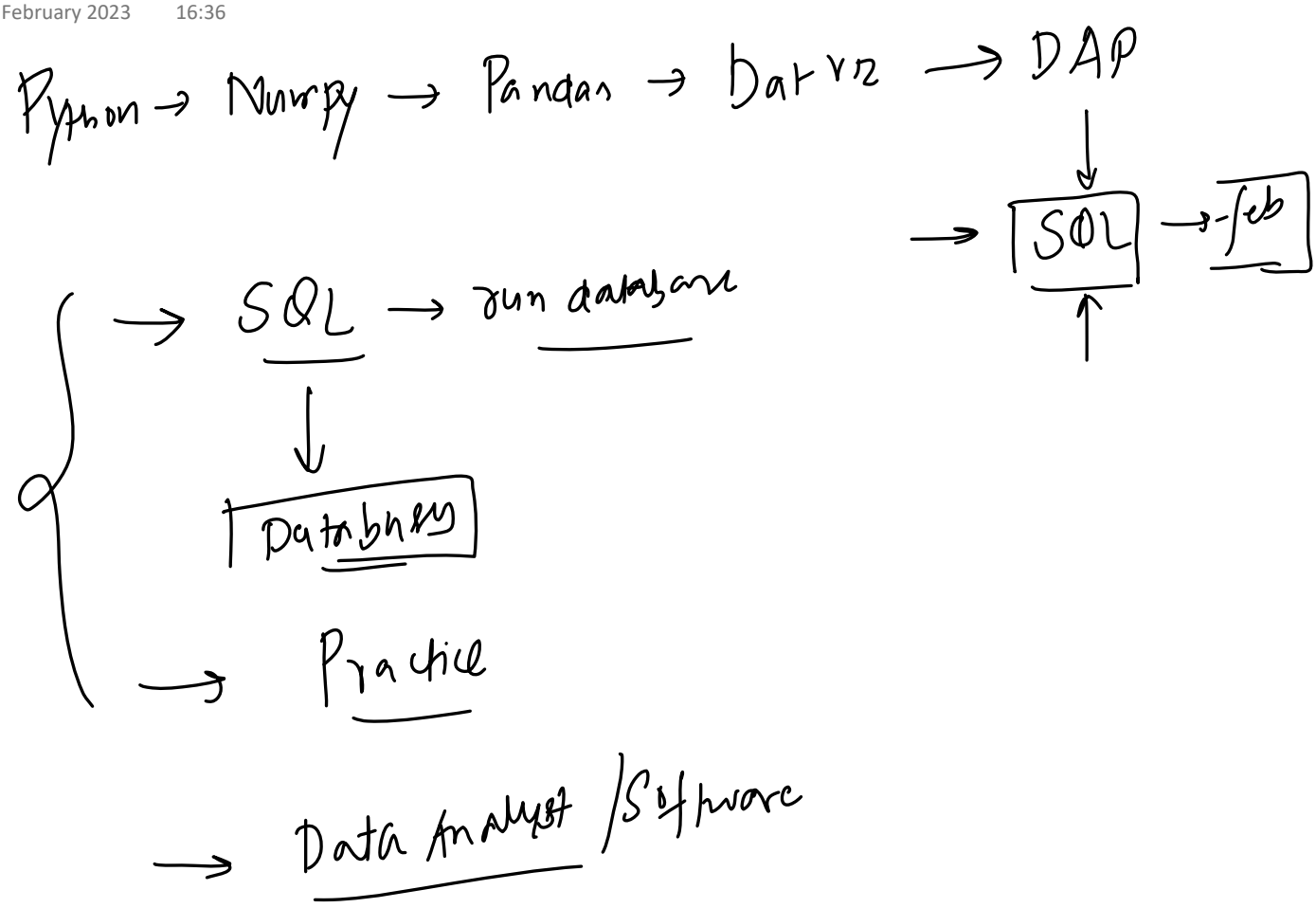
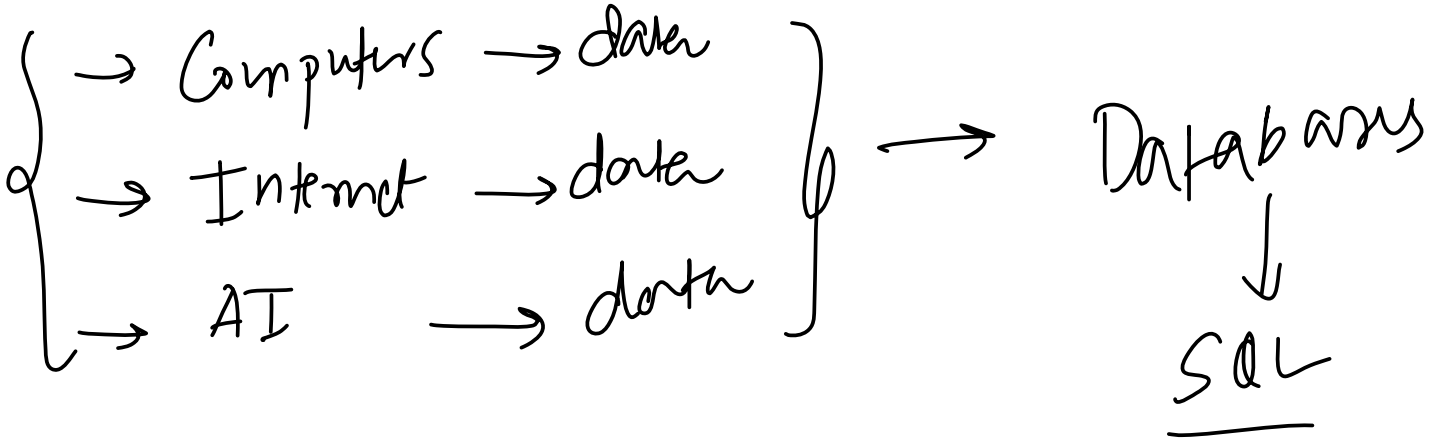
1. Before starting

06 February 2023 16:36



2. Importance of Data

06 February 2023 16:36



3. What are Databases?

06 February 2023 16:37

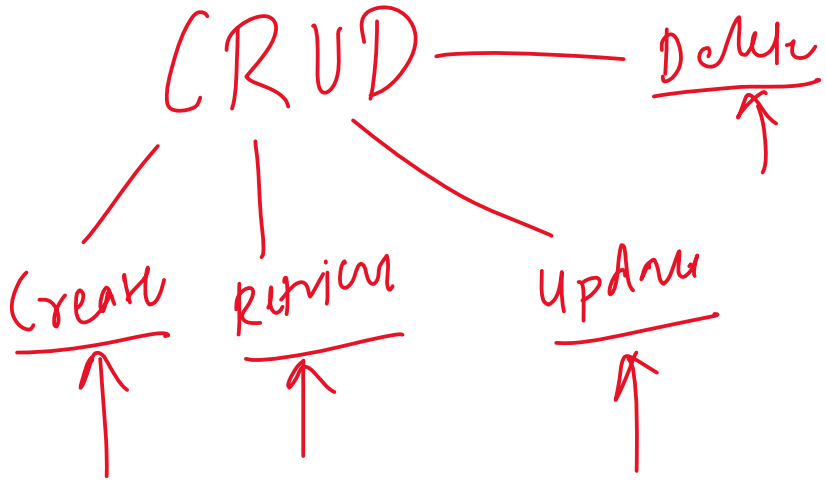
A Database is a shared collection of logically related data and description of these data, designed to meet the information needs of an organization

Data Storage: A database is used to store large amounts of structured data, making it easily accessible, searchable, and retrievable.

Data Analysis: A database can be used to perform complex data analysis, generate reports, and provide insights into the data.

Record Keeping: A database is often used to keep track of important records, such as financial transactions, customer information, and inventory levels.

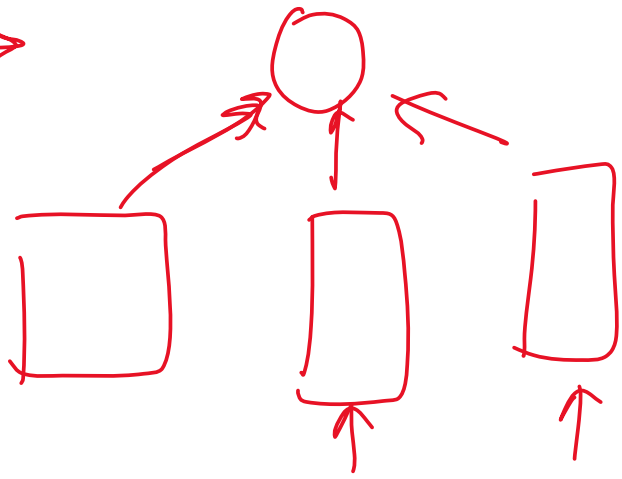
Web Applications: Databases are an essential component of many web applications, providing dynamic content and user management.



