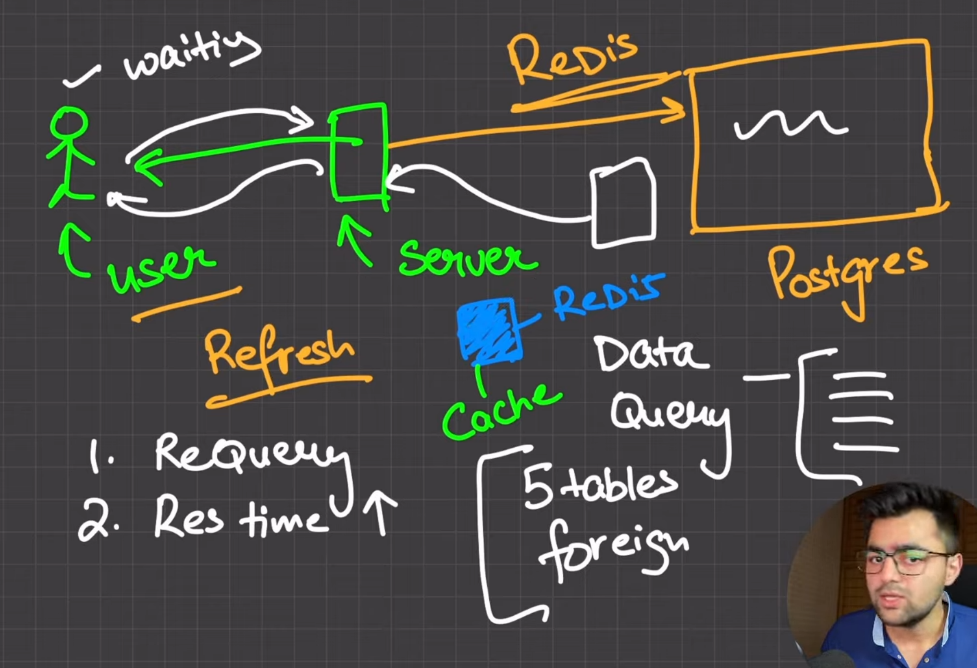
🔍 **What is REDIS**

**REDIS** (REmote DIctionary Server) is an open-source, in-memory data structure store used as a **database, cache, message broker**, and **streaming engine**. It is designed for high performance and is widely used to speed up applications by reducing the load on traditional databases.

* **In-memory**: Stores data in RAM for ultra-fast access.
* **Key-value store**: Similar to a dictionary or hash map.
* **Supports advanced data types**: Strings, lists, sets, sorted sets, hashes, bitmaps, geospatial indexes, and streams.
* **Persistent**: Can optionally save data to disk to recover from restarts.
* **Distributed**: Supports clustering and replication for scalability and availability.

**🚀 Why Redis Is Needed**

* Frequent queries to a db can be expensive, both in terms of performance and infrastructure cost.
* User waiting time also increases.
* So, we cache important data in Redis server.
* In real life scenario, Redis is mainly used to store computed data.



**📊What is computed data?**

**Computed data** is data that is **generated by performing calculations or processing on raw data**. It’s not just stored as-is — it’s derived or produced by applying some kind of operation, function, or logic.

**🧮Examples of computed data:**

1. **Sum of sales:**  
   Raw data: Individual sales transactions  
   Computed data: Total sales for the month (sum of all transactions)
2. **Average user rating:**  
   Raw data: User ratings from 1 to 5  
   Computed data: Average rating calculated from all the ratings
3. **Cache of API response:**  
   Raw data: Original data from a slow API  
   Computed data: The processed or formatted response saved for quick access
4. **Leaderboard ranking:**  
   Raw data: Players’ scores  
   Computed data: Sorted list of top players based on those scores

**🔍Why use computed data?**

* To save time: Instead of recalculating every time, you store the result.
* To improve performance: Fetching computed data is faster than recalculating.
* To simplify access: Your app can directly use the processed info.

**🚀Key Use Cases of REDIS:**

**1. Caching**

* **Why?** Speed up response time by caching database query results, API responses, or rendered HTML pages.
* **Example**: Store user sessions or recent search queries to avoid recalculating them.

