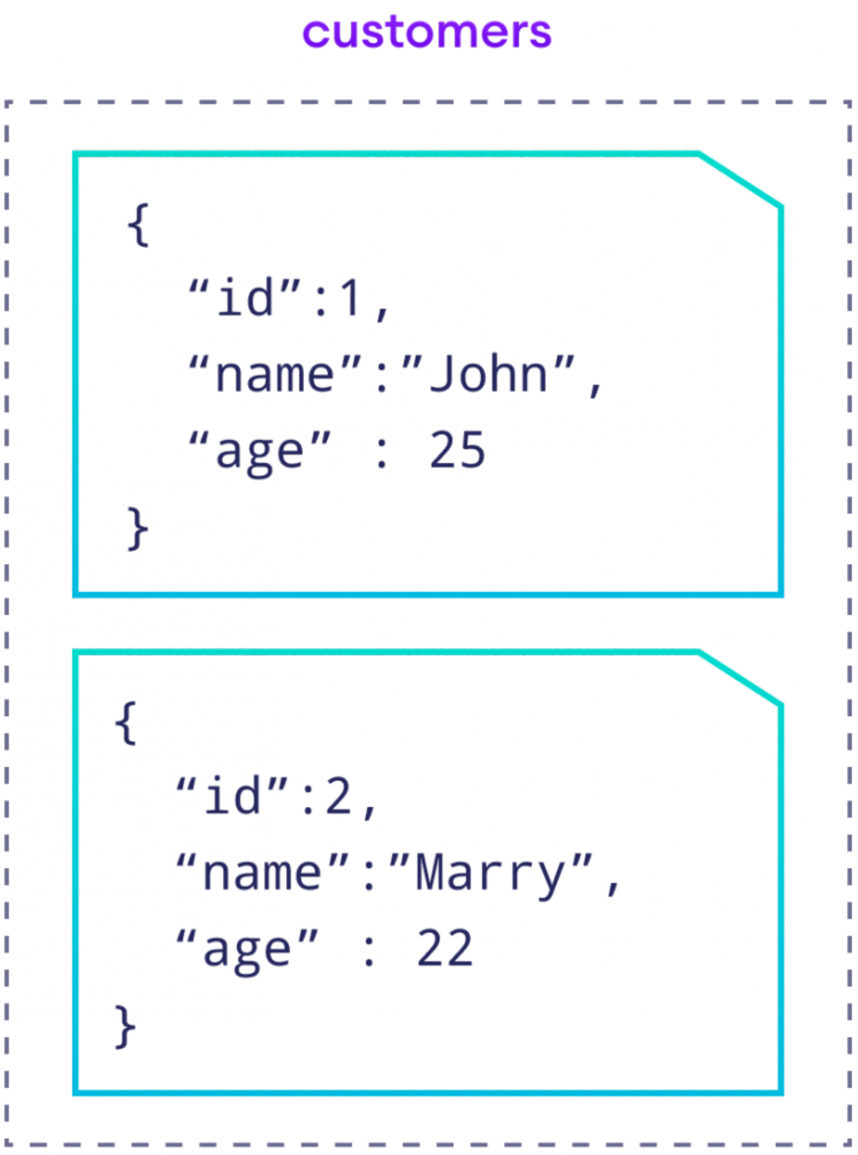
- Consistency: NoSQL databases do not provide the same level of data consistency as traditional relational databases
- Analytics and reporting: NoSQL databases are not always the best choice for analytics and reporting, particularly if the data is highly structured and requires complex queries.

In summary, NoSQL databases are a good choice for applications that deal with large amounts of unstructured data, require high scalability and availability, and are flexible enough to accommodate changes to the data structure. However, they may not be the best choice for applications that require strict data consistency, complex data relationships, transactions, or advanced analytics and reporting.

