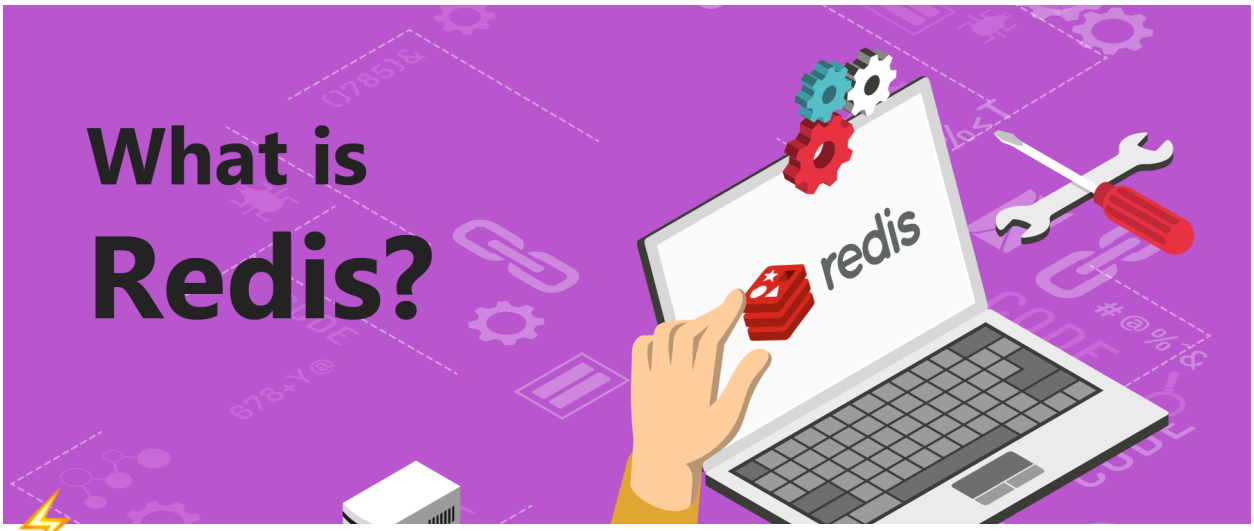



# Redis

## 📌 My personal accounts links

| | |
|---|---|
| 🔗 LinkedIn | https://www.linkedin.com/in/ahmed-hany-899a9a321?utm_source=share&utm_campaign=share_via&utm_content=profile&utm_medium=android_app |
| 🟢WhatsApp | https://wa.me/qr/7KNUQ7ZI3KO2N1 |
| 📘Facebook | https://www.facebook.com/share/1NFM1PfSjc/ |

### What is NoSQL?

**Answer:**

NoSQL is a type of database that stores data in a non-relational way. Unlike traditional SQL databases, NoSQL databases can store data as documents, key-value pairs, wide-columns, or graphs. They are often used for big data and real-time web applications because they are flexible and fast.

### What is SQL?

**Answer:**

SQL stands for Structured Query Language. It is used to store, manage, and retrieve data from relational databases. With SQL, you can create tables, insert data, update records, and search for information in a database

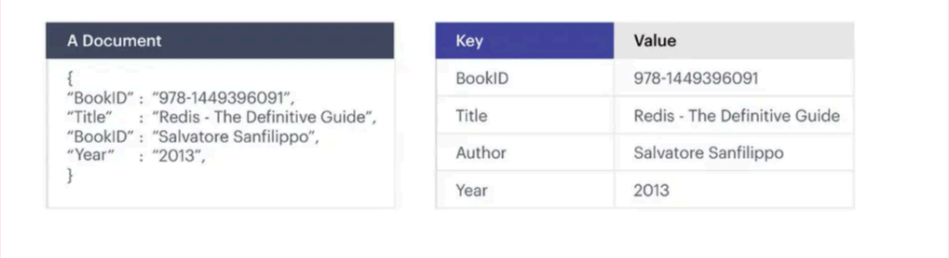
## What are the types of NoSQL databases?

**Answer:**

There are 4 main types of NoSQL databases:

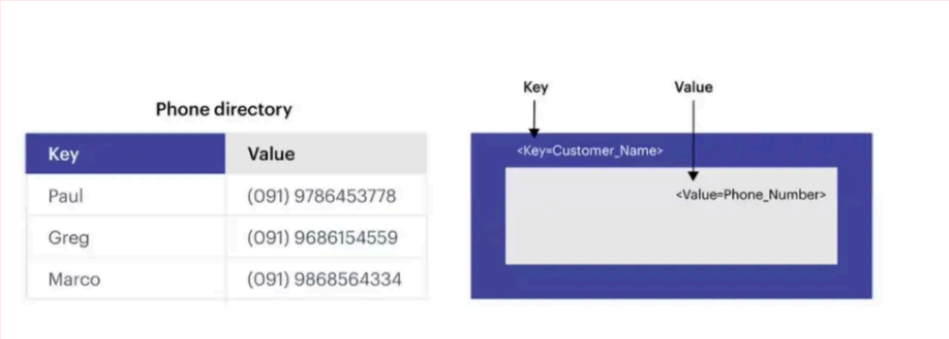
1. **Document Database**
   - Stores data as documents (like JSON).
   - **Example:** MongoDB
   - **Sample data:**

```
{ "name": "Ali", "age": 25 }
```




2. **Key-Value Store**
   - Stores data as key and value pairs.
   - **Example:** Redis
   - **Sample data:**

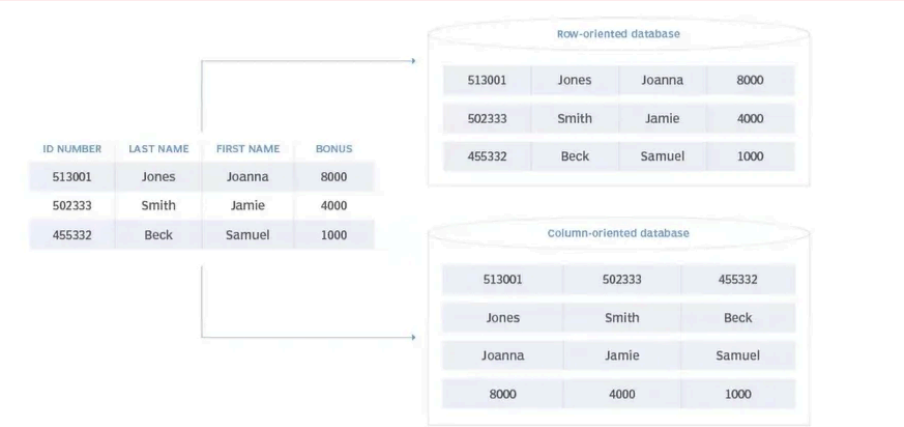
```
"name" → "Ali"
```




3. **Column-Based Store**
   - Stores data in columns instead of rows. Good for large datasets.
   - **Example:** Apache Cassandra

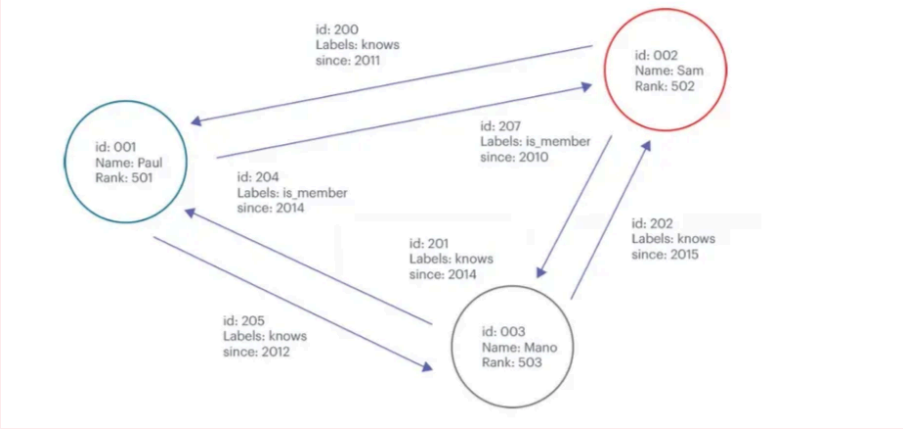
- **Sample data:**

```
Name | Age
-----|----
Ali  | 25
```




4. **Graph Database**
   - Stores data as nodes and relationships.
   - **Example:** Neo4j
   - **Sample data:**

```
(Ali)-[FRIEND]→(Sara)
```



## When should I use a NoSQL database?

**Answer:**

🥴 **Tired of complex table joins?** Use NoSQL — it's flexible and doesn't need a fixed schema!

😫 **Handling lots of unstructured data (like images, messages, or logs)?** NoSQL is your best friend!

🚀 **Need super fast performance for big data or real-time apps?** NoSQL flies high!

🔄 **Your data changes often, and you don't want to keep updating table structures?** NoSQL says: "No problem!"

🌍 **Building a scalable app for millions of users?** NoSQL is ready to grow with you!

## What is Redis and why should I use it?

**Answer:**

Redis is a **super fast**, **in-memory**, **key-value** NoSQL database. It stores data in **RAM**, not on disk — that means ⚡ **lightning speed**!

---

### Why use Redis?

🔥 **Need speed?** Redis is insanely fast. Your app will feel like it's on fire (in a good way)!

💾 **Want to cache data?** Redis keeps your most-used data ready — no need to hit the database again and again!

📊 **Handling real-time data like chats, leaderboards, or live notifications?** Redis is perfect — it's built for that kind of pressure! 💬🏆🔔

🔐 **Worried about losing data?** Redis can save snapshots or logs so your data stays safe 😌

## What types of data structures does Redis support?

**Answer:**

Redis supports the following data structures:

- **Strings** – for simple key-value pairs.
- **Lists** – ordered list of strings.
- **Sets** – unordered unique values.
- **Sorted Sets** – ordered by score.
- **Hashes** – like a dictionary (key-value inside a key).
- **Bit Arrays** – store bits (0 or 1).
- **HyperLogLogs** – estimate the number of unique items.
- **Streams** – handle real-time data.
- **Geospatial Indexes** – store and query location data.

## How can Redis help improve your application's performance and search capabilities?

**Answer:**

Redis helps in two key ways:

### 🔷 High Performance

- It can perform **more than 500,000 operations per second**.
- On **26 compute nodes**, Redis can reach **50 million operations per second**.

### 🔷 Search Features

- Supports **Autocomplete**, making search input faster.
- Enables **Result Highlighting**, improving user experience when displaying search results.

## 💡 How Redis Can Help You?

Redis is super fast and useful for:

### ✅ Real-Time Stuff

- Watching data live (like views or users)
- Catching fraud quickly

### 🎮 Games & Recommendations

- Keeping scoreboards updated
- Suggesting what users may like

### 📱 Social Apps

- Saving data fast (caching)
- Sending messages instantly (Pub/Sub)
- Handling jobs/tasks in order
- Checking user activity (analytics)
- Working with JSON easily

### 🔍 Search

- Finding things fast using RedisSearch

## ⚡ What is Caching?

- Users want websites and apps to **respond fast**.
- But giving a quick response all the time can use a lot of server power.
- To fix this, we use **caching** – it stores data temporarily so it can be shown faster next time.