**2. Real-Time Analytics**

* **Why?** Track metrics like page views or clicks with counters and sorted sets.
* **Example**: Dashboard showing real-time stats on user activity.

**3. Message Queues / Pub-Sub**

* **Why?** Handle asynchronous tasks or real-time messaging between services.
* **Example**: Redis Streams or Pub/Sub for chat apps, notification systems.

**4. Session Store**

* **Why?** Fast, temporary storage of user session data.
* **Example**: Store login tokens or shopping cart data.

**5. Rate Limiting**

* **Why?** Prevent abuse by tracking requests per user or IP.
* **Example**: "You can only try login 5 times in 10 minutes."

**6. Leaderboard / Ranking Systems**

* **Why?** Sorted sets make it easy to build leaderboards.
* **Example**: Gaming apps showing top player scores.

**⚙️ Why Choose Redis Over Traditional Databases?**

| **Feature** | **Redis** | **Traditional RDBMS** |
| --- | --- | --- |
| Speed | Extremely fast (in-memory) | Slower (disk-based) |
| Complexity | Simple key-value or structures | Complex queries and schemas |
| Use Case Focus | Caching, real-time | General-purpose data storage |

**🧠 In Summary:**

Redis is essential when you need low-latency access to data, want to reduce load on primary databases, or need a lightweight and fast solution for messaging, caching, or real-time analytics**.**

**🔧 What is Redis CLI?**

The **Redis CLI** (redis-cli) is a **command-line interface tool** that lets you **interact with a Redis server** directly from your terminal or shell

🔍 **What is Redis-Stack**

Redis Stack is an extended version of Redis that bundles the core Redis server with a set of popular and powerful Redis modules, along with tools for data visualization and search. It’s designed to make it easier for developers to build real-time, data-rich applications.

🔧 **What's Included in Redis Stack?**

Developer

Tools



Included Modules

Core Redis



| **Feature** | **Module / Capability** | **Description** |
| --- | --- | --- |
| **Core** | **Redis Core** | In-memory data store supporting strings, lists, sets, hashes, sorted sets, streams, pub/sub, transactions, and more. |
| **Search** | **Redisearch** | Full-text search, secondary indexing, filtering, and querying capabilities. |
| **JSON** | **RedisJSON** | Native JSON support with querying and updating JSON documents. |
| **Time Series** | **RedisTimeSeries** | Efficient handling of time-series data with downsampling, aggregation, etc. |
| **Graph** | **RedisGraph** | Property graph database for querying graph data using Cypher-like syntax. |
| **Probabilistic** | **RedisBloom** | Bloom filters, Count-Min Sketches, and HyperLogLogs for approximate data. |
| **Client Libraries** | **Official & community** | SDKs for languages like Python, Node.js, Java, Go, etc. |
| **UI Tool** | **RedisInsight** | GUI for interacting with Redis data, exploring modules, visualizing data. |

**🧑‍💻 Redis Stack is mostly used for GUI (visualization) feature by developers,**

**common use cases:**

* **Visualizing and exploring data:** Quickly inspect keys, values, JSON documents, time series, etc.
* **Debugging:** View data changes in real time, test queries (e.g., RediSearch, Graph).
* **Learning and prototyping:** Great for exploring modules without writing full client code.

