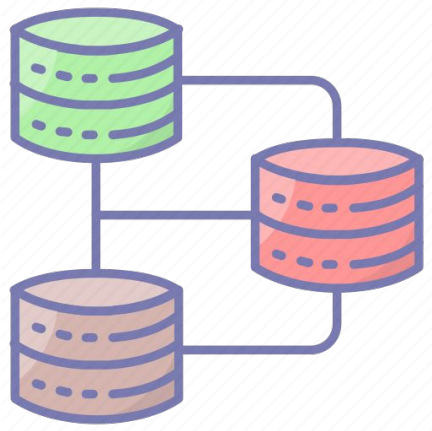
RDBMS and Scalability

Waheed Iqbal



DevOps (Fall 2023)
Department of Data Science, FCIT, University of the Punjab
Lahore, Pakistan

Relational Database

- A relational database stores data in tables consisting of rows and columns.
- In a relational database, data is organized into one or more tables, where each table represents a logical entity or concept (such as customers, orders, products, etc.) that the database is designed to manage.
- Each row in a table represents a single instance of the entity being modeled, and each column represents a specific attribute or characteristic of that entity.
    - For example, a customer table might have columns for the customer's name, address, phone number, and email address.
- The relationships between tables are established through the use of foreign keys, which allow data to be linked between tables.
    - For example, an order table might have a foreign key that links it to a customer table, indicating which customer placed the order.

Introduction (Cont.)

- Relational Database Management Systems (RDBMS) are commonly used to persist data in Web-based applications
- Public-facing e-Commerce sites/Web-based applications caused spikes
- RDMBS scalability is challenging as horizontal scaling with ACID guarantees is difficult to ensure

RDBMS ACID Properties

ACID is a set of properties in RDBMS that guarantee that database transactions are processed reliably.

- **Atomicity:** All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.
  - For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

- **Consistency**: Data is in a consistent state when a transaction starts and when it ends.
  - For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts would be consistent.

RDBMS ACID Properties (Cont.)

- **Isolation:** One transaction cannot read data from another transaction that is not yet completed.
  - For example, If two transactions are executing concurrently, each one will see the world as if they were executing sequentially, and if one needs to read data that is written by another, it will have to wait until the other is finished.

- **Durability:** Once a transaction is complete, it is guaranteed that all of the changes have been recorded to a durable medium (such as a hard disk), and the fact that the transaction has been completed is likewise recorded.
  - For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

Scalability Challenges in RDBMS

1. Performance limitations: As the size of the database and the number of concurrent users accessing it increases, performance can suffer due to limitations in disk I/O, network bandwidth, and processing power. These limitations can impact query response times.
2. Data consistency: In a distributed environment, maintaining data consistency across multiple nodes can be a challenge. Inconsistent data can lead to incorrect query results or data corruption.
3. Availability: Scaling a database often involves distributing data across multiple nodes, which can increase the likelihood of system failures. Ensuring high availability requires a robust architecture with failover mechanisms to prevent downtime.

Scalability Challenges in RDBMS (Cont.)

4. Complexity: Scaling a database requires a deep understanding of the database architecture and the ability to design and implement a scalable solution. This can be a complex task that requires significant expertise and resources.
5. Cost: Scaling a database can be expensive, both in terms of hardware and software costs. The cost of scaling a database can quickly become prohibitive, especially for smaller organizations.

Impact of Joins on Performance and Scalability

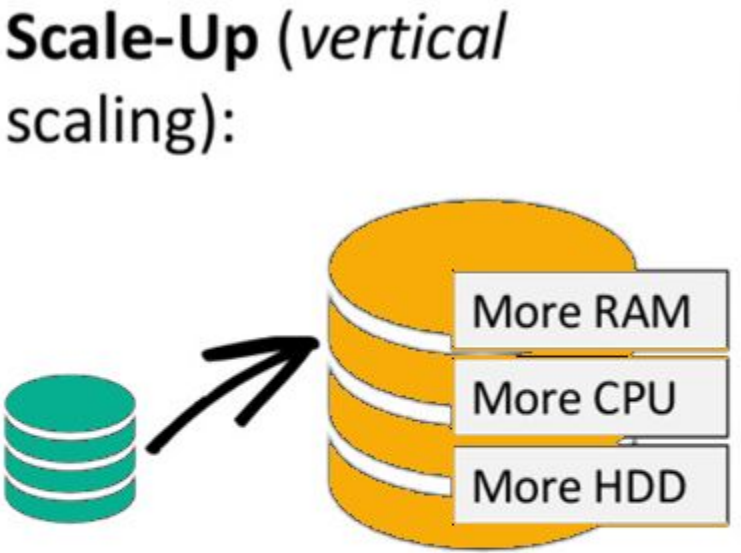
Joins in Relational Database Management Systems (RDBMS) can have a significant impact on query performance. Here are some ways joins can impact query performance in RDBMS