## 🔍 What is Full-Text Search?

- It helps you **search inside text** (like documents) very fast.
- **RediSearch** is a tool that works with **Redis** to make this search faster and smarter.
- It finds the right results in **very little time**.

## 📍 What is a Geospatial & 📈 Time-Series Data (in Redis)?

- Redis supports **location-based data** (geospatial) and **data over time** (time-series).
- It uses special types like:
  ✅
  **Geospatial index**
  ✅
  **Hash**
  ✅
  **Sorted set**
  ✅
  **Stream**
- Useful for things like:
  - Finding nearby places
  - Tracking events over time
  - Giving location-based suggestions or offers

## 🤔 يعني إيه Geospatial & Time-Series Data في Redis؟

### 🔵 أولاً: Geospatial Data = بيانات مكانية (جغرافية)

يعني بيانات ليها علاقة بالموقع الجغرافي زي:

- إحداثيات (خط الطول ودوائر العرض)
- أماكن الناس أو المحلات أو السيارات على الخريطة

### 🧠 Redis بيساعدك تعمل إيه؟

- تسجل موقع كل محل أو عميل
- تقدر تسأل:
  👉 "هاتلي كل المحلات اللي في حدود 5 كيلومتر من موقعي"

🔧 Redis بيستخدم حاجة اسمها:
✅
**Geospatial Index** علشان يسجل ويبحث في المواقع بسرعة.

---

### ⏱️ ثانياً: Time-Series Data = بيانات بتمشي مع الوقت

يعني:

- بيانات بتتسجل بترتيب زمني
- زي: درجات الحرارة كل ساعة، أو عدد الزوار كل يوم، أو سرعة العربية كل دقيقة

### 🧠 Redis بيساعدك تعمل إيه؟

- تسجل الأحداث بوقتها
- ترجع تقرأ وتحلل البيانات دي
- تقدر تقول:
  👉 "وريني السرعة اللي كنت ماشي بيها من الساعة 5 للساعة 6"

🔧 Redis بيستخدم أنواع زي:
✅
**Stream** – لتسجيل الأحداث المتتابعة

✅ **Sorted Set** – لترتيب البيانات حسب الوقت

---

### 🧰 الأنواع اللي Redis بيستخدمها:

| النوع | بيعمل إيه |
|---|---|
| **Geospatial Index** | لتخزين الأماكن ومواقعها الجغرافية |
| **Hash** | لتخزين خصائص أي كيان (مثلاً اسم، عنوان، حالة) |
| **Sorted Set** | بيخزن بيانات بترتيب زمني |
| **Stream** | لتسجيل أحداث بالتاريخ والوقت |

---

### 🎯 أمثلة من الحياة:

- تطبيق توصيل: يعرف أقرب سائق للعميل (Geospatial)
- تطبيق طقس: يسجل درجات الحرارة كل ساعة (Time-Series)
- تطبيق رياضي: يتابع خطواتك يوميًا (Time-Series)

## 📩What is  Messaging / Queuing ?

Redis can be used to handle **fast-moving data**.

🔷 **What does that mean?**

Imagine you have **millions of sensors** sending data continuously. That data needs to be **collected, processed, and analyzed** quickly.

🔷 **What can Redis do?**

Redis can:

- **Collect** data quickly
- **Stream** it to other systems
- **Process** it in real-time

🔷 **How does Redis do that?**

Using built-in features like:

- **Pub/Sub (Publish/Subscribe)**: One part publishes data, another subscribes to receive it.
- **Streams**: Like a continuous flow of data that Redis can manage and process.

---

So in simple words:

Redis is great when you have **lots of data coming in very fast**, and you need to handle or transfer it **immediately**.

### give a simple commands to set key & value in redis ?

```
# redis-cli
127.0.0.1:6379> set x:y:a 10
OK
127.0.0.1:6379> get x:y:a
"10"
127.0.0.1:6379>
```

Example With C# :

To work with **Redis** in a **.NET** application, the most commonly used and recommended package is:

📦 `StackExchange.Redis`

This is the official and actively maintained client library for Redis in .NET.

---

### ✅ To install it:

### Using NuGet Package Manager Console:

```
Install-Package StackExchange.Redis
```

Code:

```
using StackExchange.Redis;

namespace Redis_tester
{
   internal class Program
   {
      static void Main(string[] args)
      {
         var redis = ConnectionMultiplexer.Connect("localhost");
         IDatabase db=redis.GetDatabase();

         db.StringSet("name", "ahmed");
         db.StringSet("x:y:a", "test");

         var v1=db.StringGet("name");
         var v2 = db.StringGet("x:y:a");

         Console.WriteLine($"key[name]  ⇒ value[{v1}] \n" +
            $" key[x:y:a]  ⇒ value[{v2}]  ");
      }
   }
}
```

output:

```
key[name]  ⇒ value[ahmed]
 key[x:y:a]  ⇒ value[test]
```

## 🔢 What is  Redis Data Structure - String ?

In Redis, **data is stored using keys**. These keys can be anything, like strings, numbers—even images—because Redis is **binary-safe**. But most of the time, we use simple strings.

Here are some common commands you can use with strings:

---

### 📌 Basic Commands:

1. `SET user "ahmed"`
   ➤ Creates a key called `user` and sets its value to `"ahmed"`.
2. `SET user "mohamed"`
   ➤ Updates the value of `user` to `"mohamed"` (overwrites the old value).
3. `GET user`
   ➤ Returns `"mohamed"`.

---

### 🔢 Working with Numbers:

1. `SET count 20`
   ➤ Creates a key `count` with value 20.
2. `INCR count`
   ➤ Increases the `count` value by 1 → becomes 21.
3. `INCRBY count 5`
   ➤ Increases `count` by 5 → becomes 26.

---

### 🔄 More Useful Commands:

1. `GETSET name "ahmed"`
   ➤ Sets `name` to `"ahmed"` and returns the **old value** (before setting).
2. `MSET key1 value1 key2 value2 key3 value3`
   ➤ Sets multiple keys and values at once.
3. `MGET key1 key2 key3`
   ➤ Gets values of multiple keys at once.
4. `GETRANGE key 0 5`
   ➤ Returns a part of the string value from index 0 to 5.

---

```
127.0.0.1:6379> set user 'ahmed'
OK
127.0.0.1:6379> get user
"ahmed"
127.0.0.1:6379> set user 'omar'
OK
127.0.0.1:6379> get user
"omar"
127.0.0.1:6379> set count 5
OK
127.0.0.1:6379> get count
"5"
```

```
127.0.0.1:6379> incr count
(integer) 6
127.0.0.1:6379> incrby count 4
(integer) 10
127.0.0.1:6379> getset user 'said'
"omar"
127.0.0.1:6379> get user
"said"
127.0.0.1:6379> mset k1 5 k2 'kamel' k3 13
OK
127.0.0.1:6379> mget k1 k2 k3 user count
1) "5"
2) "kamel"
3) "13"
4) "said"
5) "10"
127.0.0.1:6379> getrange k2 0 2
"kam"
127.0.0.1:6379>
```

Code with C# :

```
        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost");
            IDatabase db=redis.GetDatabase();

            //Basic Commands:
            db.StringSet("user", "ahmed");
            var v1=db.StringGet("user");
            Console.WriteLine($"user => {v1} \n after change : ");
            db.StringSet("user", "ali");
            var v2=db.StringGet("user");
            Console.WriteLine($"user => {v2}");

            //Working with Numbers:

            db.StringSet("count", 5);
            var c1=db.StringGet("count");
            Console.WriteLine($"count = {c1}");

            Console.WriteLine("after increment");
            db.StringIncrement("count");
            var c2 = db.StringGet("count");
            Console.WriteLine($"count = {c2}");

            Console.WriteLine("after incrementby");
            db.StringIncrement("count",4);
            var c3 = db.StringGet("count");
            Console.WriteLine($"count = {c3}");

            // More Useful Commands:
            //user = ali
            var u1 =db.StringGetSet("user", "kamel");
```

```
            var u2 = db.StringGet("user");//user = kamel

            Console.WriteLine($"usr ⇒ old value :{u1} & new value :  {u2}");

            //mset , mget
            db.StringSet(new []
            {
               new KeyValuePair<RedisKey, RedisValue> ("k1","v1"),
               new KeyValuePair<RedisKey, RedisValue> ("k2","v2"),
               new KeyValuePair<RedisKey, RedisValue> ("k3","v3")

            });


            var values = db.StringGet(new RedisKey[] { "k1", "k2", "k3" });

            foreach (var v in values)
            {
               Console.WriteLine(v);
            }

            //getrange

            var userrange = db.StringGetRange("user", 0, 2);
            Console.WriteLine($"user ⇒ {u2} after getrange 0 > 2 : {userrange}");
        }
```

output:

```
user ⇒ ahmed
 after change :
user ⇒ ali
count = 5
after increment
count = 6
after incrementby
count = 10
usr ⇒ old value :ali & new value :  kamel
v1
v2
v3
user ⇒ kamel after getrange 0 > 2 : kam
```

## ❓ Question:

**You want to store the string `"dog"` as a JSON value in Redis under the key `animal`, then retrieve it, check its type, and see how many characters it contains.**

### Which RedisJSON commands would you use, and what do they do?

## ✅ Answer and Explanation:

### 🔷 1. `json.set animal $ '"dog"'`

**Explanation:**

This command **sets a JSON value** at the root (`$`) of the key `animal` to the string `"dog"`.

- The outer quotes are for the Redis CLI.
- The inner `'\"dog\"'` makes sure it is stored as a valid JSON string.

🧠 Think of this as:

```
animal: "dog"
```

### 🔷 2. `json.get animal $`

**Explanation:**

This command **retrieves** the JSON value stored at the root path (`$`) of the key `animal`.

📤 It will return:

```
["dog"]
```

### 🔷 3. `json.type animal $`

**Explanation:**

This command tells you the **type of the JSON value** at the specified path.

📘 Expected output:

```
string
```

### 🔷 4. `json.strlen animal $`

**Explanation:**

This command gives the **string length** of the JSON string at the given path.

🐶 For `"dog"`, it returns:

```
3
```

```
127.0.0.1:6379> json.set animal $ '"dog"'
OK
127.0.0.1:6379> json.get animal $
"[\"dog\"]"
127.0.0.1:6379> json.type animal $
```

```
1) "string"
127.0.0.1:6379> json.strlen animal $
1) (integer) 3
```

install package : `NRedisStack`

C# code :

```
using NRedisStack;
using NRedisStack.RedisStackCommands;
using StackExchange.Redis;
using System.Text.Json;

        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost");
            IDatabase db = redis.GetDatabase();

            IJsonCommands json = db.JSON();
            json.Set("animal", "$", JsonSerializer.Serialize("dog"));

            var val = json.Get("animal");
            var type = json.Type("animal").FirstOrDefault();
            var len = json.StrLen("animal").FirstOrDefault();

            Console.WriteLine($"value => {val}\n type : {type} \n length : {len}");
        }
```

output:

```
value => "dog"
 type : STRING
 length : 3
```

## ✅ What is a Redis List?

- A **linked list** data structure.
- It is stored under **a single key** that holds **multiple ordered values**.