📦**Redis Open Source implements following data types:**

| **Data Type** | **Description** | **Example Use Cases** |
| --- | --- | --- |
| **String** | Binary-safe strings (up to 512 MB). Can hold text, numbers, or serialized objects. | Caching HTML, storing tokens, counters |
| **JSON** *(via RedisJSON)* | Store, query, and manipulate JSON documents natively. *(Requires Redis Stack)* | Document storage, structured caching |
| **List (**queues and stack**)** | Ordered collection of strings. Supports push/pop operations from both ends (left/right). | Queues, task buffers, chat messages |
| **Set** | Unordered collection of unique strings. | Tags, unique visitors, common elements |
| **Hash** | Field-value pairs (like a dictionary or map). | User profiles, object storage |
| **Sorted Set (ZSet)** | Like Sets, but with an associated score for each element, sorted by score. | Leaderboards, ranking systems |
| **Stream** | Log-like data structure for time-ordered entries with IDs. | Event logs, message queues, telemetry |
| **Geo** | Store and query geospatial data (latitude, longitude). | Location-based features, geo-fencing |
| **Bitmaps** | Binary data manipulation using bits. | User feature flags, tracking daily activity |
| **Bitfields** | Perform bit-level operations with string values (grouped bits). | Efficient counters, compact data flags |
| **Probabilistic** *(via RedisBloom)* | Bloom filters, Count-Min Sketches, HyperLogLog. *(Requires Redis Stack)* | Unique counting, membership checks |
| **Time Series** *(via RedisTimeSeries)* | Time-stamped data with automatic aggregation and downsampling. *(Requires Redis Stack)* | IoT data, metrics, real-time analytics |

**(In Redis CLI)**

key

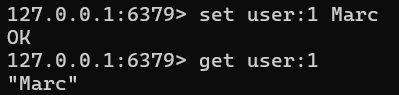
<https://redis.io/docs/latest/develop/data-types/>

1. **String**



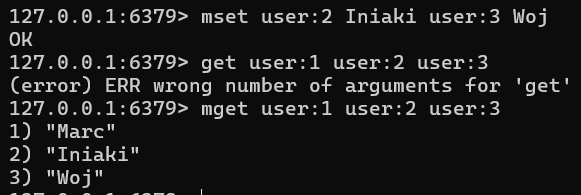
* Set <entity>:<id> value



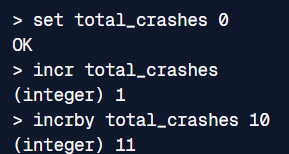
** **

* **To set multiple keys at single command**

mset and mget

****

* **String as counter**

****

**📝 Key Options Recap for SET:**

* **EX:** Expire in seconds.
* **PX:** Expire in milliseconds.
* **NX:** Set only if key doesn't exist.
* **XX:** Set only if key exists.
* **KEEPTTL:** Retain existing TTL**.**

**: TTL** stands for **Time To Live** — it’s the amount of time (in seconds) that a key will remain in Redis before it automatically expires (is deleted).

**Expiry** is necessary as if something stays in cache forever then it never gets updated to new value.

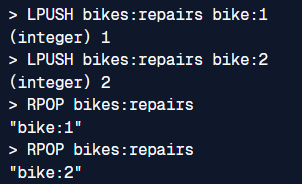
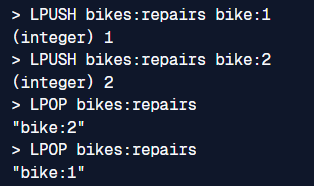
1. **List**

Redis list are implemented via Linked-List.

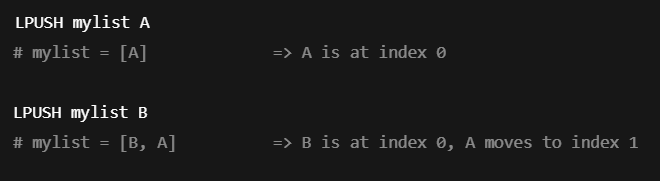
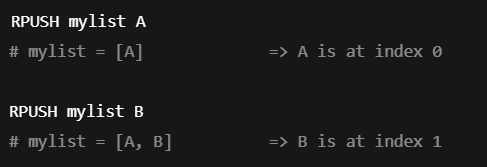
Redis lists are linked lists of string values. Redis lists are frequently used to:

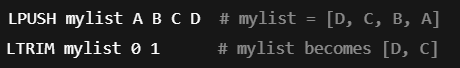
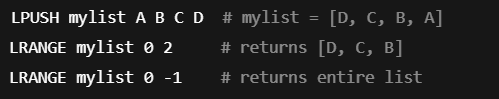
* Implement stacks and queues.
* Build queue management for background worker systems.