1. Table Size: The larger the tables being joined, the longer the query will take to execute. This is because the database engine must scan through all the rows in each table to find matching records. If the tables have a large number of rows, this can be a time-consuming process.
2. Join Type: The type of join used can also impact performance. A simple inner join is typically the most efficient, as it only returns rows that have a match in both tables. However, more complex join types, such as left joins or full outer joins, can be slower as they return more rows and require more processing.
3. Indexing: Indexes can significantly improve join performance by allowing the database engine to quickly find matching records. Indexes can be created on the join columns, as well as any other columns frequently used in the query.
4. Query Complexity: The complexity of the query can also impact join performance. Queries with many joins, subqueries, or complex conditions can take longer to execute, as the database engine must perform additional processing to fulfill the request.

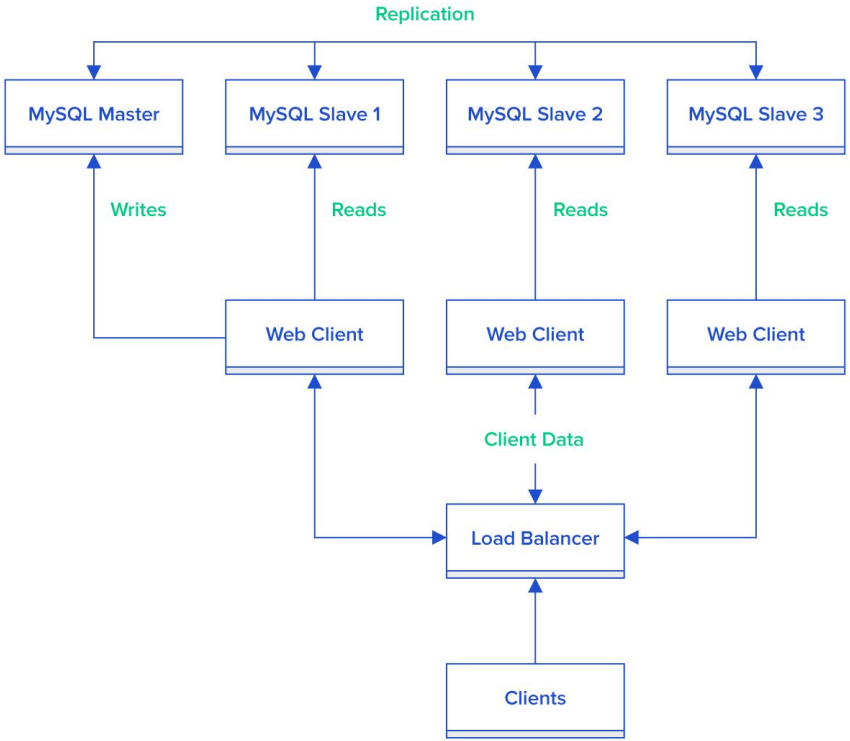
Scaling RDBMS

1. **Vertical Scaling**: This involves upgrading the hardware resources of the database server to improve performance. This can include increasing CPU, RAM, or storage capacity.
2. **Replication**: This involves creating multiple copies of the database on separate servers. Changes made to the primary database are then replicated to the secondary copies.
3. **Caching**: This involves storing frequently accessed data in a cache to improve query performance. This can be done using in-memory databases, such as Redis or Memcached.
4. **Partitioning**: This involves dividing large tables into smaller subsets, based on a partitioning key, and storing them separately. This can improve query performance by allowing the database to search a smaller subset of data, rather than scanning the entire table.
5. **Sherding**: This involves adding more servers to the database cluster to distribute the load and improve performance. In this approach, the database is sharded into smaller, more manageable subsets that can be distributed across multiple servers.