### ✅ Key Features:

1. **Values are stored as strings.**
2. You can **add values at either end**:
   - Left → also called **Head**.
   - Right → also called **Tail**.
3. You can **retrieve values by index** (position in the list).
4. **Duplicate values are allowed** within the list.
5. It can function as either:
   - A **Stack** (Last In, First Out — LIFO).
   - A **Queue** (First In, First Out — FIFO).

---

- The list stores values from **left to right**, like this: `Value5 → Value4 → Value3 → Value2 → Value1`
- Indexes go from `0` at the **left/head** to `4` at the **right/tail**.
- Commands you can use:
  - `LPUSH` → adds a value to the **left** (head).
  - `RPUSH` → adds a value to the **right** (tail).
  - `LINDEX` → gets the value at a specific index.
  - `LPOP` / `RPOP` → remove values from left or right.

---

Commands

```
127.0.0.1:6379> lpush users u1 u2 u3
(integer) 3
127.0.0.1:6379> lindex users 0
"u3"
127.0.0.1:6379> lindex users 2
"u1"
127.0.0.1:6379> rpush users u4
(integer) 4
127.0.0.1:6379> lindex users 3
"u4"
127.0.0.1:6379> lpop users
"u3"
127.0.0.1:6379> lpop users 2
1) "u2"
2) "u1"
127.0.0.1:6379> lpush users u1 u2 u3
(integer) 4
127.0.0.1:6379> rpop users 2
1) "u4"
```

```
2) "u1"
127.0.0.1:6379>
```

`llen` : get count of elements in list

`lmove` :move element from list to another

`ltrim` :cut some elements from left

`lrange` :desplay elements of list

```
# redis-cli
127.0.0.1:6379> lpush list1 u1 u2 u3
(integer) 3
127.0.0.1:6379> llen list1
(integer) 3
127.0.0.1:6379> lpush list2 a1 a2 a3
(integer) 3
127.0.0.1:6379> lmove list1 list2 right left
"u1"
127.0.0.1:6379> lindex list2 0
"u1"
127.0.0.1:6379> lmove list2 list1 left right
"u1"
127.0.0.1:6379> lindex list1 2
"u1"
127.0.0.1:6379> lindex list2 0
"a3"
127.0.0.1:6379> lpush nums 1 2 3 4 5 6 7 8 9 10
(integer) 10
127.0.0.1:6379> ltrim nums 0 4
OK
127.0.0.1:6379> lrange nums 0 -1
1) "10"
2) "9"
3) "8"
4) "7"
5) "6"
127.0.0.1:6379> lrange nums -1 0
(empty array)
127.0.0.1:6379>
```

C# Code :

```
    static void Main(string[] args)
    {
        var redis = ConnectionMultiplexer.Connect("localhost");
        IDatabase db = redis.GetDatabase();

        db.ListLeftPush("users", new RedisValue[]
        {
            "u1","u2","u3"
        });
```

```
        var u3 = db.ListGetByIndex("users",0);
        var u1 = db.ListGetByIndex("users", 2);
        var u2 = db.ListGetByIndex("users", 1);



        db.ListRightPush("users", new RedisValue[]
        {
           "u4"
        });
        var u4 = db.ListGetByIndex("users", 3);

        RedisValue[] values = new RedisValue[] { u3, u2, u1, u4 };
        Console.WriteLine("values");
        foreach (var item in values)
        {
           Console.Write(item+" ");
        }
        Console.WriteLine();

        var popleft=db.ListLeftPop("users");
        var popright=db.ListRightPop("users");

        Console.WriteLine("popleft : "+popleft);
        Console.WriteLine("popright : " + popright);


    }
```

> output
>
> values
> u3 u2 u1 u4
> popleft : u3
> popright : u4

C# Code

```
        static void Main(string[] args)
        {
           var redis = ConnectionMultiplexer.Connect("localhost");
           IDatabase db = redis.GetDatabase();

           db.ListLeftPush("list1", new RedisValue[]
           {
              "u1","u2","u3"
           });
```

```
        db.ListLeftPush("list2", new RedisValue[]
        {
            "a1","a2","a3"
        });

        //len
        Console.WriteLine("count of elements in list1 : " + db.ListLength("list1"));
        Console.WriteLine("count of elements in list2 : " + db.ListLength("list2"));

        //lmove
        db.ListMove("list1", "list2", ListSide.Right, ListSide.Left);

        //range
        RedisValue[] valueslist1 = db.ListRange("list1", 0, -1);
        RedisValue[] valueslist2 = db.ListRange("list2", 0, -1);

        Console.WriteLine("list1 after move : ");
        foreach (var item in valueslist1)
        {
            Console.Write(item + " ");
        }


        Console.WriteLine("\nlist2 after move : ");
        foreach (var item in valueslist2)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine();
        db.ListLeftPush("nums", new RedisValue[]
        {
            10,9,8,7,6,5,4,3,2,1
        });

        db.ListTrim("nums", 0, 4);

        RedisValue[] TrimNums = db.ListRange("nums", 0, -1);
        Console.WriteLine("trim [1,2,3,4,5,6,7,8,9,10] from 0 to -1 ");
        foreach (var item in TrimNums)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine();
    }
```

> output
>
> count of elements in list1 : 3
> count of elements in list2 : 3
> list1 after move :
> u3 u2

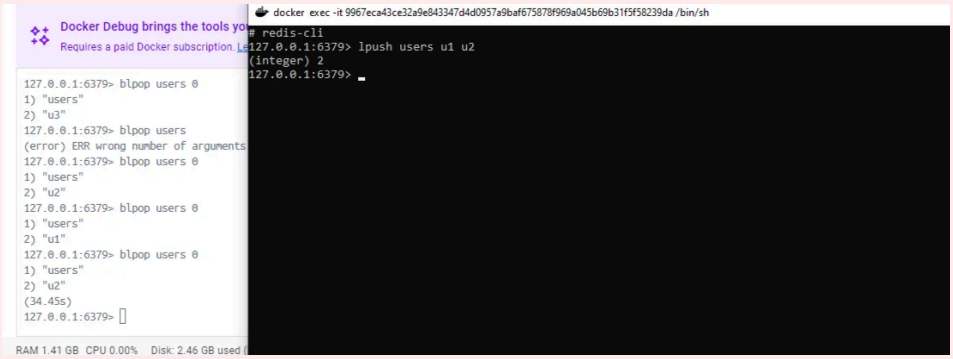
> list2 after move :
> u1 a3 a2 a1
> trim [1,2,3,4,5,6,7,8,9,10] from 0 to -1
> 1 2 3 4 5

### BLPOP (Blocking Left Pop)

**Definition:**

`BLPOP` removes and returns the **first element** (leftmost) of a list. If the list is empty, it **waits (blocks)** until an element is available or a timeout occurs.

**Use case:** Useful when you want to wait for items to arrive in a list (like a message queue).




### BLMOVE (Blocking Move)

**Definition:**

`BLMOVE` moves an element from one list to another, **blocking** until an element is available in the source list. You can specify whether to take from the left or right of the source list, and whether to insert to the left or right of the destination list.

**Use case:** Combines the behavior of popping from one list and pushing to another with blocking support.

## Redis Data Structure – List Notes:

### 🔶 Automatic creation and removal of keys

- ➤ When we add an element to an aggregate data type, if the target key does not exist, an empty aggregate data type is created before adding the element.
- ➤ When we remove elements from an aggregate data type, if the value becomes empty, the key is automatically deleted.

  **The Stream data type is the only exception to this rule.**

## 🔷 Redis Data Structure – Sets

### ✅ What is a Redis Set?

- A **Redis Set** is an **unordered collection** of **unique strings** (called members).
- It does **not allow duplicate values**.
- Sets are great for operations like tracking unique items, tags, user IDs, etc.

### 🔧 Useful Redis Set Commands:

| Command | Description |
|---|---|
| **SADD** | Add one or more members to a set. |
| **SREM** | Remove one or more members from a set. |
| **SISMEMBER** | Check if a specific value exists in the set (returns 1 or 0). |
| **SMISMEMBER** | Check if multiple values exist in the set (returns a list of 1s and 0s). |
| **SINTER** | Return the **intersection** (common elements) between sets. |
| **SDIFF** | Return the **difference** between sets (elements in the first set that aren't in the others). |
| **SCARD** | Return the **cardinality** (number of elements) of the set. |
| **SMEMBERS** | Return **all members** of the set. |
| **SUNION** | Return the **union** of multiple sets (all unique members from all sets). |
| **SPOP** | **Remove and return** a random element from the set. |
| **SRANDMEMBER** | **Return (without removing)** a random element from the set. |

Commands:

```
# redis-cli
127.0.0.1:6379> sadd users u1 u2 u3 u4 u3
(integer) 4
127.0.0.1:6379> srem users u4
(integer) 1
127.0.0.1:6379> sismember users u2
(integer) 1
127.0.0.1:6379> sismember users u4
(integer) 0
127.0.0.1:6379> smismember u1 u2 u3 u4
1) (integer) 0
2) (integer) 0
3) (integer) 0
127.0.0.1:6379> smismember users  u1 u2 u3 u4
1) (integer) 1
2) (integer) 1
3) (integer) 1
4) (integer) 0
127.0.0.1:6379> sadd users2 u6 u1 u2 u8
(integer) 4
127.0.0.1:6379> sinter users users2
1) "u1"
2) "u2"
127.0.0.1:6379> sdiff users users2
```

```
1) "u3"
127.0.0.1:6379> scard users
(integer) 3
127.0.0.1:6379> scard users2
(integer) 4
127.0.0.1:6379> smembers users
1) "u1"
2) "u2"
3) "u3"
127.0.0.1:6379> sunion users users2
1) "u1"
2) "u2"
3) "u3"
4) "u6"
5) "u8"
127.0.0.1:6379> spop users2 2
1) "u6"
2) "u2"
127.0.0.1:6379> srandmember users
"u2"
127.0.0.1:6379> srandmember users
"u2"
127.0.0.1:6379> srandmember users
"u3"
127.0.0.1:6379>
```

C# Code:

```
        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
            var db = redis.GetDatabase();

            db.SetAdd("set1", new RedisValue[]
            {
                new RedisValue("u1"),
                new RedisValue("u2"),
                new RedisValue("u3"),
                new RedisValue("u2")

            });

            RedisValue[] members = db.SetMembers("set1");
            Console.WriteLine("unique members of set : ");
            foreach (var item in members)
            {
                Console.Write(item+" ");
            }
            Console.WriteLine();

            Console.WriteLine($"check if u2 is member : {db.SetContains("set1","u2")}");
            Console.WriteLine($"randam member : {db.SetRandomMember("set1")} ");
            Console.WriteLine($"remove randam member : {db.SetPop("set1")} ");
            Console.WriteLine($"count of members in set1 {db.SetLength("set1")}");
```

```
        db.SetAdd("set2", new RedisValue[]
        {
            new RedisValue("u8"),
            new RedisValue("u4"),
            new RedisValue("u3"),
            new RedisValue("u2")

        });
        RedisValue[] members2 = db.SetMembers("set2");
        Console.WriteLine("unique members of set : ");
        foreach (var item in members2)
        {
            Console.Write(item + " ");
        }
        var diffsets = db.SetCombine(SetOperation.Difference, "set1", "set2");
        var union = db.SetCombine(SetOperation.Union, "set1", "set2");
        var intersect = db.SetCombine(SetOperation.Intersect, "set1", "set2");

        Console.WriteLine("\ndefference");
        foreach (var item in diffsets)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine("\nunion");

        foreach (var item in union)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine("\nintersect");

        foreach (var item in intersect)
        {
            Console.Write(item + " ");
        }

    }
```

> Output:
>
> unique members of set :
> u1 u2 u3
> check if u2 is member : True
> randam member : u1
> remove randam member : u2
> count of members in set1 2
> unique members of set :
> u8 u4 u3 u2

> defference
> u1
> union
> u1 u3 u8 u4 u2
> intersect
> u3

## 🔷 Redis Data Structure – Hashes

### ✅ What is a Redis Hash?

- A **Redis Hash** is like a **record** or **object**, made up of **field-value pairs**.
- It's perfect for storing small sets of data about an object, like a **user profile**.