As Queue As Stack

* [LPUSH](https://redis.io/docs/latest/commands/lpush/) adds a new element to the head of a list;
* [RPUSH](https://redis.io/docs/latest/commands/rpush/) adds to the tail.
* [LPOP](https://redis.io/docs/latest/commands/lpop/) removes and returns an element from the head of a list;
* [RPOP](https://redis.io/docs/latest/commands/rpop/) does the same but from the tails of a list.
* [LLEN](https://redis.io/docs/latest/commands/llen/) returns the length of a list.
* [LMOVE](https://redis.io/docs/latest/commands/lmove/) atomically moves elements from one list to another.
* [LRANGE](https://redis.io/docs/latest/commands/lrange/) extracts a range of elements from a list.
* [LTRIM](https://redis.io/docs/latest/commands/ltrim/) reduces a list to the specified range of elements

**Blocking commands**

Lists support several blocking commands. For example:

* [BLPOP](https://redis.io/docs/latest/commands/blpop/) removes and returns an element from the head of a list. If the list is empty, the command blocks until an element becomes available or until the specified timeout is reached.
* [BLMOVE](https://redis.io/docs/latest/commands/blmove/) atomically moves elements from a source list to a target list. If the source list is empty, the command will block until a new element becomes available.



* If mylist has items, it returns the first item.
* If mylist is empty, it waits up to 5 seconds for an item to appear.
* **RETURNS:** A two-element array: [list\_name, element]

Or nil if the timeout expires.



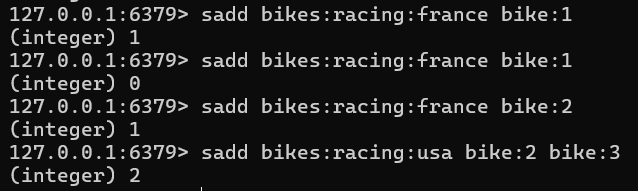
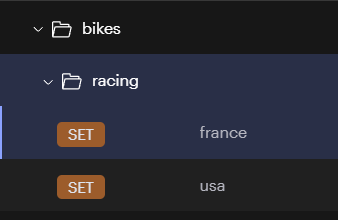
* Pops the leftmost element from queue1
* Pushes it to the right end of queue2
* Waits up to 10 seconds if queue1 is empty
* **RETURNS:** The moved element, or nil if the timeout expires.

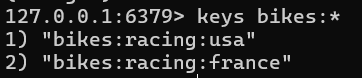
1. **Set**

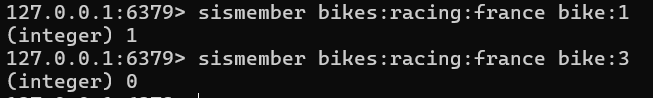
The max size of Redis Set is 2^32 – 1 (4,294,967,295) members.

A Redis set is an unordered collection of unique strings (members). You can use Redis sets to efficiently:

* Track unique items (e.g., track all unique IP addresses accessing a given blog post).
* Represent relations (e.g., the set of all users with a given role).
* Perform common set operations such as intersection, unions, and differences.
* [SADD](https://redis.io/docs/latest/commands/sadd/) adds a new member to a set.
* [SREM](https://redis.io/docs/latest/commands/srem/) removes the specified member from the set.
* [SISMEMBER](https://redis.io/docs/latest/commands/sismember/) tests a string for set membership.
* [SINTER](https://redis.io/docs/latest/commands/sinter/) returns the set of members that two or more sets have in common (i.e., the intersection).
* [SCARD](https://redis.io/docs/latest/commands/scard/) returns the size (a.k.a. cardinality) of a set.

** **

****

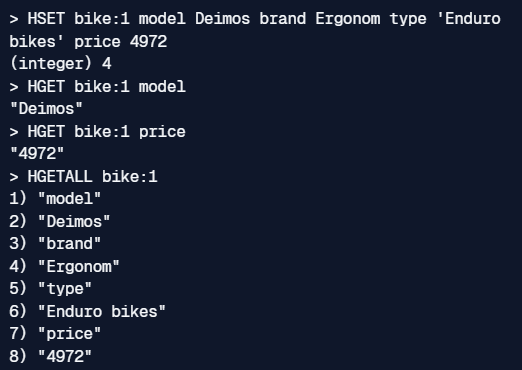
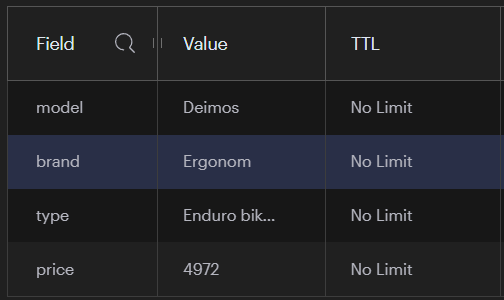
****