1. Vertical Scaling

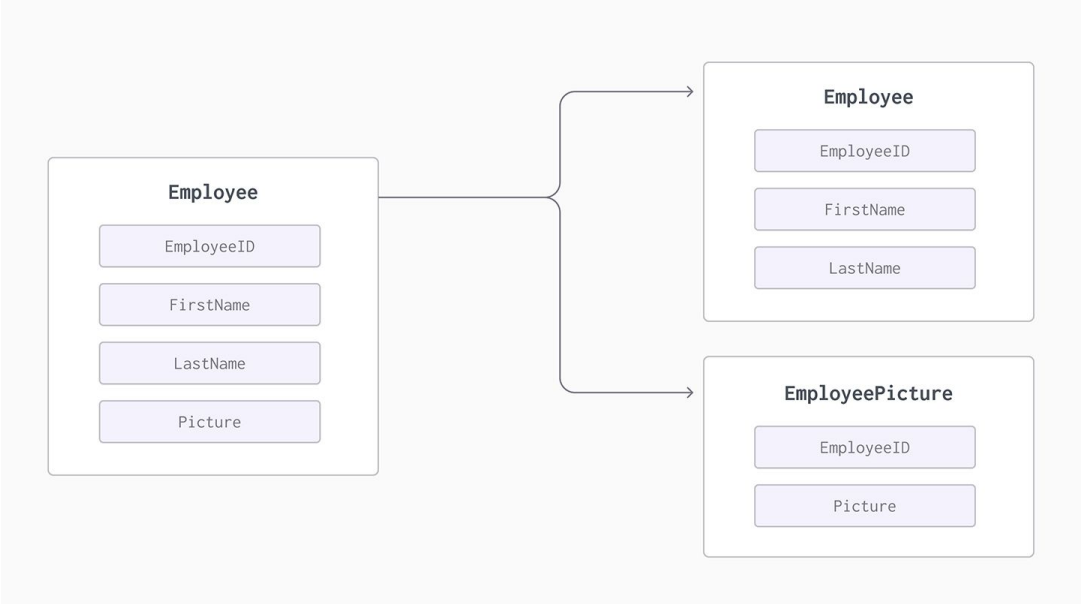


2. Replication: Master Slave

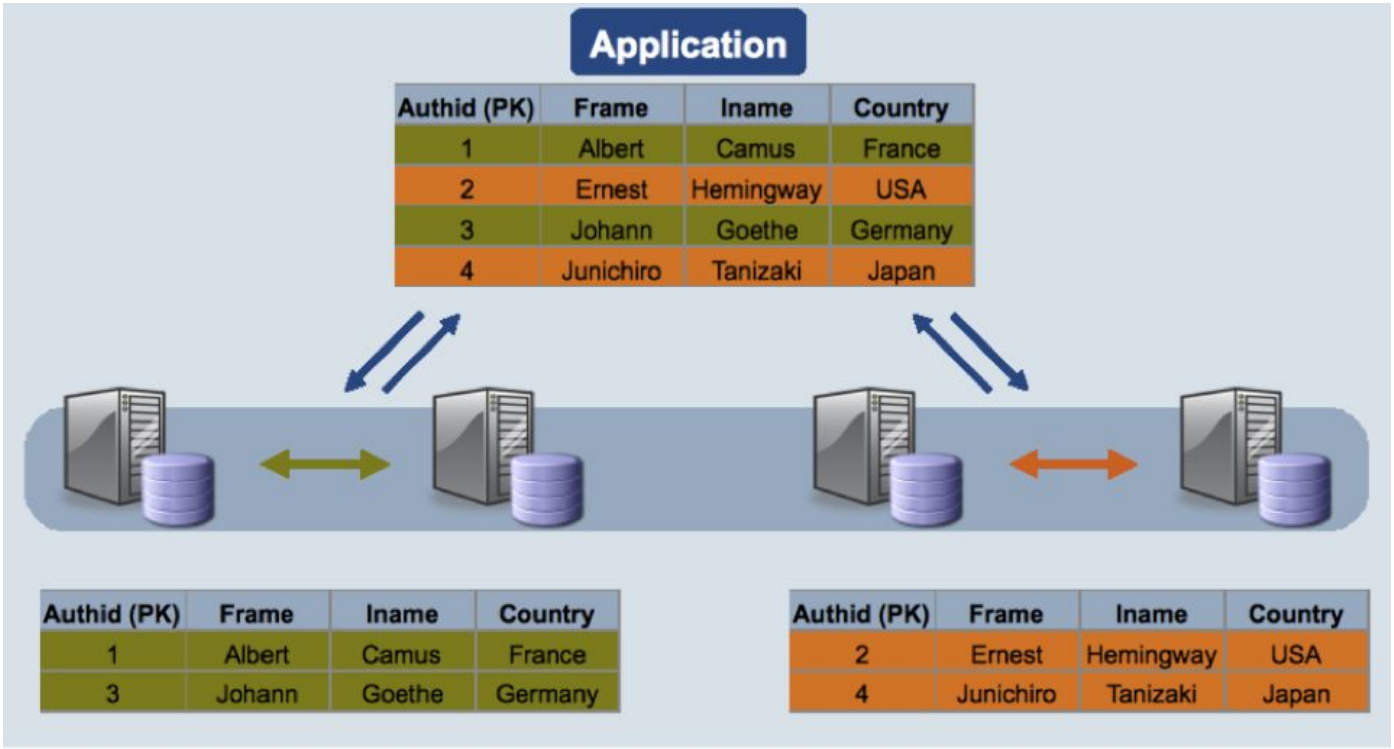


3. Caching

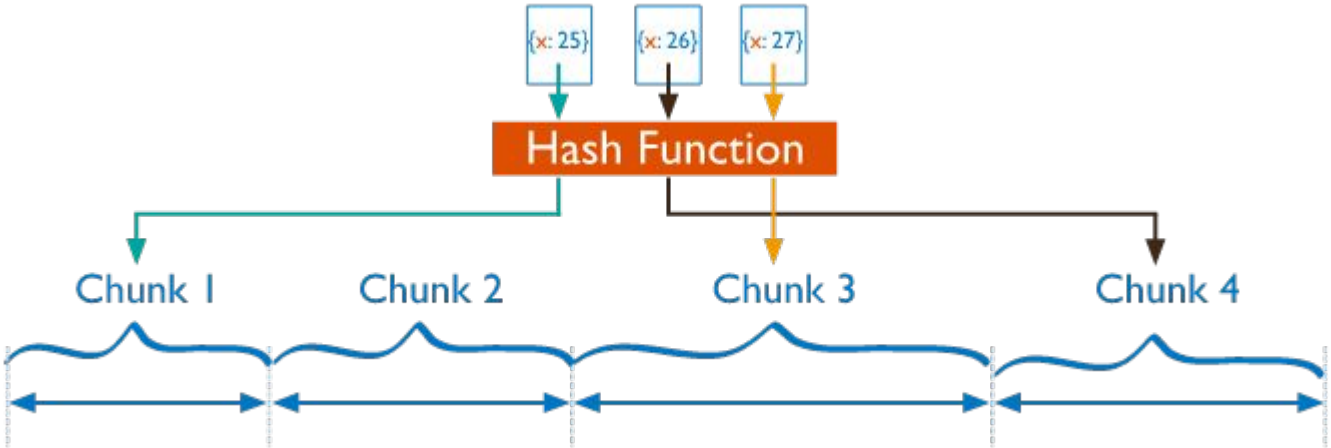
4. Partitioning



5. Data Sharding



5. Data Sharding (Cont.)



MySQL Master-Slave Replication

- MySQL Master-Slave replication is a process for copying data from a single database server to one or more additional servers.
- It can be used to distribute the load of database processing, improve data availability and provide redundancy.
- In MySQL, replication involves the master database writing down every change made to the data held in a special file known as the binary log.
- To set up MySQL Master-Slave replication, you need to enable binary logging on the master server and configure the slave server to connect to the master and replicate the data.
- Once replication has started, you need to monitor the process to ensure that it is working correctly and manage it to handle any errors or warnings that arise.
- You can monitor the replication process using the MySQL error log and the SHOW SLAVE STATUS command.

MySQL Replication on Kubernetes

We will deploy MySQL replication on Kubernetes using the following tutorial:

- https://kubernetes.io/docs/tasks/run-application/run-single-instance-stateful-application/
- https://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application/

Kubernetes Concepts