### 👤 Example:

| Key | FirstName | LastName | Age | Phone |
|---|---|---|---|---|
| User:1 | Ahmed | Ali | 25 | (00)000000 |
| User:2 | Mohamed | Khaled | 30 | (00)000000 |

### 🔧 Useful Redis Hash Commands:

| Command | Description |
|---|---|
| **HSET** | Set the value of a field in a hash. Example: `HSET User:1 FirstName "Ahmed"` |
| **HGET** | Get the value of a single field. Example: `HGET User:1 Age` |
| **HMGET** | Get the values of multiple fields. Example: `HMGET User:1 FirstName LastName` |
| **HGETALL** | Get **all fields and values** in a hash. |
| **HINCRBY** | Increase the value of a numeric field by a number. Example: `HINCRBY User:1 Age 1` |

Commands:

```
127.0.0.1:6379> hset hash1 name 'ahmed' age 20
(integer) 2
127.0.0.1:6379> hset hash2 name 'omar' age 22
(integer) 2
127.0.0.1:6379> hget hash1 name
"ahmed"
127.0.0.1:6379> hmget hash2 name age
1) "omar"
2) "22"
127.0.0.1:6379> hgetall hash1
1) "name"
2) "ahmed"
3) "age"
4) "20"
127.0.0.1:6379> hincrby hash1 age
(error) ERR wrong number of arguments for 'hincrby' command
127.0.0.1:6379> hincrby hash1 age 1
(integer) 21
127.0.0.1:6379>
```

Code C#

```
        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
            var db = redis.GetDatabase();
```

```
        db.HashSet("hash1", new HashEntry[]
        {
          new HashEntry("name","ahmed"),
          new HashEntry("age",20)
        });

        RedisValue hashvalue = db.HashGet("hash1", "name");
        HashEntry[] allhashvalues = db.HashGetAll("hash1");

        RedisValue[] hashmget = db.HashGet("hash1", new RedisValue[]
        {
           "name","age"
        });

        Console.WriteLine($"name : {hashvalue}");
        Console.WriteLine("all entries and values");
        foreach (var item in allhashvalues)
        {
           Console.WriteLine($"{item} ⇒ {item.Value}");
        }

        Console.WriteLine("all values");
        foreach (var item in hashmget)
        {
           Console.WriteLine(item);
        }


    }
```

> output:
>
> name : ahmed
> all entries and values
> name: ahmed ⇒ ahmed
> age: 20 ⇒ 20
> all values
> ahmed
> 20

## 🔷 What is a Redis Sorted Set ?

- A **Sorted Set** in Redis is a set of **unique strings (members)**.
- Each member is **associated with a score** (a `double`).
- Redis **orders the set by score**, so you can do things like: top-N, ranges, rank, etc.

## 🔶 Sorted Set Commands (from the slide)

| Command | Description |
|---|---|
| `ZADD` | Add member(s) with score(s) to a sorted set. |
| `ZRANGE` | Get members in a range (by index, low to high). |
| `ZREVRANGE` | Get members in a range (by index, high to low). |
| `ZRANGEBYSCORE` | Get members in a range (by score). |
| `ZREM` | Remove a member. |
| `ZREMRANGEBYSCORE` | Remove members by score range. |
| `ZRANK` | Get the rank (index) of a member (low to high). |
| `ZREVRANK` | Get the rank (index) of a member (high to low). |
| `ZRANGEBYLEX` | Get members in a lexicographical range (ordered by string, not score). |

## Commands

```
# redis-cli
127.0.0.1:6379> zadd players 1 "ahmed" 2 "omar" 6 "said" 4 "ali"
(integer) 4
127.0.0.1:6379> zrange players 0 -1 withscores
1) "ahmed"
2) "1"
3) "omar"
4) "2"
5) "ali"
6) "4"
7) "said"
8) "6"
127.0.0.1:6379> zrevrange players -1 0 withscores
(empty array)
127.0.0.1:6379> zrevrange players 0 -1 withscores
1) "said"
2) "6"
3) "ali"
4) "4"
5) "omar"
6) "2"
7) "ahmed"
8) "1"
127.0.0.1:6379> zrangebyscore players 1 5
1) "ahmed"
2) "omar"
3) "ali"
127.0.0.1:6379> zrank players "said" withscore
1) (integer) 3
```

```
2) "6"
127.0.0.1:6379> zrevrank players "said"
(integer) 0
127.0.0.1:6379> zrevrank players "ahmed"
(integer) 3
127.0.0.1:6379> zrangebylex players [a [p
1) "ahmed"
2) "omar"
3) "ali"
127.0.0.1:6379> zrem players "said"
(integer) 1
127.0.0.1:6379> zrange players 0 -1
1) "ahmed"
2) "omar"
3) "ali"
127.0.0.1:6379> zremrangebyscore players 1 3
(integer) 2
127.0.0.1:6379> zrange players 0 -1
1) "ali"
127.0.0.1:6379>
```

Code C# :

```
    static void Main(string[] args)
    {
        var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
        var db = redis.GetDatabase();


        db.SortedSetAdd("players", new SortedSetEntry[]
        {
            new SortedSetEntry("ahmed",1),
            new SortedSetEntry("omar",2),
            new SortedSetEntry("yasser",8),
            new SortedSetEntry("said",5),

        });

        SortedSetEntry[] range_players = db.SortedSetRangeByRankWithScores("players", start: 0, stop: -1);
        Console.WriteLine("rank ⇒ value");
        foreach (var item in range_players)
        {
            Console.WriteLine($"{item.Element} ⇒ {item.Score}");
        }

        Console.WriteLine(" rev : rank ⇒ value");

        SortedSetEntry[] revRange_players = db.SortedSetRangeByRankWithScores("players", start: 0, stop: -1,O
rder.Descending);
        Console.WriteLine("rank ⇒ value");
        foreach (var item in revRange_players)
        {
            Console.WriteLine($"{item.Element} ⇒ {item.Score}");
        }
```

```
    Console.WriteLine($"rank player ahmed : {db.SortedSetRank("players","ahmed")}");

    var rem = db.SortedSetRemove("players", "omar");
    Console.WriteLine("remove omar : "+rem);

    RedisValue[] values = db.SortedSetRangeByRank("players", start: 0, stop: -1);

    foreach (var item in values)
    {
        Console.WriteLine(item);
    }
}
```

> Output
>
> rank ⇒ value
> ahmed ⇒ 1
> omar ⇒ 2
> said ⇒ 5
> yasser ⇒ 8
> rev : rank ⇒ value
> rank ⇒ value
> yasser ⇒ 8
> said ⇒ 5
> omar ⇒ 2
> ahmed ⇒ 1
> rank player ahmed : 0
> remove omar : True
> ahmed
> said
> yasser

## ✅ What is a Redis Data Structure – Stream?

A **Redis Stream** is a powerful data structure introduced in Redis 5.0. It works like an **append-only log**, where you can continuously add new entries with unique IDs.

### 📌 Key Concepts:

- A **Redis stream** stores a list of entries in the order they are added.
- Each entry in a stream has:
  - A **unique ID** (generated by Redis unless specified).
  - One or more field-value pairs.

### 📌 Entry ID Format:

The ID is composed of **two parts**, separated by a hyphen:

```
php-template
<millisecondsTime>-<sequenceNumber>
```

### Example IDs:

```
1702457198528-0
1702457198528-1
1702457198528-2
```

- `1702457198528`: Timestamp in milliseconds.
- `0`, `1`, `2`: Sequence number to avoid collisions if multiple entries are added at the same millisecond.

### 📋 Redis Stream Commands Table

| Command | Description | Example |
|---|---|---|
| **XADD** | Adds a new entry to the stream with a unique or specified ID | `XADD mystream * name "Alice" age "30"` |
| **XDEL** | Deletes one or more entries from the stream by their IDs | `XDEL mystream 1702457198528-1` |
| **XLEN** | Returns the number of entries in the stream | `XLEN mystream` |
| **XRANGE** | Reads a range of entries from the stream (by start and end ID) | `XRANGE mystream - +` |
| **XREVRANGE** | Reads a range in reverse order | `XREVRANGE mystream + -` |
| **XREAD** | Reads new entries from one or more streams (blocking or non-blocking) | `XREAD COUNT 2 STREAMS mystream 0` |
| **XGROUP** | Manages consumer groups (create, destroy, set ID, etc.) | `XGROUP CREATE mystream mygroup $ MKSTREAM` |
| **XREADGROUP** | Reads entries from a stream using a consumer group | `XREADGROUP GROUP mygroup consumer1 STREAMS mystream >` |
| **XACK** | Acknowledges that a message has been processed (in consumer groups) | `XACK mystream mygroup 1702457198528-0` |
| **XPENDING** | Shows pending messages in a consumer group | `XPENDING mystream mygroup` |

| | | |
|---|---|---|
| **XCLAIM** | Transfers ownership of a pending message to another consumer | `XCLAIM mystream mygroup consumer2 0 1702457198528-0` |
| **XTRIM** | Trims the stream to a specific length or ID range | `XTRIM mystream MAXLEN 1000` |