4. Properties of an Ideal Database

06 February 2023 16:37

1. Integrity
2. Availability
3. Security
4. Independent of Application
5. Concurrency

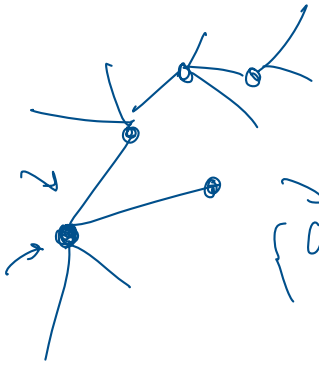


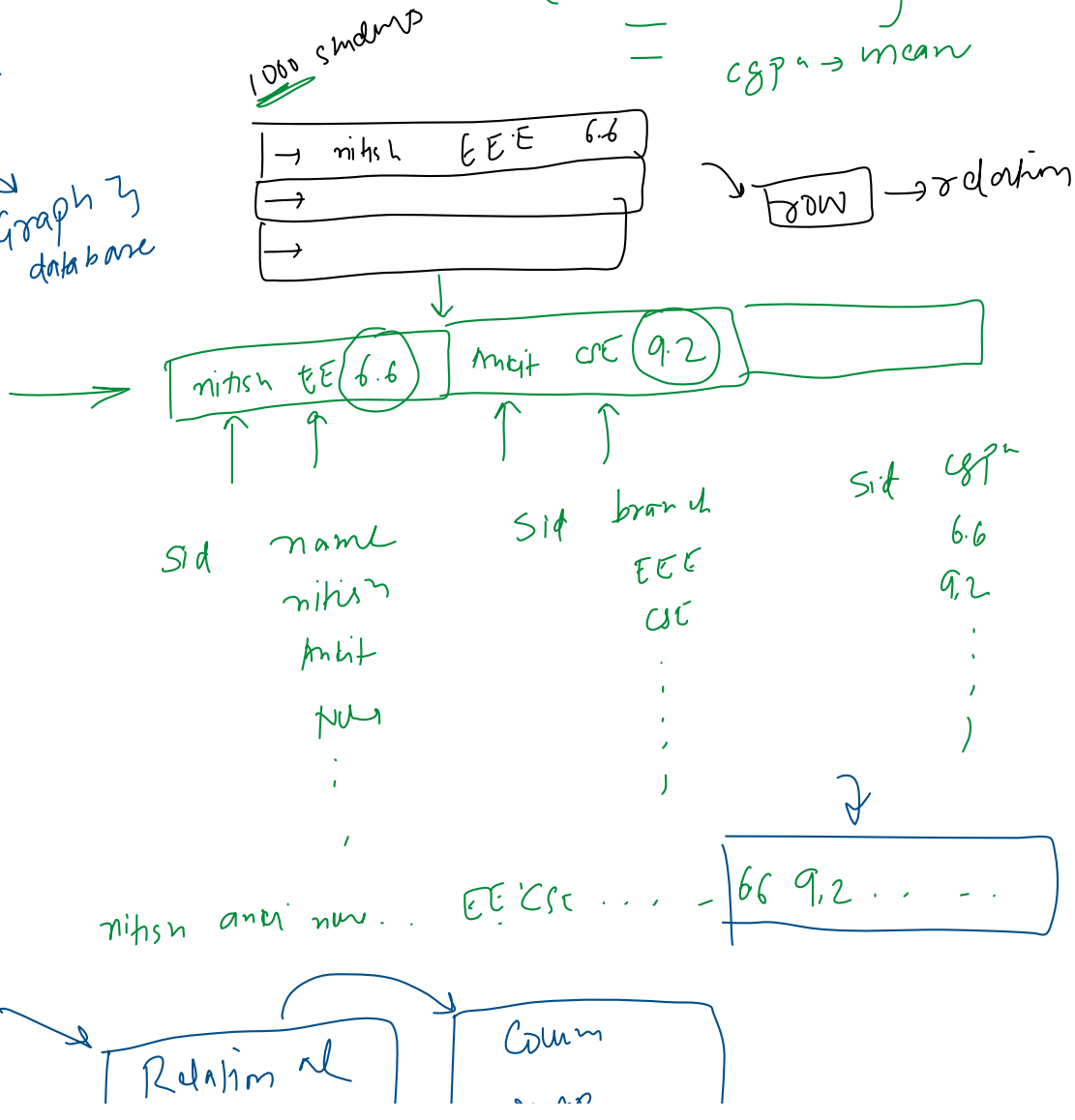
5. Types of Databases

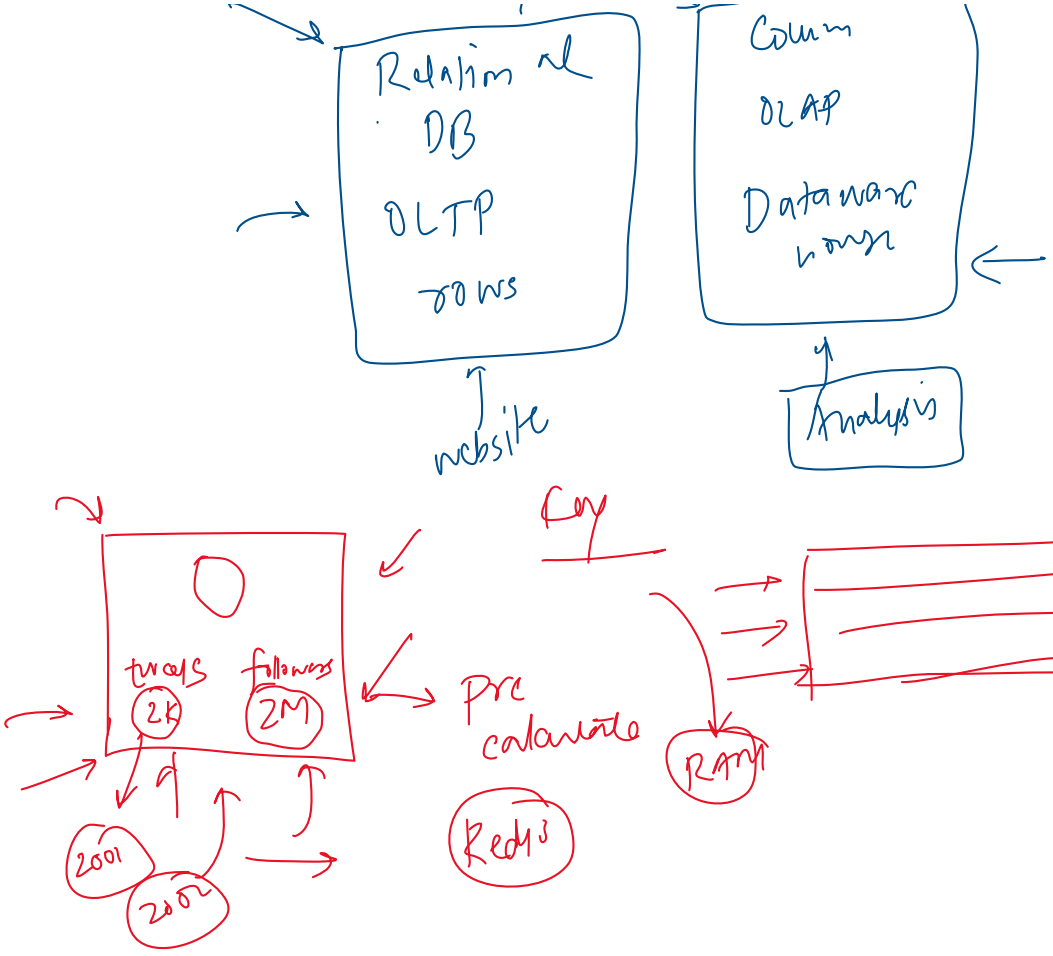
06 February 2023 16:42

1. Relational Databases -
   Also known as SQL databases, these databases use a relational model to organize data into tables with rows and columns.
2. NoSQL Databases -
   These databases are designed to handle large amounts of unstructured or semi-structured data, such as documents, images, or videos. (MongoDB)
3. Column Databases -
   These databases store data in columns rather than rows, making them well-suited for data warehousing and analytical applications. (Amazon Redshift, Google BigQuery)
4. Graph Databases -
   These databases are used to store and query graph-structured data, such as social network connections or recommendation systems. (Neo4j, Amazon Neptune)
5. Key-value databases -
   These databases store data as a collection of keys and values, making them well-suited for caching and simple data storage needs (Redis and Amazon DynamoDB)

Which one should you use?



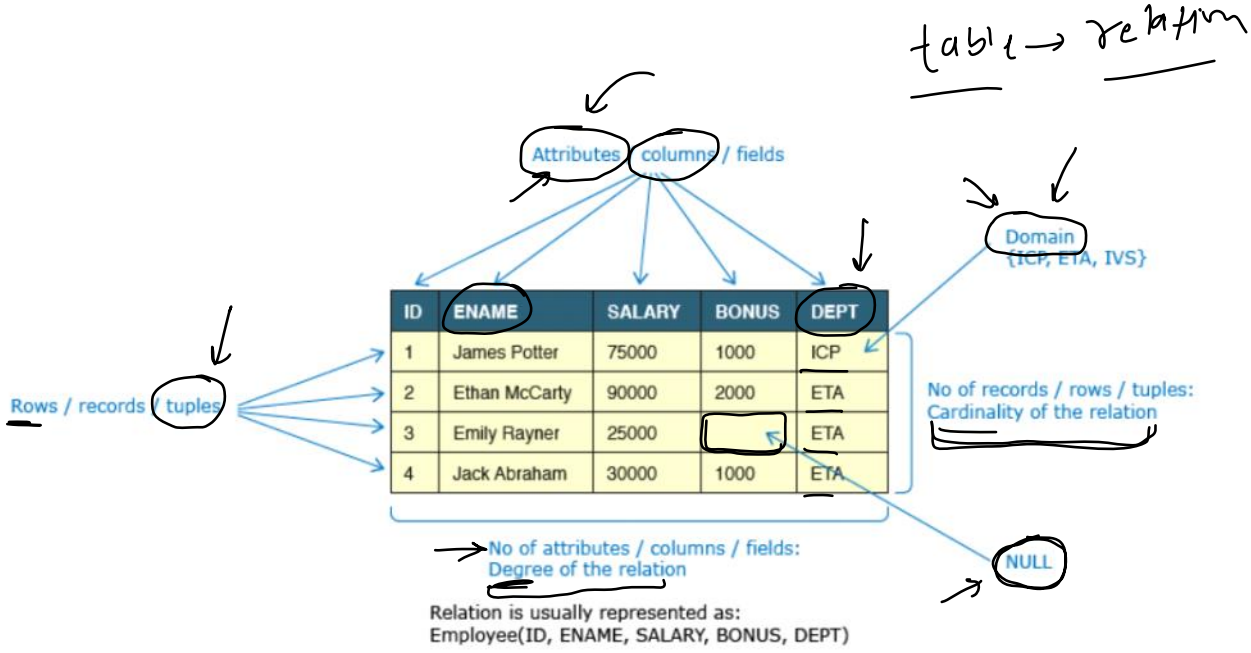



Relational
DB
OLTP
rows
website
Column
OLAP
Data warehouse
Analysis
tweets
followers
2K
2M
2001
200
Pre calculate
Redis
Key
RAM

6. Relational Databases

06 February 2023 16:42

Also known as SQL databases, these databases use a relational model to organize data into tables with rows and columns.