Types of NoSQL Database

1. Document Databases:These databases store data in a document format, such as JSON or BSON. Each document can have its own unique structure, allowing for more flexibility in data modeling. well-suited for applications that require high-speed access to large volumes of data They are also highly scalable,provide sharding capabilities allowing them to easily accommodate increases in data volume and traffic.Link Popular document databases include MongoDB, Couchbase, and RavenDB,firestore.
2. Time series Databases: Time series databases are designed to handle large volumes of time series data such as sensor data, stock market data, or website traffic data, with efficient storage and querying of data points based on their timestamps. They often provide specialized functions for analyzing and processing time series or metrics kind(application like grafana,prometheus which is basically an application tracking system) data, such as aggregation, interpolation, and anomaly detection.Time series databases typically use a columnar or compressed storage format to optimize storage and retrieval of time-stamped data.Popular time series databases include InfluxDB, TimescaleDB, and OpenTSDB. They are often used in combination with other types of databases, such as graph databases and document databases, to create a complete data management solution.
3. Column Databases:These databases store data in columns instead of rows, which allows for high scalability and fast querying of large data sets. Column-family stores are commonly used for big data applications and data warehousing.Useful for handling large volumes of structured data that require fast read and write performance. They are designed to handle horizontal scaling, and can support high levels of concurrency.Link Popular column-family stores include Apache Cassandra, HBase, and Google Bigtable.
4. Realtime Databases:Real-time databases are optimized for fast read and write performance, and are typically used in applications where data needs to be processed and analyzed in real-time, such as in monitoring and control systems, financial trading platforms, and online gaming platforms,chat applications and messaging platforms.Real-time databases support features such as low latency, high availability, and data synchronization across multiple devices and locations.They often use specialized data structures and algorithms to optimize data processing and analysis, and support features such as event-driven triggers and real-time data streaming.Popular real-time databases include Firebase Realtime Database, Apache Kafka, and Redis. They are often used in combination with other types of databases, such as document databases and key-value databases.
5. Key-value Databases:These databases store data as key-value pairs, where each value is associated with a unique key. Key-value stores are known for their simplicity and high performance.Highly scalable and performant, making them well-suited for applications that require high-speed access to large volumes of data. They are often used as a caching layer for web applications, where they can store frequently accessed data in memory for faster retrieval. Key-value databases can also be used as a persistent data store, where they can handle high write throughput and support large datasets. Popular key-value stores include Redis, Amazon DynamoDB, and Riak.They are often used in combination with other types of NoSQL databases, such as document databases and column-family stores, to create a complete data management solution.

6. Graph Databases:These databases store data as nodes and edges in a graph, allowing for easy querying of complex relationships between data points. Graph databases are often used for social networks, recommendation engines, and fraud detection. Useful for handling complex, highly interconnected data, such as social networks, recommendation engines, and network topology data. They are designed to support graph queries, which allow users to traverse and analyze the relationships between entities in the graph.They are designed to handle horizontal scaling.Popular graph databases include Neo4j, OrientDB, and ArangoDB.
7. Blob Storage: column-family stores or key-value stores are often used to store binary large objects (BLOBs) due to their ability to efficiently store and retrieve large amounts of unstructured data using CDN.One common approach is to use a key-value store such as Amazon S3 ,Google Cloud Storage, Microsoft Azure Blob Storage,DigitalOcean Spaces,Backblaze B2 Cloud Storage to store the binary data, with the key serving as a unique identifier for the data. The metadata associated with the binary data, such as file name, size, and format, can be stored in a document-based database such as MongoDB or Couchbase, with the key serving as a reference to the corresponding binary data.
8. Search Database engines: Elastic search , solar . But this are not database
9. Data Warehouse : use mainly for analytics purpose in data science and analytics Ex: Hadoop,Apache spark



```
{
  "id":1,
  "name":"John",
  "age" : 25
}
```

```
{
  "id":2,
  "name":"Marry",
  "age" : 22
}
```

Some factors to consider when deciding between SQL and NoSQL databases include:

- Data structure: SQL databases are designed for structured data with predefined schemas, while NoSQL databases are designed for unstructured or semi-structured data,No fixed Schema.
- Data access patterns: SQL databases are well-suited for complex queries and support for joins, while NoSQL databases are designed for high throughput and low latency, and often use simple key-value access patterns.
- Scalability: NoSQL databases are designed for horizontal scalability, with the ability to distribute data across multiple nodes, while SQL databases are often more limited in their scalability options.
- Performance: NoSQL databases are often faster and more efficient than SQL databases for certain types of applications and data access patterns.
- Consistency: SQL databases are designed for strong consistency and transactional integrity, while NoSQL databases often sacrifice consistency for scalability and performance.
- Data integrity: SQL databases have built-in support for enforcing data integrity constraints, such as primary key and foreign key constraints, while NoSQL databases often rely on application-level logic for data validation and consistency.
- Data storage: SQL databases store data in tables with fixed columns and data types, while NoSQL databases can store data in a variety of formats, including key-value pairs, documents, and graphs.

Horizontal Scaling vs Vertical Scaling

Vertical Scaling(SQL Database): Vertical scaling, also known as scaling up, involves adding more resources (such as CPU, RAM, and storage) to a single server to increase its capacity. For example, you can add more memory to a server or upgrade its CPU to improve its performance. This approach is often used with traditional relational databases, which are designed to run on a single server.

vertical scaling is easier to implement and can be done without modifying the application code. However, there is a limit to how much a single server can be scaled vertically, and it can become expensive to upgrade hardware beyond a certain point.

Horizontal Scaling(No SQL Database): Horizontal scaling, also known as scaling out, involves adding more servers to a database system to increase its capacity. This approach distributes the workload across multiple servers, allowing the system to handle a larger volume of data and more concurrent users. Horizontal scaling is often used with NoSQL databases and other distributed systems that can run on a cluster of servers.

Horizontal scaling provides unlimited scalability, as long as there are enough servers in the cluster. It can also improve fault tolerance, as the system can continue to operate even if some servers fail. However, horizontal scaling can be more complex to implement and may require changes to the application code to support distributed processing.

More About Databases

ORM(Object-Relational Mapping): Object-Relational Mapping (ORM) is a programming technique used to connect object-oriented programming languages with relational databases. The goal of ORM is to provide a way to interact with a database using objects and classes rather than writing raw SQL statements.It is a technique used to map object-oriented programming language concepts to a relational database.By mapping objects to database tables and vice versa, ORM frameworks provide a way to retrieve, store, and manipulate data from a database using object-oriented programming techniques.

Some popular ORM frameworks include Hibernate (Java), Entity Framework (.NET), and Django ORM (Python). ORM frameworks offer benefits such as increased productivity, reduced development time, and improved maintainability of code. However, it's important to note that using an

ORM may introduce performance overhead, as the framework must translate between the object-oriented and relational worlds.

ACID Property: ACID properties are a set of four properties that guarantee that database transactions are processed reliably. These properties are:

- Atomicity: This property ensures that a transaction is treated as a single unit of work, which means that either all of its operations are completed, or none of them are. If any part of the transaction fails, the entire transaction is rolled back, and the database returns to its previous state.
- Consistency: The data in the database should always be consistent before and after the transaction is executed.
- Isolation: This property ensures that concurrent transactions do not interfere with each other. Each transaction is executed in isolation, which means that the changes made by one transaction are not visible to other transactions until it is committed.
- Durability: This property ensures that once a transaction is committed, its changes are permanent and survive any subsequent system failures, power outages, or other similar events.

Together, these four properties ensure that database transactions are processed reliably and consistently, even in the face of system failures, power outages, or other similar events. The ACID properties are considered essential for transaction processing in a relational database management system (RDBMS).

Transaction: A transaction in a database is a logical unit of work that represents a series of database operations that must be executed together as a single, indivisible unit. A transaction can include one or more database operations such as insertions, updates, and deletions. The purpose of a transaction is to ensure the integrity and consistency of the data in the database.