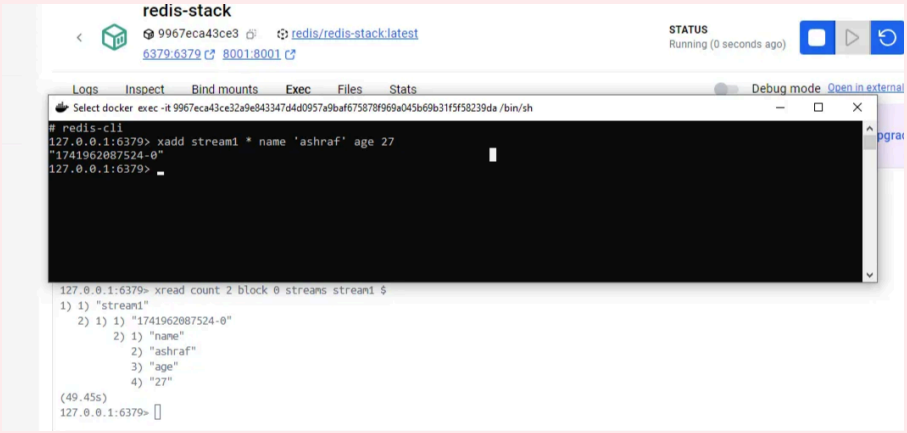
Commands:

```
# redis-cli
127.0.0.1:6379> zadd players 1 "ahmed" 2 "omar" 6 "said" 4 "ali"
(integer) 4
127.0.0.1:6379> zrange players 0 -1 withscores
1) "ahmed"
2) "1"
3) "omar"
4) "2"
5) "ali"
6) "4"
7) "said"
8) "6"
127.0.0.1:6379> zrevrange players -1 0 withscores
(empty array)
127.0.0.1:6379> zrevrange players 0 -1 withscores
1) "said"
2) "6"
3) "ali"
4) "4"
5) "omar"
6) "2"
7) "ahmed"
8) "1"
127.0.0.1:6379> zrangebyscore players 1 5
1) "ahmed"
2) "omar"
3) "ali"
127.0.0.1:6379> zrank players "said" withscore
1) (integer) 3
2) "6"
127.0.0.1:6379> zrevrank players "said"
(integer) 0
127.0.0.1:6379> zrevrank players "ahmed"
(integer) 3
127.0.0.1:6379> zrangebylex players [a [p
1) "ahmed"
2) "omar"
3) "ali"
127.0.0.1:6379> zrem players "said"
(integer) 1
127.0.0.1:6379> xadd stream1 * name 'ahmed' age 20
"1741961574808-0"
127.0.0.1:6379> xadd stream1 * name 'said'' age 25
Invalid argument(s)
127.0.0.1:6379> xadd stream1 * name 'said' age 25
"1741961608199-0"
127.0.0.1:6379> xadd stream1 * name 'omar' age 27
```

```
"1741961626884-0"
127.0.0.1:6379> xlen stream1
(integer) 3
127.0.0.1:6379> xrange stream1 0 -1
(error) ERR Invalid stream ID specified as stream command argument
127.0.0.1:6379> xrange stream1 - +
1) 1) "1741961574808-0"
   2) 1) "name"
      2) "ahmed"
      3) "age"
      4) "20"
2) 1) "1741961608199-0"
   2) 1) "name"
      2) "said"
      3) "age"
      4) "25"
3) 1) "1741961626884-0"
   2) 1) "name"
      2) "omar"
      3) "age"
      4) "27"
127.0.0.1:6379> xrevrange stream1 + -
1) 1) "1741961626884-0"
   2) 1) "name"
      2) "omar"
      3) "age"
      4) "27"
2) 1) "1741961608199-0"
   2) 1) "name"
      2) "said"
      3) "age"
      4) "25"
3) 1) "1741961574808-0"
   2) 1) "name"
      2) "ahmed"
      3) "age"
      4) "20"
127.0.0.1:6379> xdel stream1 1741961608199-0
(integer) 1
127.0.0.1:6379> xrange stream1 - +
1) 1) "1741961574808-0"
   2) 1) "name"
      2) "ahmed"
      3) "age"
      4) "20"
2) 1) "1741961626884-0"
   2) 1) "name"
      2) "omar"
      3) "age"
      4) "27"
127.0.0.1:6379> xadd stream1 * name 'kamel' age 21
"1741961881317-0"
```

```
127.0.0.1:6379> xadd stream1 * name 'ali' age 24
"1741961911743-0"
```

Commands for xRead to do this open new client :




Code C# :

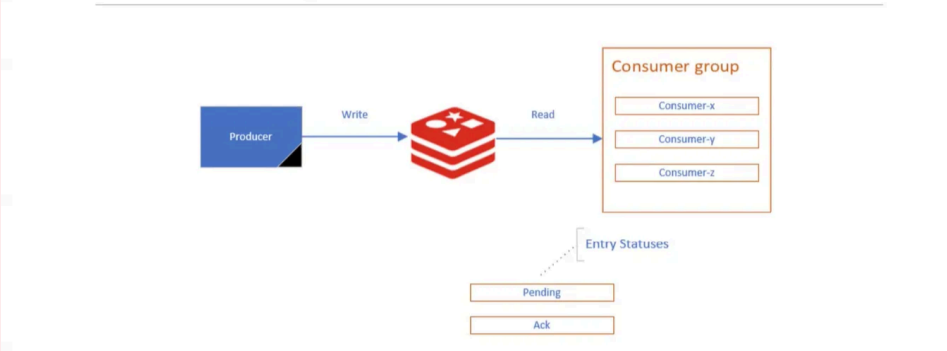
```
static void Main(string[] args)
{
   var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
   var db = redis.GetDatabase();

   var entryid1 = db.StreamAdd("user1", new NameValueEntry[]
         {
      new NameValueEntry("name","ahmed"),
      new NameValueEntry("age",20)
         });

   var entryid2 = db.StreamAdd("user1", new NameValueEntry[]
   {
      new NameValueEntry("name","said"),
      new NameValueEntry("age",22)
   });


   var entryid3 = db.StreamAdd("user1", new NameValueEntry[]
   {
      new NameValueEntry("name","omar"),
      new NameValueEntry("age",25)
   }/*,maxLength:2*/);//for specific max number of entries

   long startid = ((DateTimeOffset)DateTime.Now.AddHours(-1)).ToUnixTimeMilliseconds();

   long enddate = ((DateTimeOffset)DateTime.Now).ToUnixTimeMilliseconds();

   var Entries = db.StreamRange("user1", startid, enddate);

   foreach (var entry in Entries)
   {
```

```
        Console.WriteLine($"id = {entry.Id.ToString()}  values = " + JsonConvert.SerializeObject(entry.Value
s));
    }
    Console.WriteLine("after delete");

    db.StreamDelete("user1", new[] { entryid1 });
    var Entries_del = db.StreamRange("user1", startid, enddate);
    foreach (var entry in Entries_del)
    {
        Console.WriteLine($"id = {entry.Id.ToString()}  values = " + JsonConvert.SerializeObject(entry.Value
s));
    }
    Console.WriteLine("read stream");
    var readEntries = db.StreamRead("user1", startid, count: 1);
    foreach (var entry in readEntries)
    {
        Console.WriteLine($"id = {entry.Id.ToString()}  values = " + JsonConvert.SerializeObject(entry.Value
s));
    }

    Console.ReadKey();
  }
```

> Output:
>
> id = 1741965660267-0  values = [{"Name":"name","Value":"ahmed"},
> {"Name":"age","Value":"20"}]
> id = 1741965660270-0  values = [{"Name":"name","Value":"said"},
> {"Name":"age","Value":"22"}]
> id = 1741965660271-0  values = [{"Name":"name","Value":"omar"},
> {"Name":"age","Value":"25"}]
> after delete
> id = 1741965660270-0  values = [{"Name":"name","Value":"said"},
> {"Name":"age","Value":"22"}]
> id = 1741965660271-0  values = [{"Name":"name","Value":"omar"},
> {"Name":"age","Value":"25"}]
> read stream
> id = 1741965660270-0  values = [{"Name":"name","Value":"said"},
> {"Name":"age","Value":"22"}]

## 🔄 Redis Stream – Consumer Group Overview

A **consumer group** in Redis Streams is a way to process messages in parallel using multiple consumers. Think of the group as a manager that distributes messages to different workers (consumers).




### ✅ Key Points:

- **Pseudo-consumer:**

  A consumer group acts like a single consumer but actually contains **multiple consumers**.

- **Message Distribution:**

  Each message in the stream is **delivered to one specific consumer** within the group (not broadcasted to all).

- **Identified Consumers:**

  Every consumer is **identified** by a unique name (, `Consumer-x` , `Consumer-y` , `Consumer-z` ).

- **Pending Messages Tracking:**

  The group **tracks all pending messages** (messages delivered but not acknowledged yet).

- **First ID Concept:**

  Redis uses the idea of the **"first never consumed"** ID to help determine which messages haven't been read.

- **Explicit Acknowledgment:**

  Consumers must **acknowledge** that they've processed a message, otherwise it stays pending.

- **Useful Commands:**

  - `XGROUP` : Create, delete, or manage consumer groups.
  - `XREADGROUP` : Read messages as a part of a consumer group.

### 📦 Use Case Example:

Imagine a task queue (like for email sending or image processing). You can use Redis Streams + Consumer Groups to:

- Split tasks between multiple workers (consumers)
- Track which tasks are still pending
- Reassign unacknowledged tasks if a consumer crashes

Commands :

1. For create a consumer group and start first  consume from id = 0 ,stream not exists use `mkstream` to create one

```
# redis-cli
127.0.0.1:6379> xgroup create users mygroup 0 mkstream
```

2. For create a consumer group and start first  from the bigger id : the last one addded to group

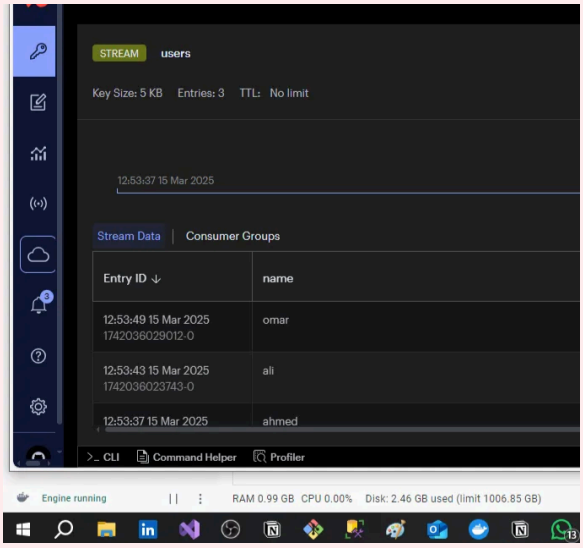
```
xgroup create users fgroup $
```

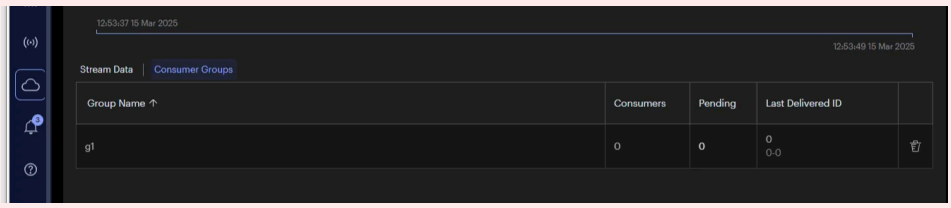
3. for create consumer and read entries from consumer group that no one read

```
127.0.0.1:6379> xreadgroup group fgroup con1 count 1 streams users >
```

### let's see this in redis inside :

```
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xadd users * name ahmed
"1742036017770-0"
127.0.0.1:6379> xadd users * name ali
"1742036023743-0"
127.0.0.1:6379> xadd users * name omar
"1742036029012-0"
```

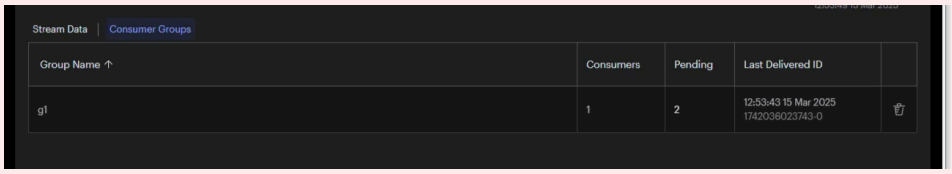



4. Consume: here you consume the first one entry that added to consumer group after it created

```
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
```