****

1. **Hash**

Redis hashes are record types structured as collections of field-value pairs. You can use hashes to represent basic objects and to store groupings of counters, among other things.

* Most Redis hash commands are O(1).
* A few commands, such as [HKEYS](https://redis.io/docs/latest/commands/hkeys/), [HVALS](https://redis.io/docs/latest/commands/hvals/), [HGETALL](https://redis.io/docs/latest/commands/hgetall/), and most of the expiration-related commands, are O(n), where *n* is the number of field-value pairs.
* Every hash can store up to 4,294,967,295 (2^32 - 1) field-value pairs

** **

1. **Stream**

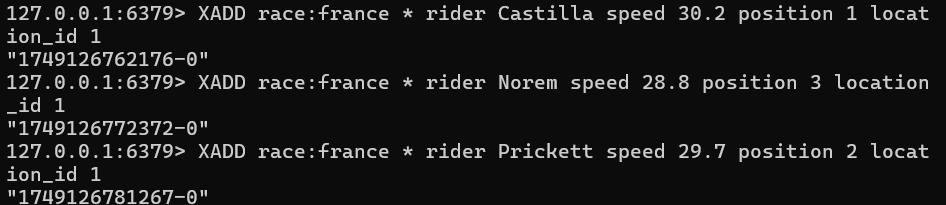
A Redis stream is a data structure that acts like an append-only log but also implements several operations to overcome some of the limits of a typical append-only log. These include random access in O(1) time and complex consumption strategies, such as consumer groups. You can use streams to record and simultaneously syndicate events in real time. Examples of Redis stream use cases include:

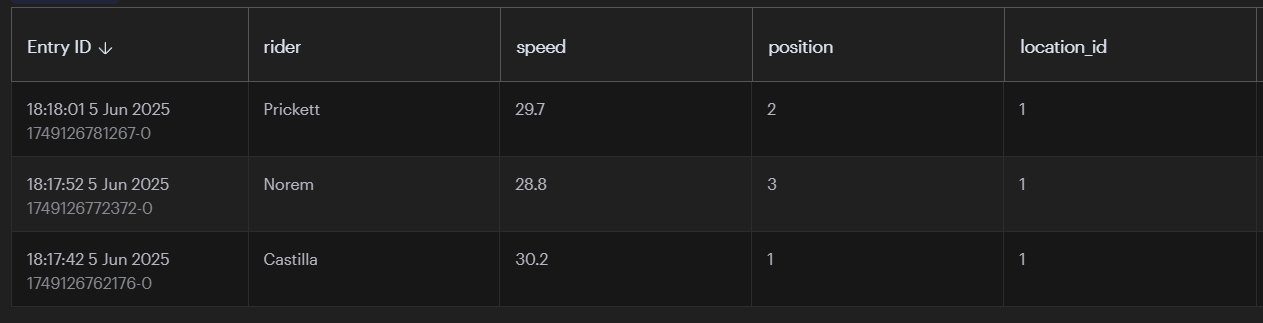
* Event sourcing (e.g., tracking user actions, clicks, etc.)
* Sensor monitoring (e.g., readings from devices in the field)
* Notifications (e.g., storing a record of each user's notifications in a separate stream)
* An **append-only log** is a data structure where **new entries are only added to the end** — nothing is overwritten or inserted in the middle. Older entries stay as-is, and the log grows over time.
* Adding an entry to a stream is O(1). Accessing any single entry is O(n), where *n* is the length of the ID.
* Each entry in a Redis stream has an **ID** like:

Its automatically generated by redis.