In a database, transactions are used to ensure that data is not lost or corrupted due to system failures or other events. Transactions are executed in a way that guarantees the ACID properties, which ensure that transactions are processed reliably and consistently.
A typical transaction follows a series of steps, which include:

1. Begin: This step begins a transaction.
2. Perform database operations: This step includes one or more database operations that need to be executed as part of the transaction.
3. Commit: This step commits the transaction, which means that all of the database operations executed as part of the transaction are permanently saved in the database.
4. Rollback: This step rolls back the transaction, which means that all of the database operations executed as part of the transaction are undone, and the database is returned to its previous state.

Transactions can be initiated manually by the application, or they can be automatically initiated by the database management system. Transactions are a fundamental concept in database management systems and are used extensively in many different types of applications, including banking, finance, and e-commerce.

N+1 Query Problem: The N+1 query problem in a database is a performance issue that can occur when retrieving data from a database using an ORM (Object-Relational Mapping) framework. The problem arises when a query to retrieve a collection of objects results in an additional query being executed for each object in the collection.

For example, consider an application that needs to retrieve a list of customers and their orders from a database. If the ORM framework used to retrieve the data follows a lazy loading strategy, it may execute one query to retrieve the list of customers and then execute an additional query for each customer to retrieve their orders. This results in N+1 queries being executed (N queries to retrieve the customers and 1 query for each customer to retrieve their orders).

The N+1 query problem can lead to significant performance issues, particularly when dealing with large collections of data. The additional queries executed by the ORM framework can cause the application to suffer from slow response times, increased network traffic, and increased load on the database.To solve the N+1 query problem, the ORM framework can be configured to use eager loading, which retrieves all the required data in a single query. Alternatively, the application can use a more complex query that retrieves all the required data in a single query. Both of these solutions can significantly improve the performance of the application by reducing the number of queries executed against the database.

Database Normalization: Database normalization is the process of organizing the data [columns(attributes) and tables(relations)]in a relational database in a structured and efficient manner by reducing data redundancy and improving data integrity. The objective of normalization is to minimize data redundancy and prevent data inconsistencies, which can lead to data anomalies and inaccuracies.

Normalization involves breaking down large tables into smaller, more manageable tables and creating relationships between them. There are several levels of normalization, with each level building upon the previous level:

- First Normal Form (1NF): This level ensures that each table has a primary key, and each column in the table contains only atomic values (values that cannot be further divided).
- Second Normal Form (2NF): This level builds upon the previous level and ensures that each non-key column in the table is functionally dependent on the entire primary key, rather than on a subset of the key.
- Third Normal Form (3NF): This level builds upon the previous level and ensures that each non-key column in the table is dependent only on the primary key and not on other non-key columns in the same table.

There are additional levels of normalization beyond 3NF, including Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF), which are designed to further reduce data redundancy and improve data integrity.

Normalization is an important aspect of database design and is critical for ensuring that the data in a database is accurate, consistent, and easily maintainable. Properly normalized databases are also easier to scale, backup, and restore.

Failure Modes: There are several different failure modes that can occur in a database, including:

- Read contention: This occurs when multiple clients or processes are trying to read data from the same location in the database at the same time, which can lead to delays or errors.
- Write contention: This occurs when multiple clients or processes are trying to write data to the same location in the database at the same time, which can lead to delays or errors.
- Thundering herd: This occurs when a large number of clients or processes try to access the same resource simultaneously, which can lead to resource exhaustion and reduced performance.
- Cascade: This occurs when a failure in one part of the database system causes a chain reaction that leads to failures in other parts of the system.
- Deadlock: This occurs when two or more transactions are waiting for each other to release a lock on a resource, leading to a standstill.
- Corruption: This occurs when data in the database becomes corrupted, which can lead to errors or unexpected results when reading or writing to the database.
- Hardware failure: This occurs when hardware components, such as disk drives or memory, fail, which can lead to data loss or corruption.
- Software failure: This occurs when software components, such as the database management system or application, fail, which can lead to errors or unexpected results.
- Network failure: This occurs when the network connection between the database and the client is lost, which can lead to errors or timeouts when trying to access the database.
- Denial of service (DoS) attack: This occurs when a malicious actor attempts to overwhelm the database with requests, leading to resource exhaustion and reduced performance.