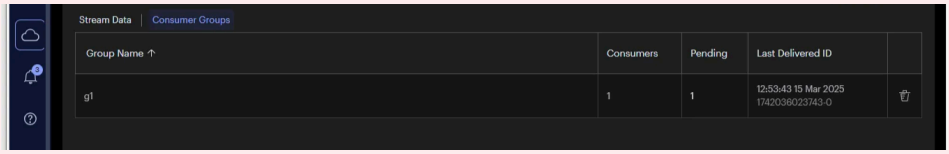
```
   2) 1) 1) "1742036017770-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742036023743-0"
         2) 1) "name"
            2) "ali
```




**here there are two pending**

5. after acknowledged

```
127.0.0.1:6379> xack users g1 1742036017770-0
(integer) 1
```




6. Code C#

```
namespace Redis_tester
{
    public class User
    {
        public string Name { get; set; }
        public int Age { get; set; }
    }
    internal class Program
    {

        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
            var db = redis.GetDatabase();

            var createdconsumergroup=db.StreamCreateConsumerGroup("users","g1","$",createStream:true);

            Console.WriteLine($"created my consumer group g1 : {createdconsumergroup}");

            db.StreamAdd("users", new NameValueEntry[]
            {
                new NameValueEntry("name","ahmed"),
                new NameValueEntry("age",20)
```

```
        });

        db.StreamAdd("users", new NameValueEntry[]
        {
           new NameValueEntry("name","ali"),
           new NameValueEntry("age",22)

        });

        db.StreamAdd("users", new NameValueEntry[]
        {
           new NameValueEntry("name","omar"),
           new NameValueEntry("age",26)

        });


        var readentries=db.StreamReadGroup("users","g1","c1",">",count:1);

        foreach (var Entry in readentries)
        {
           Dictionary<string, string> dic = new Dictionary<string, string>();
           foreach (var item in Entry.Values)
           {
              dic[item.Name] = item.Value;
           }
           var json= JsonConvert.SerializeObject(dic);
           var user=JsonConvert.DeserializeObject<User>(json);

           Console.WriteLine
           ($"Consumed: {Entry.Id} ⇒ Name: {user.Name}, Age: {user.Age}");
        }

        //acknowledged   consume entry
        foreach (var item in readentries)
        {
            db.StreamAcknowledge("users", "g1", item.Id);
        }

     }
   }
}
```

> Output
>
> created my consumer group g1 : True
> Consumed: 1742039333890-0 ⇒ Name: ahmed, Age: 20

How can know pending ?

```
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xadd users * name ahmed
"1742036017770-0"
127.0.0.1:6379> xadd users * name ali
"1742036023743-0"
127.0.0.1:6379> xadd users * name omar
"1742036029012-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742036017770-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742036023743-0"
         2) 1) "name"
            2) "ali"
127.0.0.1:6379> xack users g1 1742036017770-0
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xpending users g1
1) (integer) 0
2) (nil)
3) (nil)
4) (nil)
127.0.0.1:6379> xadd users * name ahmed
"1742040770128-0"
127.0.0.1:6379> xadd users * name ali
"1742040773905-0"
127.0.0.1:6379> xadd users * name omar
"1742040779064-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams >
(error) ERR Unbalanced 'xreadgroup' list of streams: for each stream key an ID or '>' must be specified.
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742040770128-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c2 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742040773905-0"
         2) 1) "name"
            2) "ali"
127.0.0.1:6379> xpending users g1
1) (integer) 2
2) "1742040770128-0"
3) "1742040773905-0"
4) 1) 1) "c1"
      2) "1"
   2) 1) "c2"
```

```
      2) "1"
127.0.0.1:6379> xack users g1 1742040773905-0
(integer) 1
127.0.0.1:6379> xpending users g1
1) (integer) 1
2) "1742040770128-0"
3) "1742040770128-0"
4) 1) 1) "c1"
      2) "1"
127.0.0.1:6379>
```

For identify the range of pending :

```
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xadd users * name ahmed
"1742036017770-0"
127.0.0.1:6379> xadd users * name ali
"1742036023743-0"
127.0.0.1:6379> xadd users * name omar
"1742036029012-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742036017770-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742036023743-0"
         2) 1) "name"
            2) "ali"
127.0.0.1:6379> xack users g1 1742036017770-0
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xpending users g1
1) (integer) 0
2) (nil)
3) (nil)
4) (nil)
127.0.0.1:6379> xadd users * name ahmed
"1742040770128-0"
127.0.0.1:6379> xadd users * name ali
"1742040773905-0"
127.0.0.1:6379> xadd users * name omar
"1742040779064-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams >
(error) ERR Unbalanced 'xreadgroup' list of streams: for each stream key an ID or '>' must be specified.
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742040770128-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c2 count 1 streams users >
1) 1) "users"
```

```
   2) 1) 1) "1742040773905-0"
         2) 1) "name"
            2) "ali"
127.0.0.1:6379> xpending users g1
1) (integer) 2
2) "1742040770128-0"
3) "1742040773905-0"
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xadd users * name ahmed
"1742041397411-0"
127.0.0.1:6379> xadd users * name ali
"1742041400729-0"
127.0.0.1:6379> xadd users * name omar
"1742041405529-0"
127.0.0.1:6379> xadd users * name said
"1742041410497-0"
127.0.0.1:6379> xadd users * name kamel
"1742041424849-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 5 streams users >
1) 1) "users"
   2) 1) 1) "1742041397411-0"
         2) 1) "name"
            2) "ahmed"
      2) 1) "1742041400729-0"
         2) 1) "name"
            2) "ali"
      3) 1) "1742041405529-0"
         2) 1) "name"
            2) "omar"
      4) 1) "1742041410497-0"
         2) 1) "name"
            2) "said"
      5) 1) "1742041424849-0"
         2) 1) "name"
            2) "kamel"
127.0.0.1:6379> xpending users g1 - + 3
1) 1) "1742041397411-0"
   2) "c1"
   3) (integer) 28158 ⇒ time from last time consumed
   4) (integer) 1 ⇒ here this entry send one time to one conumer
2) 1) "1742041400729-0"
   2) "c1"
   3) (integer) 28158
   4) (integer) 1
3) 1) "1742041405529-0"
   2) "c1"
   3) (integer) 28158
   4) (integer) 1
127.0.0.1:6379>
```

Claiming : Transfer pending entry from consumer to another

```
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xadd users * name ahmed
"1742036017770-0"
127.0.0.1:6379> xadd users * name ali
"1742036023743-0"
127.0.0.1:6379> xadd users * name omar
"1742036029012-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742036017770-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742036023743-0"
         2) 1) "name"
            2) "ali"
127.0.0.1:6379> xack users g1 1742036017770-0
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xpending users g1
1) (integer) 0
2) (nil)
3) (nil)
4) (nil)
127.0.0.1:6379> xadd users * name ahmed
"1742040770128-0"
127.0.0.1:6379> xadd users * name ali
"1742040773905-0"
127.0.0.1:6379> xadd users * name omar
"1742040779064-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams >
(error) ERR Unbalanced 'xreadgroup' list of streams: for each stream key an ID or '>' must be specified.
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742040770128-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c2 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742040773905-0"
         2) 1) "name"
            2) "ali"
127.0.0.1:6379> xpending users g1
1) (integer) 2
2) "1742040770128-0"
3) "1742040773905-0"
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xadd users * name ahmed
```

```
"1742041397411-0"
127.0.0.1:6379> xadd users * name ali
"1742041400729-0"
127.0.0.1:6379> xadd users * name omar
"1742041405529-0"
127.0.0.1:6379> xadd users * name said
"1742041410497-0"
127.0.0.1:6379> xadd users * name kamel
"1742041424849-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 5 streams users >
1) 1) "users"
   2) 1) 1) "1742041397411-0"
         2) 1) "name"
            2) "ahmed"
      2) 1) "1742041400729-0"
         2) 1) "name"
            2) "ali"
      3) 1) "1742041405529-0"
         2) 1) "name"
            2) "omar"
      4) 1) "1742041410497-0"
         2) 1) "name"
            2) "said"
      5) 1) "1742041424849-0"
         2) 1) "name"
            2) "kamel"
127.0.0.1:6379> xpending users g1 - + 3
127.0.0.1:6379> xgroup create users g1 $ mkstream
OK
127.0.0.1:6379> xadd users * name ahmed
"1742042438053-0"
127.0.0.1:6379> xadd users * name ali
"1742042441489-0"
127.0.0.1:6379> xadd users * name omar
"1742042445674-0"
127.0.0.1:6379> xadd users * name said
"1742042450851-0"
127.0.0.1:6379> xreadgroup group g1 c1 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742042438053-0"
         2) 1) "name"
            2) "ahmed"
127.0.0.1:6379> xreadgroup group g1 c2 count 1 streams users >
1) 1) "users"
   2) 1) 1) "1742042441489-0"
         2) 1) "name"
            2) "ali"
127.0.0.1:6379> xclaim users g1 c2 0 1742042438053-0
1) 1) "1742042438053-0"
   2) 1) "name"
      2) "ahmed"
127.0.0.1:6379> xpending users g1
1) (integer) 2
2) "1742042438053-0"
```

```
3) "1742042441489-0"
4) 1) 1) "c2"
      2) "2"
127.0.0.1:6379>

//auto pending

127.0.0.1:6379> xautoclaim users g1  c1 0 1742042438053-0 count 2
1) "0-0"
2) 1) 1) "1742042438053-0"
      2) 1) "name"
         2) "ahmed"
   2) 1) "1742042441489-0"
      2) 1) "name"
         2) "ali"
3) (empty array)
127.0.0.1:6379> xpending users g1
1) (integer) 2
2) "1742042438053-0"
3) "1742042441489-0"
4) 1) 1) "c1"
      2) "2"
```

C# Code:

```
namespace Redis_tester
{
    public class User
    {
        public string Name { get; set; }
        public int Age { get; set; }
    }
    internal class Program
    {

        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
            var db = redis.GetDatabase();

            var createdconsumergroup=db.StreamCreateConsumerGroup("users","g1","$",createStream:true);

            Console.WriteLine($"created my consumer group g1 : {createdconsumergroup}");

        var entryid=    db.StreamAdd("users", new NameValueEntry[]
            {
                new NameValueEntry("name","ahmed"),
                new NameValueEntry("age",20)

            });

            db.StreamAdd("users", new NameValueEntry[]
            {
                new NameValueEntry("name","ali"),
```

```
        new NameValueEntry("age",22)

    });

    db.StreamAdd("users", new NameValueEntry[]
    {
        new NameValueEntry("name","omar"),
        new NameValueEntry("age",26)

    });


    var readentries1=db.StreamReadGroup("users","g1","c1",">",count:1);

    foreach (var Entry in readentries1)
    {
        Dictionary<string, string> dic = new Dictionary<string, string>();
        foreach (var item in Entry.Values)
        {
            dic[item.Name] = item.Value;
        }
        var json= JsonConvert.SerializeObject(dic);
        var user=JsonConvert.DeserializeObject<User>(json);

        Console.WriteLine($"Consumed1: {Entry.Id} => Name: {user.Name}, Age: {user.Age}");
    }

    var readentries2 = db.StreamReadGroup("users", "g1", "c2", ">", count: 1);

    foreach (var Entry in readentries2)
    {
        Dictionary<string, string> dic = new Dictionary<string, string>();
        foreach (var item in Entry.Values)
        {
            dic[item.Name] = item.Value;
        }
        var json = JsonConvert.SerializeObject(dic);
        var user = JsonConvert.DeserializeObject<User>(json);

        Console.WriteLine($"Consumed2: {Entry.Id} => Name: {user.Name}, Age: {user.Age}");
    }


    var claimresult = db.StreamClaim("users", "g1", "c2", 0, new[] { entryid });

    Console.WriteLine($"claiming : {JsonConvert.SerializeObject(claimresult)}");

    Console.WriteLine("pending : ");
    var pending = db.StreamPending("users", "g1");
    Console.WriteLine($"pending : {JsonConvert.SerializeObject(pending)}");

}
```

```
    }
}
```