- When you create a headless service by setting clusterIP: None, no load-balancing is done and no cluster IP is allocated for this service. Only DNS is automatically configured. When you run a DNS query for headless service, you will get the list of the Pods IPs and usually client dns chooses the first DNS record.

- Overall, headless services can be a powerful tool in Kubernetes for scenarios where direct access to the pods is needed, such as in distributed databases, stateful sets, and other applications that require unique IP addresses for each instance.

- ConfigMap is an API object used to store configuration data separately from the container image or deployment specification. ConfigMaps allow you to decouple configuration details from container images and deployments, making it easier to manage and update configurations independently.

Kubernetes Concepts (Cont.)

- In Kubernetes, a PersistentVolume (PV) is a storage abstraction that represents a piece of networked storage in the cluster. A PV is a way to decouple the storage configuration from the pod definition, allowing for more flexibility in how storage is consumed and managed within a cluster.

- A PersistentVolumeClaim (PVC), on the other hand, is a request for a specific amount of storage from a PV. PVCs are used by pods to dynamically provision storage resources from a pool of available storage.

References

- https://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application/
- https://hevodata.com/learn/understanding-mysql-sharding-simplified/
- https://www.mysql.com/products/cluster/features.html
- https://github.com/mysql/mysql-ndb-operator
- https://dev.mysql.com/doc/ndb-operator/en/introduction.html