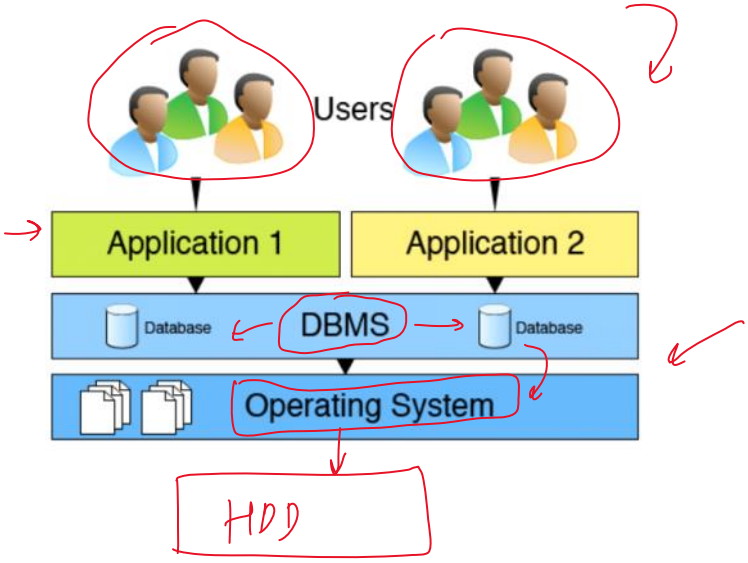


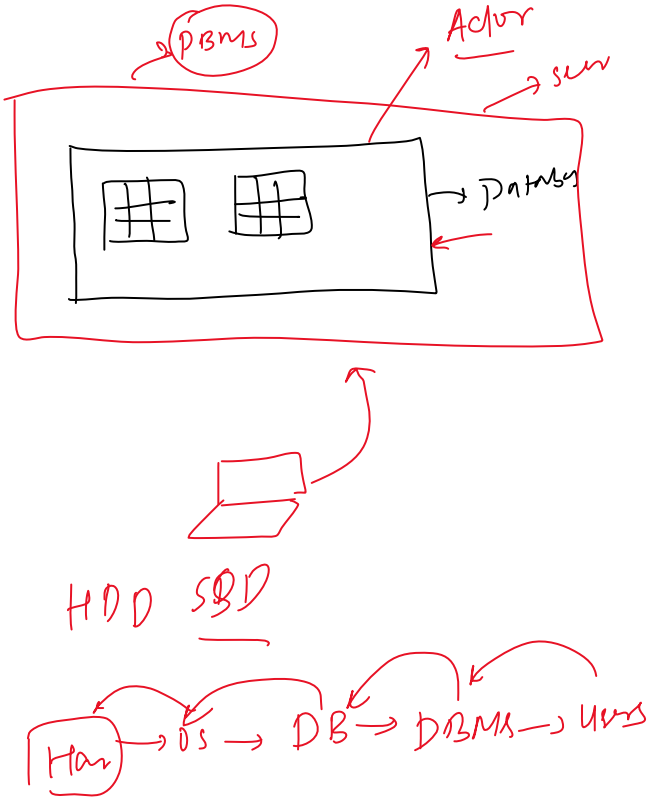
ID	ENAME	SALARY	BONUS	DEPT
1	James Potter	75000	1000	ICP
2	Ethan McCarty	90000	2000	ETA
3	Emily Rayner	25000		ETA
4	Jack Abraham	30000	1000	ETA

7. What is a DBMS

06 February 2023 16:41

A database management system (DBMS) is a software system that provides the interfaces and tools needed to store, organize, and manage data in a database. A DBMS acts as an intermediary between the database and the applications or users that access the data stored in the database.





8. Core Functionalities of a DBMS

06 February 2023 16:41

Functions of DBMS

Data Management - Store, retrieve and modify data

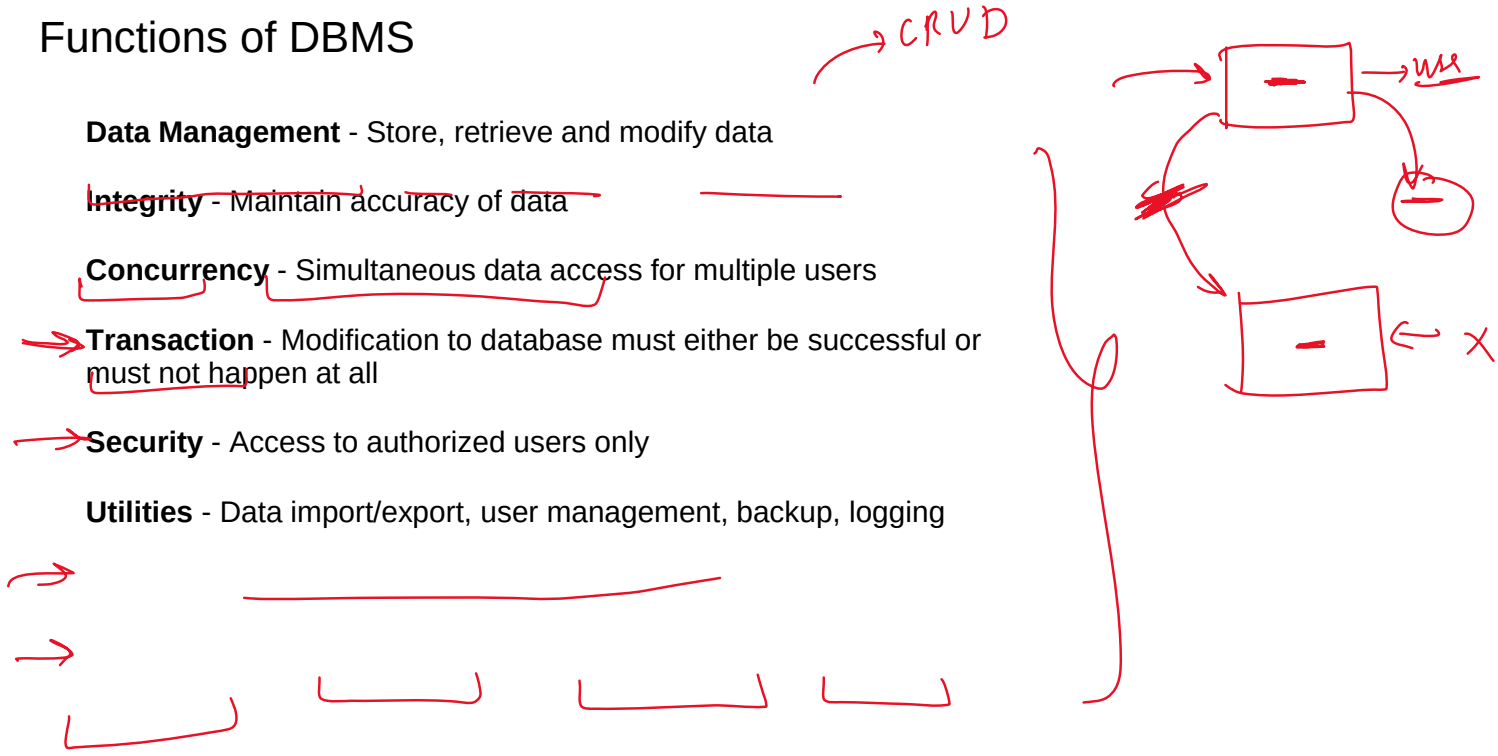
Integrity - Maintain accuracy of data

Concurrency - Simultaneous data access for multiple users

Transaction - Modification to database must either be successful or must not happen at all

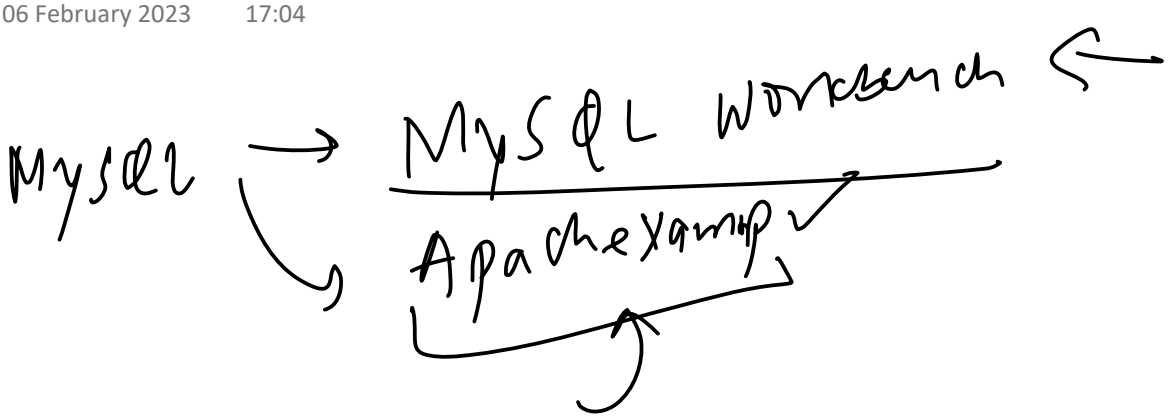
Security - Access to authorized users only

Utilities - Data import/export, user management, backup, logging



9. Practical

06 February 2023 17:04



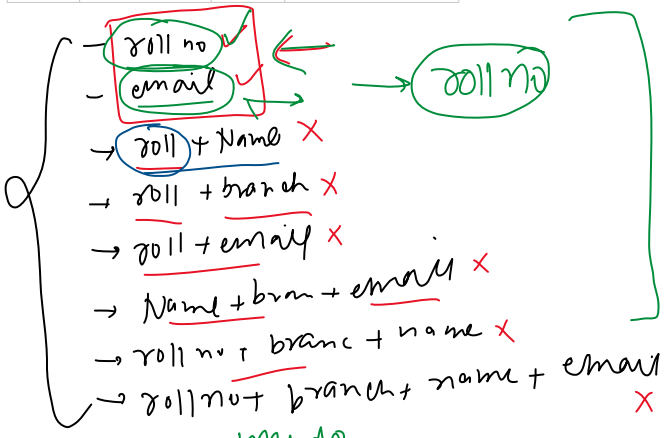
10. Database Keys

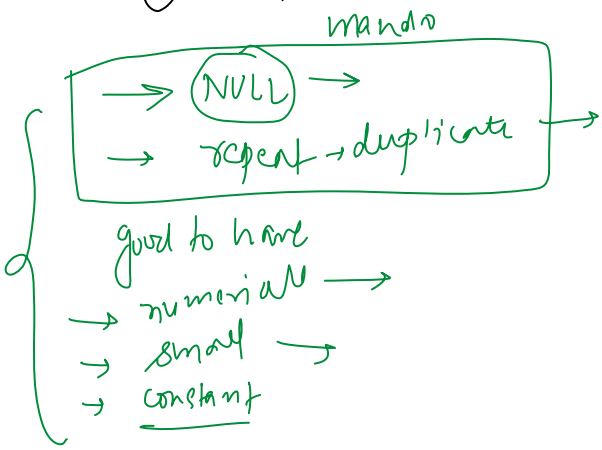
06 February 2023 17:07

A key in a database is an attribute or a set of attributes that uniquely identifies a tuple (row) in a table. Keys play a crucial role in ensuring the integrity and reliability of a database by enforcing unique constraints on the data and establishing relationships between tables.