Output

```
created my consumer group g1 : True
Consumed1: 1742046354247-0 ⇒ Name: ahmed, Age: 20
Consumed2: 1742046354250-0 ⇒ Name: ali, Age: 22
claiming : [{"Id":"1742046354247-0","Values":[{"Name":"name","Value":"ahmed"},
{"Name":"age","Value":"20"}],"IsNull":false}]
pending :
pending : {"PendingMessageCount":2,"LowestPendingMessageId":"1742046354247-
0","HighestPendingMessageId":"1742046354250-0","Consumers":
[{"Name":"c2","PendingMessageCount":2}]}
```

Xinfo:

1. for stream

```
127.0.0.1:6379> xinfo stream users
 1) "length"
 2) (integer) 3
 3) "radix-tree-keys"
 4) (integer) 1
 5) "radix-tree-nodes"
 6) (integer) 2
 7) "last-generated-id"
 8) "1742044777208-0"
 9) "max-deleted-entry-id"
10) "0-0"
11) "entries-added"
12) (integer) 3
13) "recorded-first-entry-id"
14) "1742044777204-0"
15) "groups"
16) (integer) 1
17) "first-entry"
18) 1) "1742044777204-0"
    2) 1) "name"
       2) "ahmed"
       3) "age"
       4) "20"
19) "last-entry"
20) 1) "1742044777208-0"
    2) 1) "name"
       2) "omar"
       3) "age"
       4) "26"
```

2. for groups

```
127.0.0.1:6379> xinfo groups users
1)  1) "name"
    2) "g1"
    3) "consumers"
    4) (integer) 1
    5) "pending"
    6) (integer) 0
    7) "last-delivered-id"
    8) "1742044777204-0"
    9) "entries-read"
   10) (integer) 1
   11) "lag"
   12) (integer) 2
```

3. for consumers

```
127.0.0.1:6379> xinfo consumers users g1
1) 1) "name"
   2) "c1"
   3) "pending"
   4) (integer) 0
   5) "idle"
   6) (integer) 338498
   7) "inactive"
   8) (integer) 338498
```

C# Code :

```
namespace Redis_tester
{
    public class User
    {
        public string Name { get; set; }
        public int Age { get; set; }
    }
    internal class Program
    {

        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
            var db = redis.GetDatabase();

            var createdconsumergroup=db.StreamCreateConsumerGroup("users","g1","$",createStream:true);

            Console.WriteLine($"created my consumer group g1 : {createdconsumergroup}");

            db.StreamAdd("users", new NameValueEntry[]
            {
                new NameValueEntry("name","ahmed"),
                new NameValueEntry("age",20)

            });

            db.StreamAdd("users", new NameValueEntry[]
            {
                new NameValueEntry("name","ali"),
                new NameValueEntry("age",22)

            });

            db.StreamAdd("users", new NameValueEntry[]
            {
                new NameValueEntry("name","omar"),
                new NameValueEntry("age",26)

            });
```

```
            var readentries1=db.StreamReadGroup("users","g1","c1",">",count:1);

            foreach (var Entry in readentries1)
            {
               Dictionary<string, string> dic = new Dictionary<string, string>();
               foreach (var item in Entry.Values)
               {
                  dic[item.Name] = item.Value;
               }
               var json= JsonConvert.SerializeObject(dic);
               var user=JsonConvert.DeserializeObject<User>(json);

               Console.WriteLine($"Consumed1: {Entry.Id} => Name: {user.Name}, Age: {user.Age}");
            }

            var readentries2 = db.StreamReadGroup("users", "g1", "c1", ">", count: 1);

            foreach (var Entry in readentries2)
            {
               Dictionary<string, string> dic = new Dictionary<string, string>();
               foreach (var item in Entry.Values)
               {
                  dic[item.Name] = item.Value;
               }
               var json = JsonConvert.SerializeObject(dic);
               var user = JsonConvert.DeserializeObject<User>(json);

               Console.WriteLine($"Consumed2: {Entry.Id} => Name: {user.Name}, Age: {user.Age}");
            }

            //Xinfo
            var streaminfo = db.StreamInfo("users");
            var groupsinfo=db.StreamGroupInfo("users");
            var consumersinfo = db.StreamConsumerInfo("users", "g1");

            Console.WriteLine($"streaminfo : {JsonConvert.SerializeObject(streaminfo)}");

            Console.WriteLine("groups info : ");
            foreach (var item in groupsinfo)
            {
               Console.WriteLine($"groupinfo : {JsonConvert.SerializeObject(item)}");

            }

            Console.WriteLine("consumers info : ");
            foreach (var item in consumersinfo)
            {
               Console.WriteLine($"consumerinfo : {JsonConvert.SerializeObject(item)}");

            }
         }
      }
   }
```

Output

```
created my consumer group g1 : True
Consumed1: 1742045739395-0 ⇒ Name: ahmed, Age: 20
Consumed2: 1742045739397-0 ⇒ Name: ali, Age: 22
streaminfo :
{"Length":3,"RadixTreeKeys":1,"RadixTreeNodes":2,"ConsumerGroupCount":1,"FirstEntry":
{"Id":"1742045739395-0","Values":[{"Name":"name","Value":"ahmed"},
{"Name":"age","Value":"20"}],"IsNull":false},"LastEntry":{"Id":"1742045739400-0","Values"
[{"Name":"name","Value":"omar"},
{"Name":"age","Value":"26"}],"IsNull":false},"LastGeneratedId":"1742045739400-0"}
groups info :
groupinfo :
{"Name":"g1","ConsumerCount":1,"PendingMessageCount":2,"LastDeliveredId":"17420457
0","EntriesRead":2,"Lag":1}
consumers info :
consumerinfo : {"Name":"c1","PendingMessageCount":2,"IdleTimeInMilliseconds":10}
```

## 📍 Redis Data Structure – GeoSpatial

Redis provides **GeoSpatial features** to store, query, and retrieve locations using longitude and latitude — great for location-based applications like nearby stations, shops, drivers

### 🔧 Commands:

#### ✅ `GEOADD`

Used to add a geospatial item (longitude, latitude, name) to a Redis key.

**Example:**

```
GEOADD stations -122.27652 37.805186 Station1
GEOADD stations -122.2674626 37.8062344 Station2
GEOADD stations -122.2469854 37.8104049 Station3
```

#### ✅ `GEOSEARCH`

Searches for items within a certain radius or bounding box from a location or another item.

**Example:**

```
GEOSEARCH stations FROMLONLAT -122.27 37.80 BYRADIUS 2 km
```

This will return all stations within a 2 km radius of the given location.

### 📦 Data in This Example:

You're storing **station locations** like this:

| Station | Longitude | Latitude |
|---|---|---|
| Station1 | -122.27652 | 37.805186 |
| Station2 | -122.2674626 | 37.8062344 |
| Station3 | -122.2469854 | 37.8104049 |

### ✅ Use Cases:

- Find nearby stores
- Nearest bus/train stations
- Matching users/drivers by location
- Delivery zones

### 💡 Summary:

- Use `GEOADD` to store locations.
- Use `GEOSEARCH` to find nearby points by radius or box.
- Redis handles all the **math and spatial logic** behind the scenes.

Commands EX:

```
127.0.0.1:6379> geoadd stations -122.27652  37.805186 station1
(integer) 1
127.0.0.1:6379> geoadd stations -122.2674626 37.8062344  station2
```

```
(integer) 1
127.0.0.1:6379> geoadd stations -122.2469854  37.8104049 station3
(integer) 1
127.0.0.1:6379> GEOSEARCH stations FROMLONLAT -122.27 37.80 BYRADIUS 2 km
1) "station1"
2) "station2"
127.0.0.1:6379> GEOSEARCH stations FROMLONLAT -122.27 37.80 BYRADIUS 2 km withdist
1) 1) "station1"
   2) "0.8130"
2) 1) "station2"
   2) "0.7285"
127.0.0.1:6379>
```

C# Code:

```
        static void Main(string[] args)
        {
            var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
            var db = redis.GetDatabase();


            db.GeoAdd("stations", -122.27652, 37.805186, "station1");
            db.GeoAdd("stations", -122.2674626, 37.8062344, "station2");
            db.GeoAdd("stations", -122.2469854, 37.8104049, "station3");

            var results = db.GeoRadius("stations", -122.27, 37.80, 5, GeoUnit.Kilometers, options: GeoRadiusOption
s.WithDistance);

            foreach (var item in results)
            {
                Console.WriteLine($"position : {JsonConvert.SerializeObject(item.Member)} , destance ⇒ {item.Dist
ance}");
            }
        }
```

> Output:
>
> position : "station1" , destance ⇒ 0.813
> position : "station2" , destance ⇒ 0.7285
> position : "station3" , destance ⇒ 2.3303

## 🧠 Redis Data Structure – BitMap

BitMaps in Redis are a **space-efficient way** to store binary values (0s and 1s) at specific **bit offsets** inside a key. They are great for tracking things like:

- User activity (did user log in on day X?)
- Feature usage (did user use feature Y?)
- Unique counts (bitmaps + HyperLogLog)

---

### 🔧 Main Commands:

#### ✅ SETBIT

Sets or clears the bit at a specified offset.

**Example:**

```
SETBIT bitkey 1 1   # Sets bit at offset 1 to 1 → bitkey = 01
SETBIT bitkey 5 1   # Sets bit at offset 5 to 1 → bitkey = 010001
SETBIT bitkey 11 1  # Sets bit at offset 11 to 1 → bitkey = 01000100001
```

> Bits in between are implicitly 0 unless explicitly set.

#### ✅ GETBIT

Gets the value (0 or 1) at the specified offset.

```
GETBIT bitkey 5  # Output: 1
GETBIT bitkey 6  # Output: 0
```

#### ✅ BITCOUNT

Counts the number of bits set to 1 in the bitmap.