To mitigate the impact of failure modes in a database, it is important to implement backup and recovery procedures, redundancy and failover mechanisms, and to regularly test the database to ensure that it is functioning properly. Additionally, implementing appropriate security measures can help to prevent security breaches and protect the integrity of the data.

Profiling Performance: Profiling performance in a database refers to the process of analyzing and measuring the performance of a database to identify areas of inefficiency and to optimize its performance. Profiling performance is an important aspect of database management as it helps to ensure that the database is running at optimal levels and that users are getting the best possible experience.Profiling performance involves measuring various performance metrics, such as query response times, CPU and memory usage, disk I/O, network traffic, and other relevant parameters. This data is collected and analyzed to identify areas of the database that may be experiencing performance issues.
Some common performance profiling techniques used in databases include:

- Query profiling: This involves analyzing the performance of individual database queries to identify slow or inefficient queries and to optimize them for faster execution.
- Resource profiling: This involves analyzing the usage of system resources such as CPU, memory, disk I/O, and network traffic to identify areas of resource contention or overutilization.
- Index profiling: This involves analyzing the usage and effectiveness of database indexes to identify areas where additional or different indexes may be needed to improve performance.
- Workload profiling: This involves analyzing the types and frequency of database operations to identify patterns in the workload and to optimize the database to handle the workload more efficiently.
- Monitor system performance: You can use tools like the Windows Task Manager or the Unix/Linux top command to monitor the performance of your database server. These tools allow you to see the overall CPU, memory, and disk usage of the system, which can help identify any resource bottlenecks.
- Use database-specific tools: Most database management systems (DBMSs) have their own tools for monitoring performance. For example, Microsoft SQL Server has the SQL Server Management Studio (SSMS) and the sys.dm_os_wait_stats dynamic management view, while Oracle has the Oracle Enterprise Manager and the v$waitstat view. These tools allow you to see specific performance metrics, such as the amount of time spent waiting on locks or the number of physical reads and writes.
- Use third-party tools: There are also several third-party tools that can help you profile the performance of a database. Some examples include SolarWinds Database Performance Analyzer, Quest Software Foglight, and Redgate SQL Monitor. These tools often provide more in-depth performance analysis and can help you identify specific issues or bottlenecks.
- Analyze slow queries: If you have specific queries that are running slowly, you can use tools like EXPLAIN PLAN or SHOW PLAN in MySQL or SQL Server to see the execution plan for the query and identify any potential issues. You can also use tools like the MySQL slow query log or the SQL Server Profiler to capture slow queries and analyze them further.
- Monitor application performance: If you are experiencing performance issues with a specific application that is using the database, you can use tools like Application Insights or New Relic to monitor the performance of the application and identify any issues that may be related to the database.

Source

Scaling Databases

Database Indexes: A database index is a data structure that improves the speed and efficiency of data retrieval operations in a database. It is similar to an index in a book, which helps readers to quickly locate specific information without having to read through the entire book.

An index is created on one or more columns of a database table, and it contains a copy of the data from those columns along with a pointer to the location of the full record. When a query is executed that involves the indexed columns, the database can use the index to quickly find the relevant records, rather than scanning the entire table.

Indexes can significantly improve the performance of database queries, especially on large tables with many rows of data. However, indexes also have some downsides. Creating and maintaining indexes can require additional disk space and computational resources, and they can also slow down data modification operations like inserts, updates, and deletes.