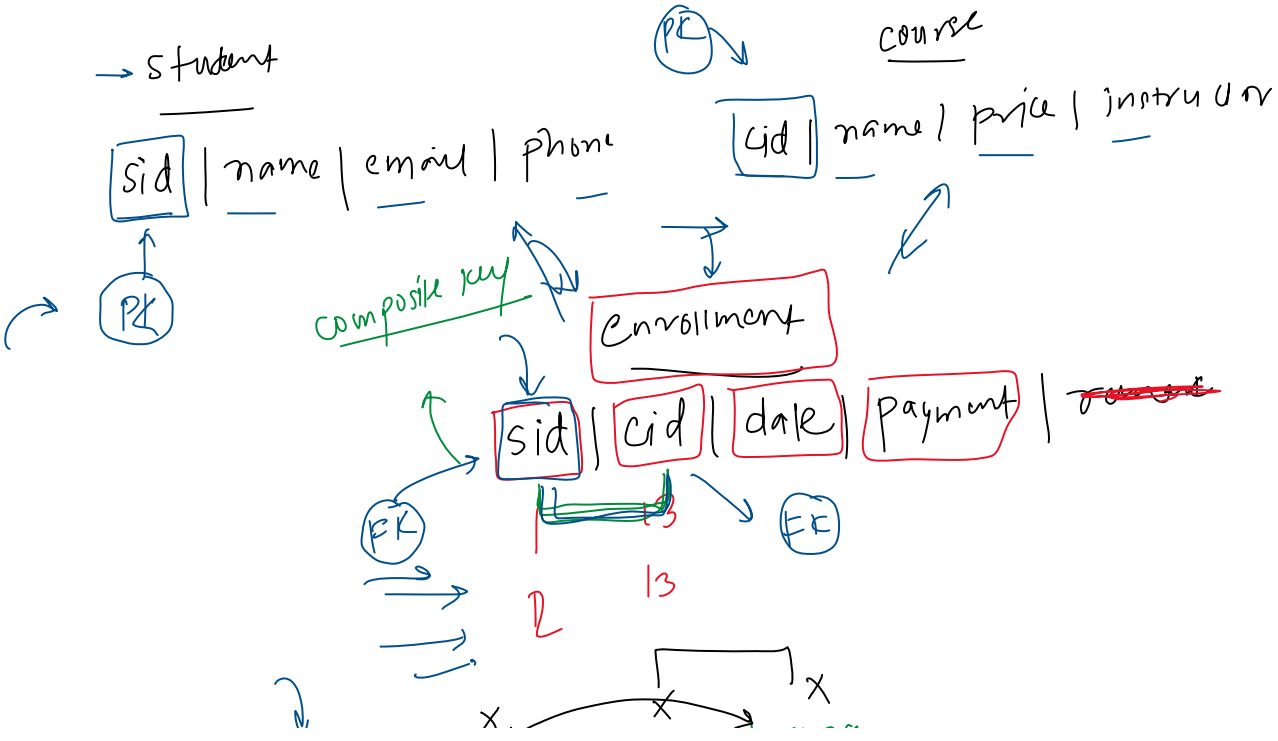
Roll no	Name	Branch	Email
1	Nitish Singh	CSE	nitish@gmail.com
2	Ankit Sharma	EEE	ankit@gmail.com
3	Neha Verma	ME	neha@gmail.com

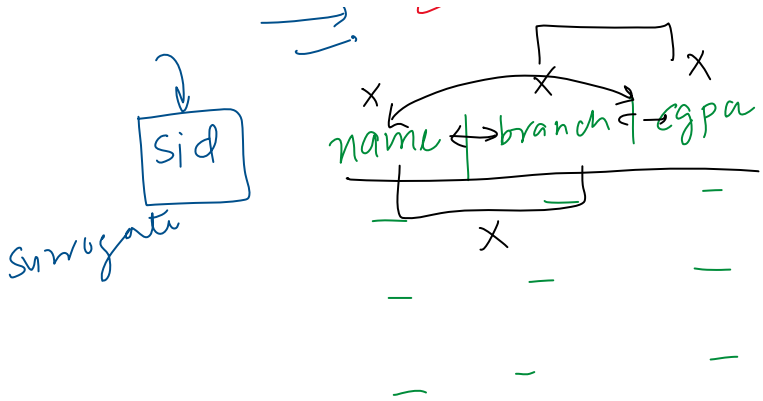
1. **Super Key** -
   A Super key is a combination of columns that uniquely identifies any row within a relational database management system (RDBMS) table
2. **Candidate key** -
   A candidate key is a minimal Super key, meaning it has no redundant attributes. In other words, it's the smallest set of attributes that can be used to uniquely identify a tuple (row) in the table
3. **Primary Key** -
   A primary key is a unique identifier for each tuple in a table. There can only be one primary key in a table, and it cannot contain null values.
4. **Alternate Key** -
   An alternate key is a candidate key that is not used as the primary key.
5. **Composite Key** -
   A composite key is a primary key that is made up of two or more attributes. Composite keys are used when a single attribute is not sufficient to uniquely identify a tuple in a table.
6. **Surrogate Key** -
7. **Foreign Key** -
   A foreign key is a primary key from one table that is used to establish a relationship with another table.





CK − PK = AK

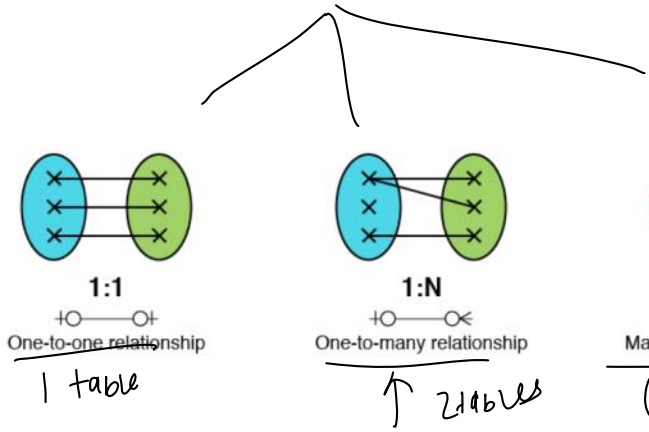



sid
Surrogate
name
branch
cgpa

11. Cardinality of Relationships

06 February 2023 16:43

Cardinality in database relationships refers to the number of occurrences of an entity in a relationship with another entity. Cardinality defines the number of instances of one entity that can be associated with a single instance of the related entity.



Examples

1. Person -> Driving License Number
2. Student -> college branch
3. Restaurants -> orders
4. Restaurants -> menu
5. Students -> courses

12. Drawbacks of Databases

06 February 2023 16:39

Complexity: Setting up and maintaining a database can be complex and time-consuming, especially for large and complex systems.

Cost: The cost of setting up and maintaining a database, including hardware, software, and personnel, can be high.

Scalability: As the amount of data stored in a database grows, it can become more difficult to manage, leading to performance and scalability issues.

Data Integrity: Ensuring the accuracy and consistency of data stored in a database can be a challenge, especially when multiple users are updating the data simultaneously.

Security: Securing a database from unauthorized access and protecting sensitive information can be difficult, especially with the increasing threat of cyber attacks.

Data Migration: Moving data from one database to another or upgrading to a new database can be a complex and time-consuming process.

Flexibility: The structure of a database is often rigid and inflexible, making it difficult to adapt to changing requirements or to accommodate new types of data.

Row vs Column Oriented Databases

Last modified: June 08, 2020

There are two ways to organize relational databases:

- **Row oriented**
- **Column oriented** (also known as **columnar** or **C-store**)

Row oriented databases are databases that organize data by record, keeping all of the data associated with a record next to each other in memory. Row oriented databases are the traditional way of organizing data and still provide some key benefits for storing data quickly. They are optimized for reading and writing rows efficiently.

Common row oriented databases:

- Postgres
- MySQL

Column oriented databases are databases that organize data by field, keeping all of the data associated with a field next to each other in memory. Columnar databases have grown in popularity and provide performance advantages to querying data. They are optimized for reading and computing on columns efficiently.

Common column oriented databases:

- Redshift
- BigQuery
- Snowflake

Row Oriented Databases

Traditional Database Management Systems were created to store data. They are optimized to read and write a single row of data which lead to a series of design choices including having a row store architecture.

In a row store, or row oriented database, the data is stored row by row, such that the first column of a row will be next to the last column of the previous row.

For instance, let's take this Facebook_Friends data:

Facebook_Friends

Name	City	Age
Matt	Los Angeles	27
Dave	San Francisco	30
Tim	Oakland	33

This data would be stored on a disk in a row oriented database in order row by row like this:

Matt	Los Angeles	27	Dave	San Francisco	30	Tim	Oakland	33

This allows the database write a row quickly because, all that needs to be done to write to it is to tack on another row to the end of the data.

Writing to Row Store Databases

Let's use the data stored in a database:

Matt	Los Angeles	27	Dave	San Francisco	30	Tim	Oakland	33

If we want to add a new record:

Jen	Vancouver	30

We can just append it to the end of the current data:

Matt	Los Angeles	27	Dave	San Francisco	30	Tim	Oakland	33	Jen	Vancouver	30

Row oriented databases are still commonly used for Online Transactional Processing (OLTP) style applications since they can manage writes to the database well. However, another use case for databases is to analyze the data within them. These Online Analytical Processing (OLAP) use cases need a database that can support ad hoc querying of the data. This is where row oriented databases are slower than C-store databases.