* The milliseconds time part is actually the local time in the local Redis node generating the stream ID
* The sequence number is used for entries created in the same millisecond.



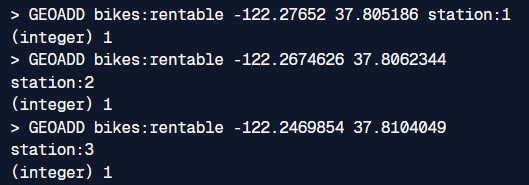


1. **Geospatial**

Redis geospatial indexes let you store coordinates and search for them. This data structure is useful for finding nearby points within a given radius or bounding box**.\**

**Basic commands**

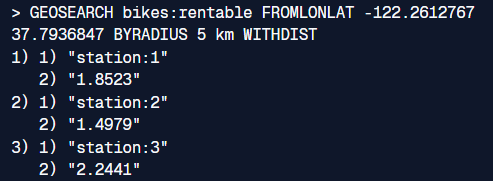
* [GEOADD](https://redis.io/docs/latest/commands/geoadd/) adds a location to a given geospatial index (note that longitude comes before latitude with this command).
* [GEOSEARCH](https://redis.io/docs/latest/commands/geosearch/) returns locations with a given radius or a bounding box.

****

Name of place



Find all locations within a 5 kilometer radius of a given location, and return the distance to each location:



**🔹Redus PUB/SUB**

Redis **Pub/Sub** (short for **Publish/Subscribe**) is a **messaging system** built into Redis that allows real-time communication between different parts of your application — without needing persistent storage.

It’s a **fire-and-forget** messaging model where:

* **Publishers** send messages to a **channel**
* **Subscribers** listen to that channel and receive messages in real-time

📌 There is **no storage** — if no subscriber is listening when a message is published, it is **lost**.

**🧭 Key Concepts**

| **Term** | **Description** |
| --- | --- |
| **Channel** | A named stream/message pipe (e.g., "chat-room-1") |
| **Publisher** | Sends messages to a channel using PUBLISH |
| **Subscriber** | Listens to channels using SUBSCRIBE |
| **Pattern Matching** | Use PSUBSCRIBE for wildcard subscriptions (e.g., news.\*) |

**🧪 Example**

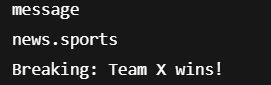
**1. Subscriber**



**2. Publisher**



The subscriber receives:



**✅ Use Cases**

| **Use Case** | **Why Use Pub/Sub?** |
| --- | --- |
| **Real-time chat** | Users receive messages instantly |
| **Notifications** | Push alerts to connected clients |
| **Game updates** | Sync state across players |
| **IoT systems** | Devices publish sensor data, apps consume it |

**🔗Redis on NODE.JS**

**Redis server** uses **port 6379** by default.

📦 **Redis Client for Node.js**

A **Redis client** is a software library or tool that allows applications to interact with a **Redis** database. The client acts as a bridge between your application and the Redis server.

* (check out code for more details)
* **redis** and **ioredis**—two popular Redis clients in the Node.js ecosystem

| **Feature** | **redis (aka node-redis)** | **ioredis** |
| --- | --- | --- |
| **Official Status** | **Official** Redis client for Node.js | **Community** maintained, very popular |
| **Cluster Support** | Basic support (v4+) | ✅ Fully supported & production-ready |
| **Sentinel Support** | Partial / limited | ✅ Fully supported |
| **Performance** | Fast, efficient | Slightly heavier but very robust |
| **Promises / Async** | ✅ Native async/await support (v4+) | ✅ Fully supports promises |
| **Pipelining** | ✅ Supported | ✅ Supported |
| **Pub/Sub** | ✅ Supported | ✅ Supported |
| **Typescript Support** | ✅ Good | ✅ Good |
| **Streams / Events** | Basic | ✅ Rich event system |
| **Multi-Redis** | ❌ | ✅ Easily connects to multiple Redis nodes |
| **Learning Curve** | Simpler, more minimal | Slightly more complex, but powerful |
| **Redis Modules Compatibility** | ✅ Native support for Redis Stack modules | ⚠️ Limited: only via raw call() commands |