There are several types of indexes that can be used in a database, including:

1. B-tree index: This is the most common type of index, used in most relational database management systems. It organizes data in a tree-like structure, with nodes that contain pointers to other nodes.
2. Hash index: This type of index is used for fast lookups on exact values, such as primary keys. It uses a hash function to compute a location for each key value, making lookups very fast.
3. Bitmap index: This type of index is used for low-cardinality columns, where the values repeat frequently. It stores a bitmap for each unique value in the column, making it efficient to search for records with a specific value.

Data Replication: Data replication is the process of creating and maintaining multiple copies of the same data in multiple locations. This can be done for a variety of reasons, such as improving performance, increasing availability, or ensuring data resilience in the event of a failure.
In a database environment, data replication involves copying data from one database to another, typically across different physical locations or servers data residing on a physical/virtual server(s) or cloud instance (primary instance) is continuously replicated or copied to a secondary server(s) or cloud instance (standby instance). There are several different replication models that can be used, depending on the specific requirements of the application and the underlying database management system. Some common replication models include:

- Master-slave replication: In this model, one database server is designated as the "master" and all updates are made to this server. The "slave" servers are then updated with a copy of the data from the master.
- Multi-master replication: In this model, multiple database servers are designated as "masters" and any updates made to one server are propagated to the other servers in the replication group.
- Peer-to-peer replication: In this model, each database server is both a master and a slave, and all servers are kept in sync with each other.

Data replication can be used to provide several benefits, including:

- Improved performance: By maintaining multiple copies of the data, queries can be distributed across multiple servers, reducing the load on any one server and improving overall query response times.
- Increased availability: By replicating data to multiple locations, the data can still be accessed even if one server or location goes offline.
- Data resilience: By maintaining multiple copies of the data, the system can recover more easily from failures or data corruption.

However, data replication also has some downsides, including increased complexity and cost. Replicating data across multiple locations requires additional network bandwidth and storage resources, and it can be more difficult to maintain consistency across all copies of the data. Additionally, data replication can introduce additional points of failure and security risks, especially if the data is replicated across public networks or to untrusted locations.
Video Link

Database Sharding: Database sharding is a technique used in distributed database systems to partition large data sets across multiple servers or nodes. Each shard contains a subset of the data, and queries are distributed across the shards based on the data being queried. This allows the database system to handle a larger data volume and higher query loads than a single server could handle.
Sharding involves breaking up the database into smaller, more manageable units called shards. Each shard is a self-contained subset of the data and can be stored on a separate physical server or node.

When a query is received, the database system routes the query to the appropriate shard based on the data being queried. The results are then aggregated and returned to the user.Sharding can offer several benefits, including: Scalability,Availability,Performance

However, sharding also has some downsides, including increased complexity, higher maintenance costs, and the potential for inconsistent data across shards.

LINK

Difference Between Scaling vs Sharding in Database

Scaling and sharding are both techniques used to handle the growth of a database system, but they differ in their approach and use cases. The main differences between scaling and sharding are:

- Approach: Scaling involves increasing the capacity of a single server or node in a database system, while sharding involves partitioning data across multiple servers or nodes in a distributed database system.
- Scope: Scaling typically applies to the entire database system, while sharding applies to specific data subsets or shards.
- Purpose: Scaling is typically used to handle increased data loads or user traffic, while sharding is used to handle large data volumes and to improve performance and scalability.
- Implementation: Scaling can be implemented through hardware upgrades or by adding more servers to the system, while sharding requires a more complex data partitioning strategy and query routing mechanisms.
- Trade Offs: Scaling is simpler to implement and maintain, but may reach a limit in terms of hardware capacity or cost-effectiveness. Sharding offers virtually unlimited scalability but requires a more complex infrastructure and can lead to data consistency issues.

In summary, scaling is a good option for small to medium-sized databases, while sharding is a more advanced technique suited for larger and more complex systems.

CAP Theorem: Link
Link