Reading from Row Store Databases

Row oriented databases are fast at retrieving a row or a set of rows but when performing an aggregation it brings extra data (columns) into memory which is slower than only selecting the columns that you are performing the aggregation on. In addition the number of disks the row oriented database might need to access is usually larger.

Extra data into Memory

Say we want to get the sum of ages from the Facebook_Friends data. To do this we will need to load all nine of these pieces of data into memory to then pull out the relevant data to do the aggregation.

Matt	Los Angeles	27	Dave	San Francisco	30	Tim	Oakland	33

This is wasted computing time.

Number of Disks accessed

Let's assume a Disk can only hold enough bytes of data for three columns to be stored on each disk. In a row oriented database the table above would be stored as:

Disk 1		
Name	City	Age
Matt	Los Angeles	27

Disk 2		
Name	City	Age
Dave	San Francisco	30

Disk 3		
Name	City	Age
Tim	Oakland	33

To get the sum of all the people's ages the computer would need to look through all three disks and across all three columns in each disk in order to make this query.

So we can see that while adding data to a row oriented database is quick and easy, getting data out of it can require extra memory to be used and multiple disks to be accessed.

Column Oriented Databases

Data Warehouses were created in order to support analyzing data. These types of databases are read optimized.

In a C-Store, columnar, or Column-oriented database, the data is stored such that each row of a column will be next to other rows from that same column.

Let's look at the same data set again and see how it would be stored in a column oriented database.

Facebook_Friends

Name	City	Age
Matt	Los Angeles	27
Dave	San Francisco	30
Tim	Oakland	33

A table is stored one column at a time in order row by row:

Matt	Dave	Tim	Los Angeles	San Francisco	Oakland	27	30	33

Writing to a Column Store Databases

If we want to add a new record:

Jen	Vancouver	30

We have to navigate around the data to plug each column in to where it should be.

Matt	Dave	Tim	Jen	Los Angeles	San Francisco	Oakland	Vancouver	27	30	33	30

If the data was stored on a single disk it would have the same extra memory problem as a row oriented database, since it would need to bring everything into memory. However, column oriented databases will have significant benefits when stored on separate disks.

If we placed the table above into the similarly restricted three columns of data disk they would be stored like this:

Disk 1		
Name		
Matt	Dave	Tim

Disk 2		
City		
Los Angeles	San Francisco	Oakland

Disk 3		
Age		
27	30	33

Reading from a Column store Database

To get the sum of the ages the computer only needs to go to one disk (Disk 3) and sum all the values inside of it. No extra memory needs to be pulled in, and it accesses a minimal number of disks.

While this is a slight over simplification, it illustrates that by organizing data by column the number of disks that will need to be visited will be reduced and the amount of extra data that has to be held in memory is minimized. This greatly increases the overall speed of the computation.

There are other ways in which a column oriented database can get more performance.

Coding the data into more compact forms

Let's first examine an encoding technique that can be used by row or column oriented databases. The example of one of the columns being for states of the United States will show dictionary and bitmap encodings.

- There are 50 so we could encode the whole database with 6 bits since this would provide us 64 unique patterns.
- To store the actual abbreviations would require 16 bits since this would provide us with 256 unique patterns for each of the two ASCII characters.
- Worst of all if we stored the full name the lengths would be variable and the amount of bits needed would be a lot more.

Now let's take a look at Run-length encoding. This allows you to replace any sequence of the same value with a count and value indicator. For instance we can replace aaaab with 4a1b. This becomes even more powerful when you create projections with columns that are sorted since all values that are the same are next to each other.

Compressing the data

If each piece of data is the same number of bits long then all of the data can be further compressed to be the number of pieces of data times that number of bits for a single piece of data.

Ordering the data

When doing ad hoc queries there are a number of different sort orders of the data that would improve performance. For instance, we might want data listed by date, both ascending and descending. We might be looking for a lot of data on a single customer so ordering by customer could improve performance.

In Row oriented databases, indexes can be created but data is rarely stored in multiple sort orders. However, in Column oriented databases you can have the data stored in an arbitrary number of ways. In fact, there are benefits beyond query performance. These different sort ordered columns are referred to as projections and they allow the system to be more fault tolerant, since the data is stored multiple times.

Original Database (WS)

Disk 1		
Name		
Matt	Dave	Tim

Disk 2		
City		
Los Angeles	San Francisco	Oakland

Disk 3		
Age		
27	30	33

Name Ordered ASC (RS)

Disk 1		
Name		
Dave	Matt	Tim

Disk 2		
City		
San Francisco	Los Angeles	Oakland

Disk 3		
Age		
30	27	33

Name Ordered DESC (RS)

Disk 1		
Name		
Tim	Matt	Dave

Disk 2		
City		
Oakland	Los Angeles	San Francisco

Disk 3		
Age		
33	27	30

This seems like a complicated set of tables to update, and it is. This is why the architecture of a C-store database has a writeable store (WS) and a read optimized store (RS). The writeable store has the data sorted in the order it was added, in order to make adding data into it easier. We can easily append the relevant fields to our database as seen below:

Original Database (WS)

Disk 1			
Name			
Matt	Dave	Tim	Evan

Disk 2			
City			
Los Angeles	San Francisco	Oakland	Blacksburg

Disk 3			
Age			
27	30	33	20

Then the read-optimized store can have multiple projections. It then has a tuple mover which manages the relevant updates from the WS to the RS. It has to navigate the multiple projections and insert the data in the proper places.

Name Ordered ASC (RS)

Disk 1			
Name			
Dave	Evan	Matt	Tim

Disk 2			
City			
San Francisco	Blacksburg	Los Angeles	Oakland

Disk 3			
Age			
30	20	27	33

Name Ordered DESC (RS)

Disk 1			
Name			
Tim	Matt	Evan	Dave

Disk 2			
City			
Oakland	Los Angeles	Blacksburg	San Francisco

Disk 3			
Age			
33	27	20	30

This architecture means that while the data is being updated from the WS to the RS the partially added data must be ignored by queries to the RS until the update is complete.

Summary

Column Oriented databases came out with a 2005 paper explaining the design that Redshift, BigQuery and Snowflake are all built upon. It's why they all have about the same performance and relative costs. This column oriented database is being used by most major providers of cloud data warehouses. This has become the dominant architecture in relational databases to support OLAP.

Written by: Blake Barnhill, Matt David
Reviewed by:

Find more free data resources at DataSchool.com

Our Web Books

DATABASES

What is Redis and Why is it used by leading industries?

By Jaidev Singh Bhui - June 02, 2021 - 10 min read

In this age of modern technology, no one likes to wait for a long time for their search results or Twitter feeds to show up. Similarly, if you're playing a game you would want to view the leaderboard updated in real-time. All these needs require a solution that is highly performant and rapid, which helps us in accessing data faster. Re

What is Redis?

Redis actually stands for Remote Dictionary Server.

It is basically a data-structure store that stores the data in the primary memory of a computer system. It uses key-value pairs to store the data, which is just like a HashMap in Java, a dictionary in Python, or an object in JavaScript. This is also why it is sometimes referred to as a NoSQL database.

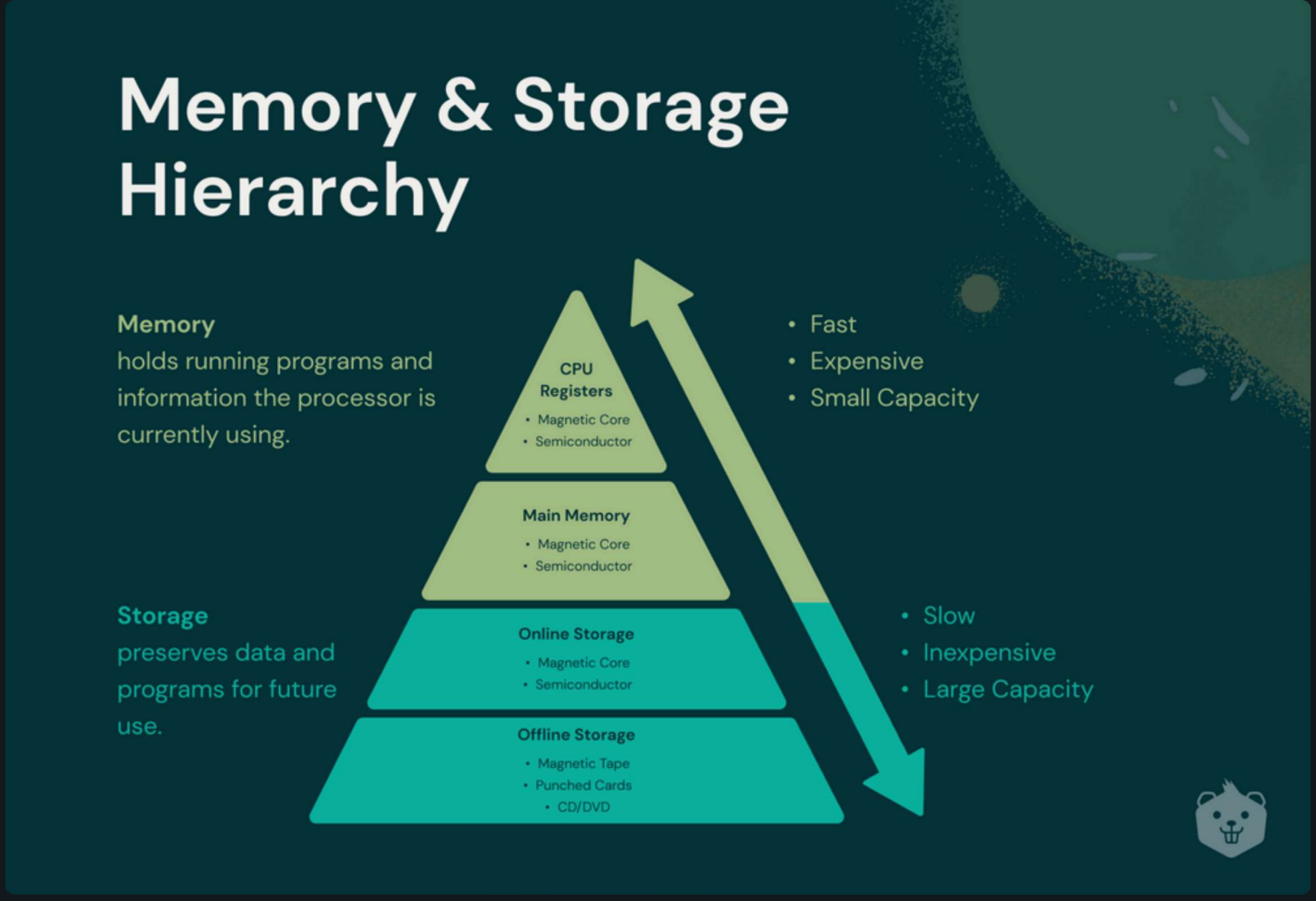
Redis is an open-source project with a flourishing community.

Features of Redis

In-memory storage

Well, conventionally all the databases store and access the data from hard disks and SSDs, which are secondary memory. As we all know, primary memory is faster than secondary memory, as it can be accessed directly by the processor.

Now, since Redis stores its data on the primary memory, reading and writing are made faster than databases that store data on disks.



Memory & Storage Hierarchy

This is also why Redis is used as a cache in many applications, to provide results rapidly. But we could store our data directly on the primary memory or in the system cache, we won't require Redis then, right?

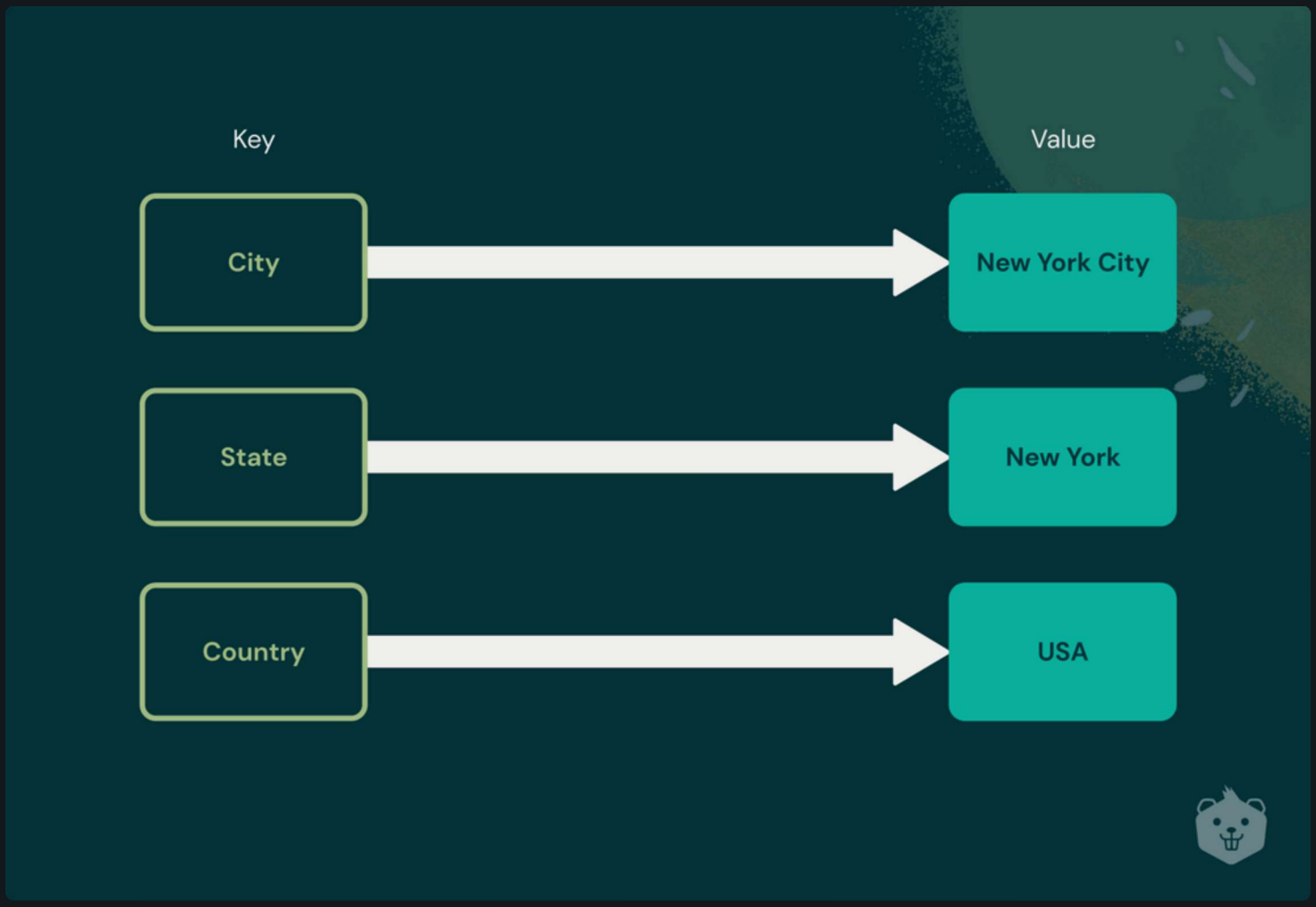
Redis is so much more than 'just a cache', as you will see.

Advanced Data Structures

Redis stores its data in a key-value pair and has the ability to store the data using a variety of data structures like:

- Strings
- Lists
- Sets
- Sorted Sets
- Hashes

- HyperLogLogs
- Streams



Key-value store

This is not possible if you would wish to store data directly in the memory, without using Redis. Even Memcached, which is another very popular in-memory key-value store used as a caching mechanism, only supports Strings but not such data structures which Redis provides.

As a developer, you must have used some of these data structures, which also makes Redis easier to use and implement.

There is no serialization required here since the data is stored directly in the form of the data structures you are coding in. Had this not been the case, you would have to convert or serialize the data into Strings before storing it into any other data store or database.

Persistence

Another disadvantage of directly using primary memory or system cache is the data being volatile. All the data on primary memory gets washed off or cleaned as soon as there is no power source or when the system is shut down.

Redis offers two mechanisms to help prevent this - Snapshots and AOFs (Append Only Files), which are basically a backup of the Redis data-store on the secondary memory, so that if there is a system failure or if there is a power outage, it can recall the current state of the database, by loading data from these backups present on the disks.

Replication

Redis uses master-slave (primary-replica) architecture, where data can be replicated to multiple replica servers. This boosts the read performance since requests can split among different servers. This also helps in faster recovery in case the master or primary server experiences an outage.

Huge Support

Being an Open Source project with a diverse community, Redis has no technology constraint, as it is based on open standards, and supports open data formats. It also supports a rich set of clients, with support in more than 40 programming languages.

Since Redis stores everything in-memory, in the form of a wide variety of data structures, and also provides persistence and the ability of replicating servers, it stands out when comparing to other databases/caches which cannot serve the same purpose as Redis.

Here is a quick comparison of Redis with other major databases and key-value stores.

Name	Type	Data storage options	Query types	Additional features
Redis	In-memory non-relational database	Strings, lists, sets, hashes, sorted sets	Commands for each data type for common access patterns, with bulk operations, and	Publish/Subscribe, master/slave replication, disk persistence, scripting

			partial transaction support	
memcached	In-memory key-value cache	Mapping of keys to values	Commands for create, read, update, delete, and a few others	Multithreaded server for additional performance
MySQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions	SELECT, INSERT, UPDATE, DELETE, functions, stored procedures	ACID compliant (with InnoDB), master/slave and master/master replication
PostgreSQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions, customizable types	SELECT, INSERT, UPDATE, DELETE, built-in functions, custom stored procedures	ACID compliant, master/slave replication, multi-master replication (third party)
MongoDB	On-disk non-relational document store	Databases of tables of schema-less BSON documents	Commands for create, read, update, delete, conditional queries, and more	Supports map-reduce operations, master/slave replication, sharding, spatial indexes

Where is Redis being used?

Take a look at some prominent uses of Redis with real-world examples:

Twitter - Caching with low latency

Twitter is a social-networking site, which is much like any other social media application out there, with features like a user can create posts, like other user's posts, comment on a post, or even follow other users. Posts in this context are tweets, but let's keep it general.

You would want to save user profiles in a cache, with information like number of posts, followers and following, so that a lot of overhead is reduced on the system, to fetch data from the database and recompute every time the user asks for it.

Similarly, when you would open the application signed in as a user, the first screen you would see is your feed or timeline, which would have posts from all the users you follow.

Now, of course, if you follow only 2-3 users, getting their posts in reverse order and then merging them is not that heavy of a computation. But if it's a pretty big application, with millions of users, where they are following hundreds of thousands of users, this performs very poorly, as this needs to be computed for every user, every time they open the app.

This is where Redis comes into the picture and solves the problem by computing and storing all of these timelines in a cache, and serving it to the clients from there.

It's faster, provides lower latency, and performs way better.

This is made even easier by using appropriate data structures like lists in Redis.

Twitter has its own data structures built on Redis (ziplists), to further optimize the solution.

Twitter Timeline (source: https://twitter.com)

Pinterest - Caching

Pinterest is another social media application, which uses Redis as a cache to store:

- A list of users you are following
- A list of users following you

- A list of users who are following your boards
- A list consisting of the boards you unfollowed after following a user
- The followers and unfollowers of each board

Similarly, Twitter and Instagram also use Redis to store their timelines, feeds and social graphs.

Dream11/My11Circle - Highly performant leaderboards with concurrent read and writes

These are the leading fantasy team gaming apps in India, with millions of users participating in contests every day. The users need to create virtual teams from real players playing on the field, from the world of cricket, football, basketball, and other sports. The users earn points according to how well their players perform in the matches.

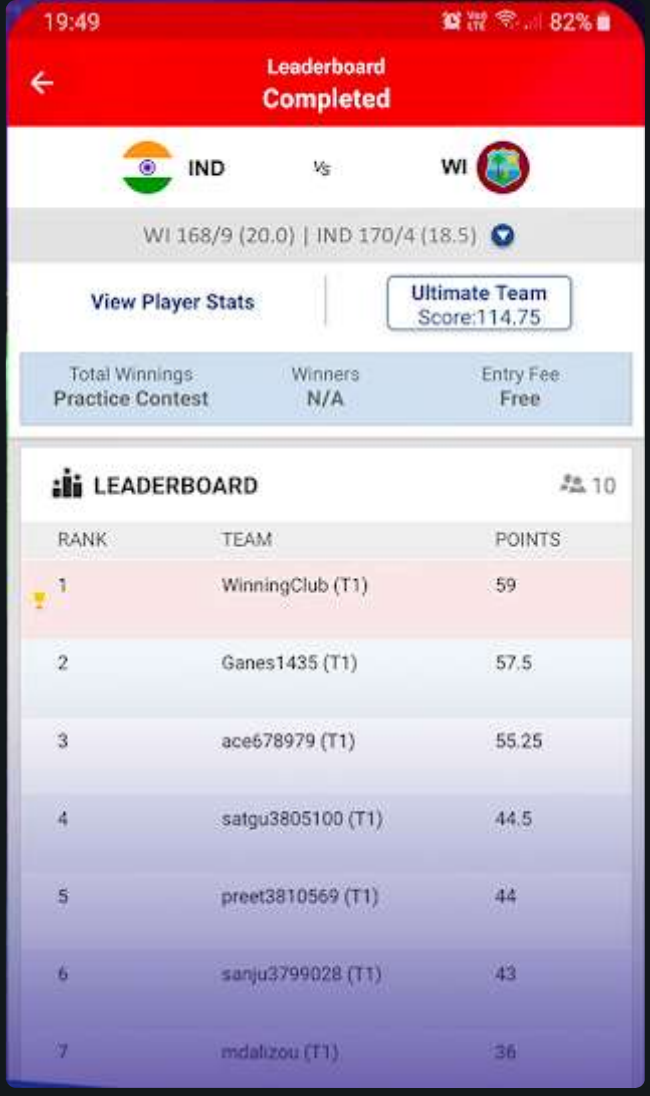
Both of these apps make use of the sorted set data structure of Redis, to generate gaming leaderboards with millions of entries being updated in real-time.

Sorted set takes keys and keeps them in an order based on the value, so if it's a gaming leaderboard like the ones in Dream 11, the key would be the user id and the value would be the points their team scored, the users on top having the highest points.

Since these apps are dependent on real-time events happening on the field, most users create or update their team after the playing 11 (for a cricket or football match) is announced, which is around 30 minutes prior to the start of a game.

Naturally, there is a huge load on the database to make concurrent writes and reads during this time.

Redis is highly efficient here as well, as it is an in-memory, single-threaded store, which writes data in a sequential manner. These sequential writes maintain consistency at the caching layer and have helped scale these applications.



My11Circle Leaderboard (source: https://play.google.com/store/apps)

StackOverflow - Displaying top-voted answers on top

The one site which doesn't need any introduction to all the developers out there. For others, it's basically a Q&A site like Quora, where technical problems are discussed.

Sorted sets are used here as well, to keep the most upvoted or liked answers on the top, to maintain good quality content.

Valorant - Load balancing across server instances

This is an FPS (First-Person-Shooter) video game, which is getting very popular and

system.

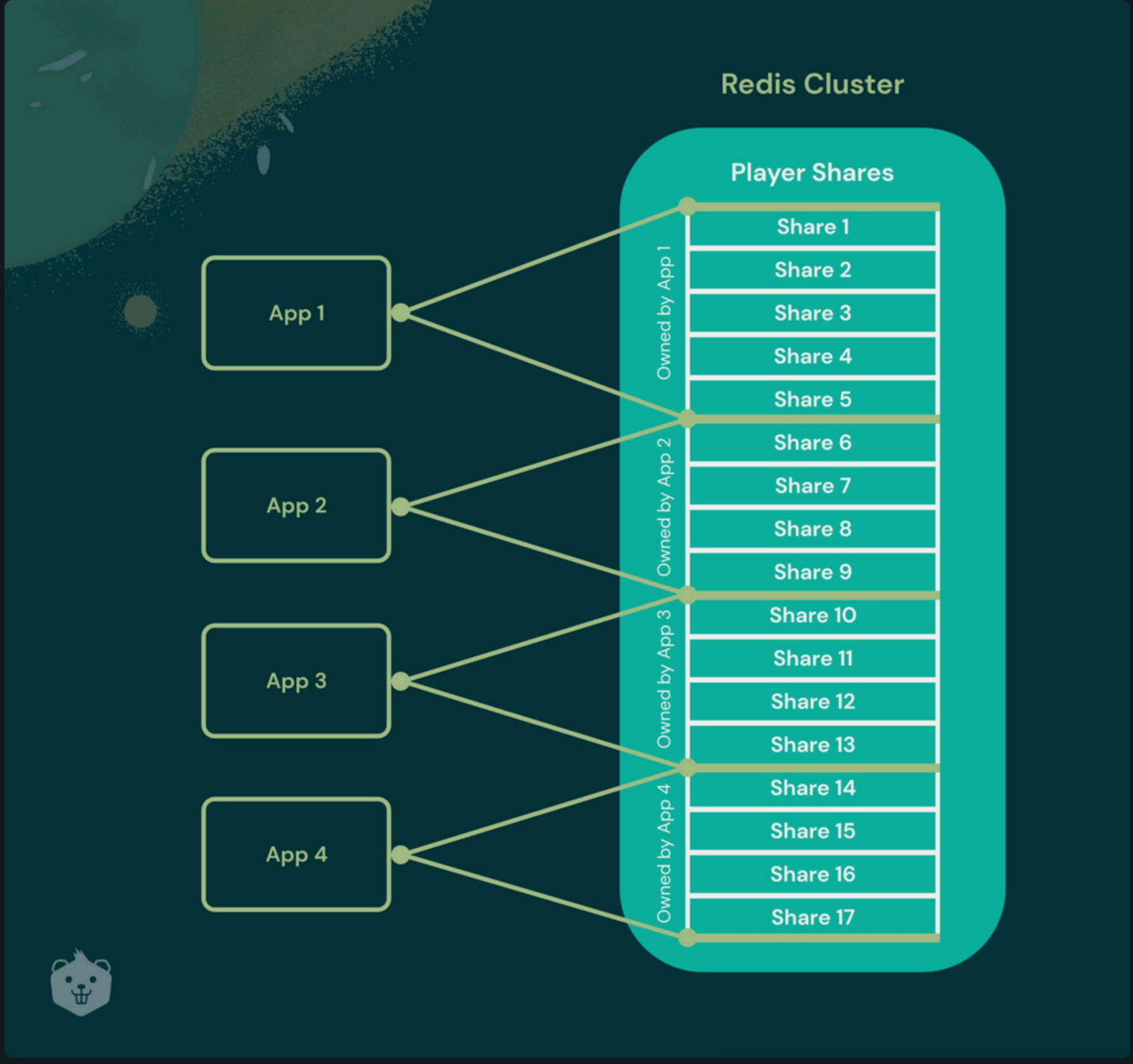
Let's take the example of a session microservice, which checks whether a player is online or not. There are multiple instances of the application running to handle the load.

Now, if the user is disconnected or their PC crashed, which does not explicitly call or trigger the microservice, there is a question as to which session service instance should manage this, by counting whether the inactive time exceeds the threshold?

Note that this has to be done for every user at the same time.

This is being solved by using hashes, storing in a Redis cluster the user id in a list based on the modulo of their player ID hash. When the service starts, each app instance checks the list and takes up a share of users by registering their own instance ID. When there are additional instances started to handle the load, the shares get distributed evenly amongst the instances.

Dream11 also uses Redis for a similar use case of task distribution.



Load balancing using Redis

Doordash - Machine Learning inference with low latency

Doordash is an online food delivery company, which is the largest food delivery company in the United States. Being such a large company, with millions of users, and since they also build Machine Learning (ML) models, they have feature data that can grow to billions of records with millions of records being used for inference in real-time.

Machine Learning inference is the process of running live data points into an ML model

during inference.

To match the low latency constraints while processing such a large dataset in real-time, Redis hashes are being used, which when benchmarked with other competitor software, performs the best.

Home Depot - Real-time data analytics

Home Depot is the largest home improvement retailer in the United States, providing tools and construction products.

Redis powers their order management system which is designed to process 30k transactions per second.

Sorted sets and Geodata structures are used for inventory forecasting and calculation in real-time, while hyperloglogs are used to count the number of unique customers, sales per item and sales per category.

Similarly, many other applications use Redis for Real-time analytics, like fraud or spam detection systems, because Redis is capable of reading and writing large amounts of data with low latencies.

As a message broker

This is when there is a channel or a group of users, where they can push or view messages, like a Whatsapp group or a Telegram Channel. This is a Pub/Sub model, where there are publishers and subscribers. Every time a new message is created or published, all the subscribers or listeners receive the message.

Redis supports this feature and is blazingly fast.

As a session store

Redis is also used to maintain and store the session of a user in an application.

Like in an e-commerce application, you would want the items in your cart to persist even when you close the site. It is also very common to store user metadata, like profiles and credentials, and Redis is the most popular choice in these scenarios.

When to avoid using Redis

Using Redis as the sole database

Since Redis is an in-memory key-value store, all the data must fit in the memory. It's storage size is dependent on how much RAM or primary memory is present in the system, which is much less in size and more costly when compared to hard disks.

So, it should not be used as the only database for very large applications.

Redis is used mainly with other RDBMS to supplement them by providing a caching mechanism.

Using Redis as a transactional server

Another pitfall of having storage in memory is security. Although Redis provides solutions for persistence, it is still not as secure as a real transactional server, which provides redo/undo logging, block checksumming, point-in-time recovery, flashback capabilities, etc.

Using Redis as a Relational Database

Redis is a data structure server, which unlike any RDBMS, does not provide a query language and there is no support for any relational algebra. So, the developer has to anticipate all the data accesses and needs to define proper data access paths, which means a lot of flexibility is lost.

Redis is more complex to scale

Redis instances must be deployed and started (forming clusters).

Alternatives to Redis

A major competitor of Redis in caching is Memcached, which provides multithreading.

MongoDB is another choice of NoSQL database, as it provides document-oriented storage with its own query language.

RabbitMQ is one of the most used message brokers, and is more durable and preferred over Redis, even though it's more complicated.

Cassandra is a partitioned row store, in which rows are organized into tables with a required primary key. This is highly used for its easy-to-scale capabilities, and for being more fault-tolerant.

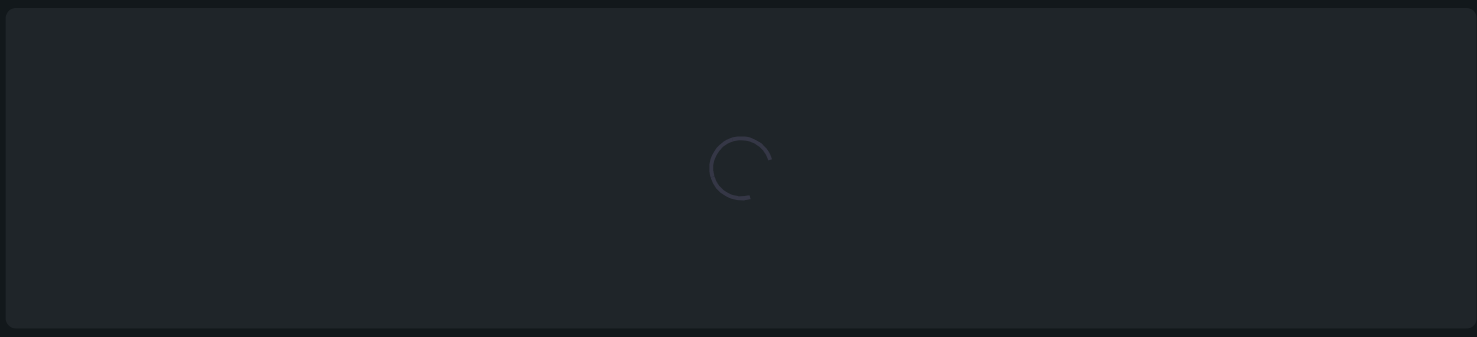
Final thoughts

As you can see, Redis is basically a swiss army knife and has multiple use cases in the real world. I hope that now you have a better knowledge of how these real-world applications use Redis, and what causes your feeds and gaming leaderboards to update in real-time.

Written by Jaidev Singh Bhui

MORE POST BY JAIDEV SINGH BHUI →

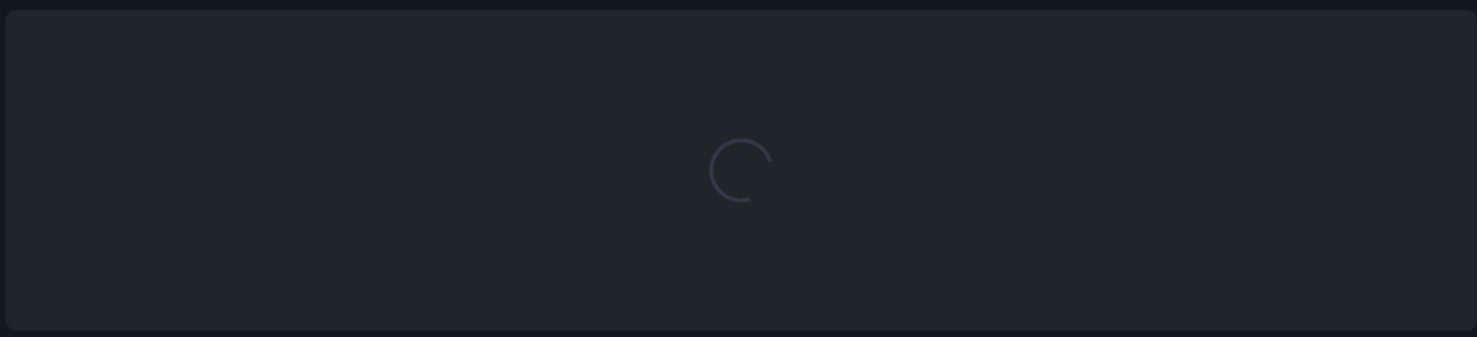
You might also like



DATABASES

Virtual Memory (Memory Leaks)

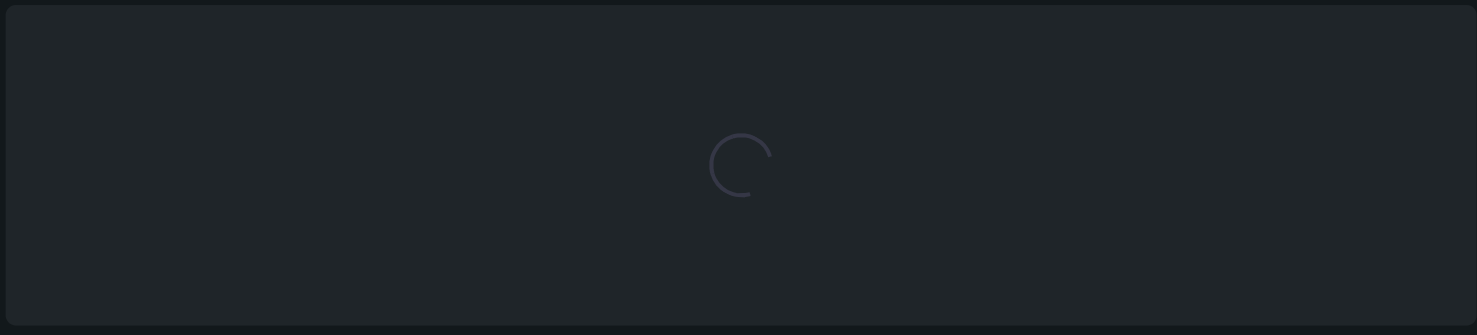
August 29, 2022 - 10 min read



DATABASES

Cache Memory Explained for Developers

July 23, 2021 - 14 min read



May 26, 2021 - 6 min read

← OLDER POST

Why choose MongoDB as your next database?

May 26, 2021 - 6 min read

NEWER POST →

Interview Experience with Grab

June 25, 2021 - 5 min read

What do you think?

35 Responses

 Upvote

 Love

 Enlightened

4 Comments

Login ▼

G

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

8 • Share

Best Newest Oldest

reaperwolverine

3 years ago

Very nice article. Very informative😁

You seem to be a Boston Celtics fan. Sorry for your team's bad season☹️

2 0 Reply

J

Jaidev Singh Bhui ↗ reaperwolverine

3 years ago

Haha yeah, tough season.

0 0 Reply

B

Birend kumar

10 months ago

good one very nice

0 0 Reply

J

Jack Richard

2 years ago

Hi there. Thanks for sharing the valuable information. I found this article very helpful. Would love to read more from you.
I have read a similar article on Top 7 Use Cases of Redis. Please check it out.

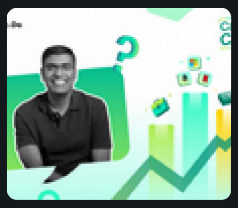
0 0 Reply

Billing info update failed.

LATEST POSTS

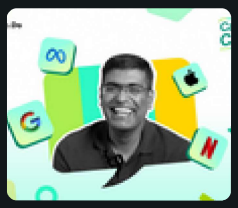
How to Switch from a non-IT to IT role? | Career Cafe

November 20, 2023 - 3 min read



Better salary, a bigger role or a prestigious company - What to prioritize? | Career Cafe

November 09, 2023 - 3 min read



Is it possible for anyone to get into MAANG Companies? | Career Cafe

November 06, 2023 - 3 min read

Career Cafe - Here's what's brewing at Crio.Do!

November 02, 2023 - 1 min read

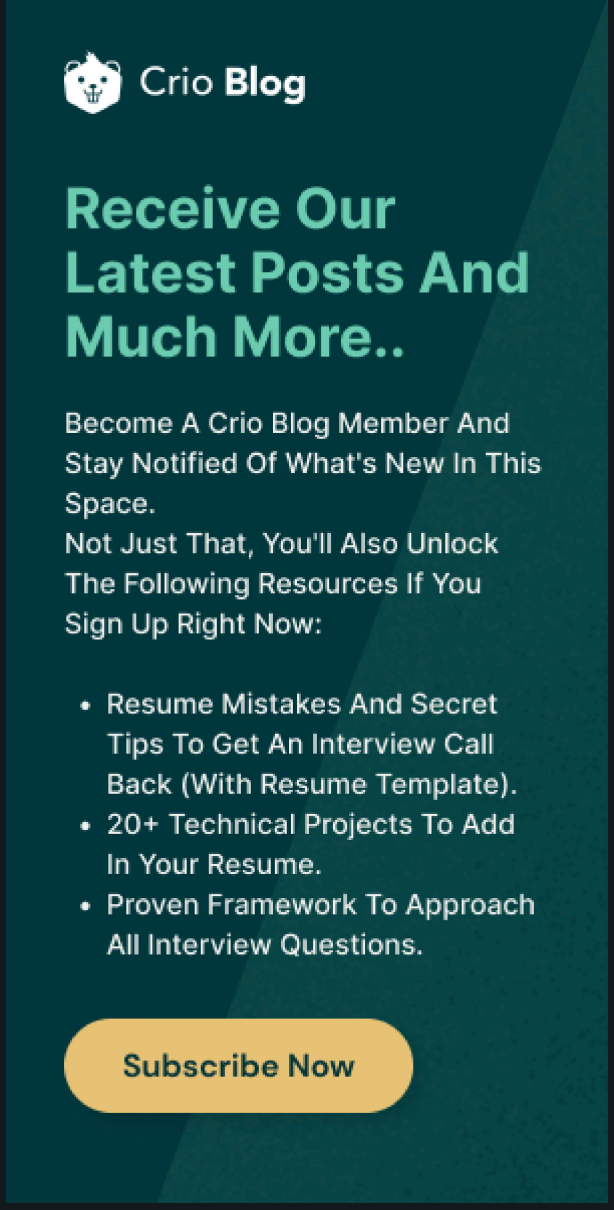
Empowering the Future: Celebrating Developers of India

August 11, 2023 - 2 min read

CATEGORIES

AUTOMATION TESTING | CAREER GUIDANCE | CRIO COMMUNITY | DATA STRUCTURES AND ALGORITHMS | DATABASES | DEVELOPER ESSENTIALS | IN THE NEWS | MINI PROJECTS | PROGRAMMING LANGUAGES | WEB DEVELOPMENT | WHY CRIO

Billing info update failed.



STAY CONNECTED

NAVIGATION

Mini Projects

Data Structures and Algorithms

Career Guidance

Developer Essentials

Web Development

Databases

Billing info update failed.

In the News

Why Crio

Subscribe to newsletter

Stay up to date! Get all the latest posts delivered straight to your inbox.

Your email address

Subscribe