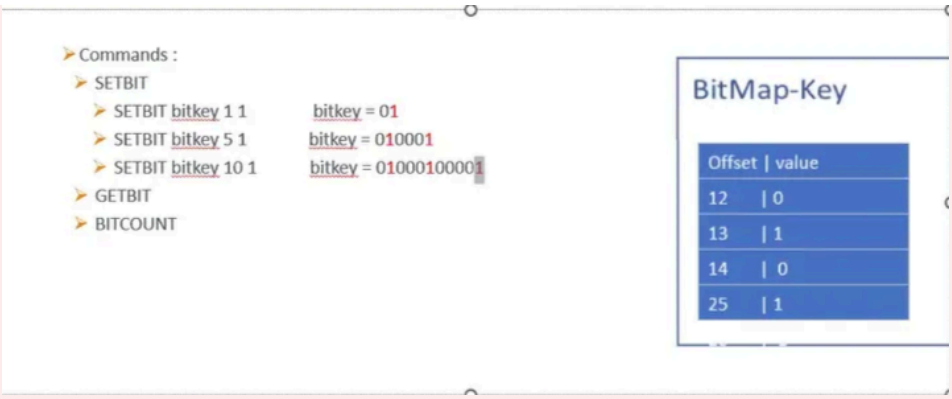
```
BITCOUNT bitkey  # Output: total number of 1s
```

---

### 📦 Example (BitMap-Key Table):

| Offset | Value |
|---|---|
| 12 | 0 |
| 13 | 1 |
| 14 | 0 |
| 25 | 1 |

These represent binary data at various offsets, useful for tracking presence/absence.




### 💡 Use Cases:

- User presence tracking (active/inactive)
- Event participation logs
- Efficient large-scale booleans
- Daily logins, attendance, etc.

Commands Ex:

```
127.0.0.1:6379> setbit k1 1 1
(integer) 0
127.0.0.1:6379> setbit k1 4 0
(integer) 0
127.0.0.1:6379> setbit k1 7 1
(integer) 0
127.0.0.1:6379> setbit k1 11 0
(integer) 0
127.0.0.1:6379> getbit k1 7
(integer) 1
127.0.0.1:6379> getbit k1 4
(integer) 0
127.0.0.1:6379> bitcount k1
(integer) 2
127.0.0.1:6379> getbit k1 40
(integer) 0
127.0.0.1:6379>
```

Code C# :

```
    static void Main(string[] args)
    {
        var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
        var db = redis.GetDatabase();


        db.StringSetBit("k1",1,true);
        db.StringSetBit("k1", 4, true);
        db.StringSetBit("k1", 7, false);
```

```
        db.StringSetBit("k1", 18, false);
        db.StringSetBit("k1", 9, true);


        Console.WriteLine($"offset 1 : value :  {db.StringGetBit("k1",1)}");
        Console.WriteLine($"offset 4 : value :  {db.StringGetBit("k1", 4)}");
        Console.WriteLine($"offset 7 : value :  {db.StringGetBit("k1", 7)}");
        Console.WriteLine($"offset 18 : value :  {db.StringGetBit("k1", 18)}");
        Console.WriteLine($"offset 9 : value :  {db.StringGetBit("k1", 9)}");
        Console.WriteLine($"offset 10 : value :  {db.StringGetBit("k1", 10)}");


        Console.ReadKey();
    }
```

> Output:
>
> offset 1 : value :  True
> offset 4 : value :  True
> offset 7 : value :  False
> offset 18 : value :  False
> offset 9 : value :  True
> offset 10 : value :  False

## 🧠 Redis Data Structure – BitFields

**BitFields** allow you to manipulate arbitrary-sized integers within a string value in Redis using bit-level operations. This is useful for efficiently packing and working with multiple integers in a compact form.

### ✨ Supported Operations:

- **Set** a value at a bit offset
- **Get** a value from a bit offset
- **Increment** a value at a bit offset

### 🧱 BitField Offsets:

Offsets define where the data starts within the bit string.

- `offset = 0 or 1 or 2 ...`
  Just regular positions like array indexes.
- `#0 × u32 = 0`
  → Start at position 0, treating values as 32-bit unsigned ints.
- `#1 × u32 = 32`
  → The next 32-bit chunk starts at offset 32.

This means Redis treats each u32 (unsigned 32-bit int) as a 4-byte field, so the position is a multiple of 32.

### 🔧 Commands:

#### ✅ `BITFIELD`

Used to perform read/write/increment operations.

**Example:**

```
BITFIELD mykey SET u8 0 100
```

→ Sets an 8-bit unsigned integer at offset 0 to value 100.

```
BITFIELD mykey GET u8 0
```

→ Gets the 8-bit unsigned integer at offset 0.

```
BITFIELD mykey INCRBY u8 0 1
```

→ Increments value at offset 0 by 1.

#### ✅ `BITFIELD_RO`

Read-only version of `BITFIELD`, useful in Redis replicas or safe reads.

### 🧰 Use Cases:

- Compact counters
- Storing game scores or levels
- Packing multiple small values into one key

- Efficient tracking of attributes (e.g., 10 properties in one 32-bit field)

**Commands:**

```
127.0.0.1:6379> bitfield player:1 set u32 #0 1000
1) (integer) 0
127.0.0.1:6379> bitfield player:1 get u32 #0
1) (integer) 1000
127.0.0.1:6379> bitfield player:1 incrby u32 #0 50 incrby u32 #1 1
1) (integer) 1050
2) (integer) 1
127.0.0.1:6379> bitfield player:1 get u32 #1
1) (integer) 1
127.0.0.1:6379> bitfield_Ro player:1 get u32 #0
1) (integer) 1050
```

# 📊 Redis Data Structure – HyperLogLog

## 🧠 What is HyperLogLog?

**HyperLogLog** is a probabilistic data structure used in Redis to estimate the **number of unique elements** in a set.

✅ It provides a **memory-efficient** way to count unique items without storing them.

---

## 🔧 Common Commands:

### 1. `PFADD key element [element ...]`

- Adds the specified elements to the HyperLogLog.
- Redis will use the elements to update its internal state (it **does not store the elements** themselves).

**Example:**

```
PFADD visitors user1 user2 user3
```

### 2. `PFCOUNT key [key ...]`

- Returns the **approximate number of unique elements** added.

**Example:**

```
PFCOUNT visitors
```

---

## 📦 Memory Usage:

- Uses only about **12 KB** of memory to track millions of unique elements.

---

## 🧪 Use Cases:

- Counting unique users visiting a website.
- Counting distinct IP addresses or search terms.
- Counting unique transactions or devices.

---

Commands:

```
127.0.0.1:6379> pfadd user ahmed
(integer) 1
127.0.0.1:6379> pfadd user ali
(integer) 1
127.0.0.1:6379> pfadd user ali
(integer) 0
127.0.0.1:6379> pfadd user ahmed
(integer) 0
127.0.0.1:6379> pfcount user
(integer) 2
```

C# Code :

```
        static void Main(string[] args)
        {
```

```
            var redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");
            var db = redis.GetDatabase();

        var ah1=  db.HyperLogLogAdd("user1", "ahmed");
        var al1=   db.HyperLogLogAdd("user1", "ali");
        var ah2= db.HyperLogLogAdd("user1", "ahmed");

            Console.WriteLine($"ahmed added : {ah1}");
            Console.WriteLine($"ali added : {al1}");
            Console.WriteLine($"ahmed added : {ah2}");


            Console.WriteLine($"unique values : {db.HyperLogLogLength("user1")}");

            Console.ReadKey();
        }
```

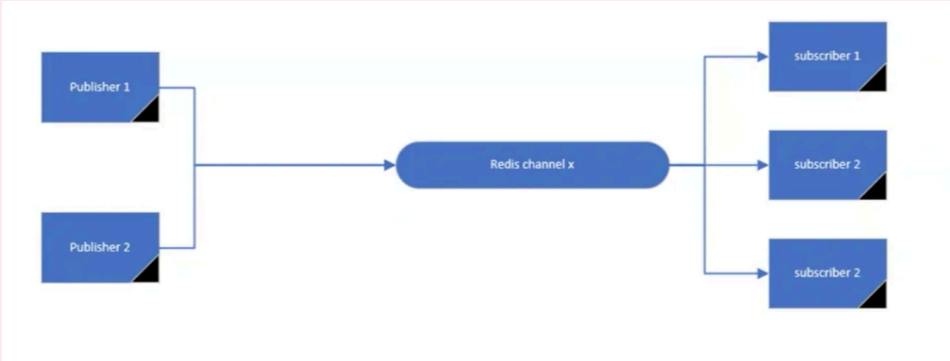
Output

> ahmed added : True
> ali added : True
> ahmed added : False
> unique values : 2

# 🔄 Redis Pub/Sub (Publish/Subscribe)

## 🧩 What is Pub/Sub?

Redis can act as a **fast and efficient message broker** using the **Publisher/Subscriber** pattern:




- **Publishers** send messages to a **channel**.
- **Subscribers** listen to that channel and receive messages.

This allows **real-time communication** between components.

---

## 🔧 Core Commands:

### 1. `PUBLISH <channel> <message>`

- Sends a message to a channel.

### 2. `SUBSCRIBE <channel>`

- Subscribes a client to a channel to start receiving messages.

---

## 📊 Diagram Explanation:

- **Publishers (1 & 2)** send messages to **Redis Channel X**.
- **Subscribers (1, 2, 3)** are listening on **Channel X**.
- Every message sent to the channel is broadcast to **all active subscribers**.

---

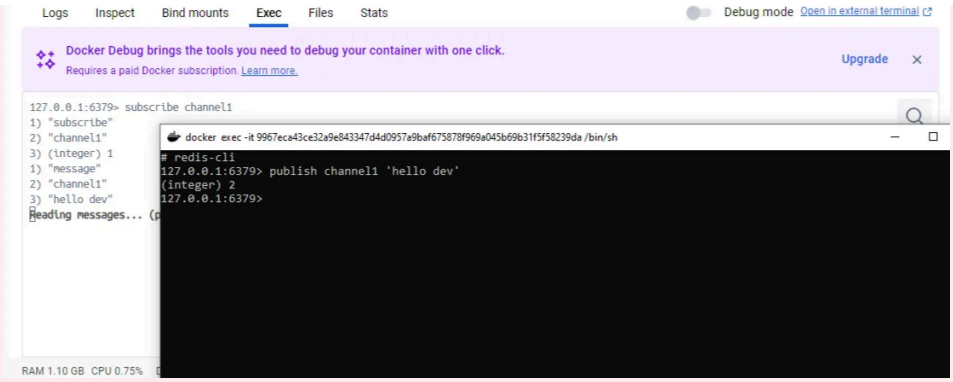
## 🛠️ Example in Redis CLI:

```
# Terminal 1: Subscriber
SUBSCRIBE news

# Terminal 2: Publisher
PUBLISH news "Breaking news: Redis is awesome!"
```

**Result:** The subscriber instantly receives the message.

---

Example:




**Example in C#:**

```
using StackExchange.Redis;

//Connect to localhost on port 6379.
ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");
IDatabase db = redis.GetDatabase();

PublishSample(db);

1 reference
void PublishSample(IDatabase db)
{
    while(true)
    {
        Console.WriteLine("enter your message here");
        string message = Console.ReadLine();
        db.Publish(
        Console.Wri
    }

    //Console.ReadLine();
}
```



```
using StackExchange.Redis;

//Connect to localhost on port 6379.
ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");
IDatabase db = redis.GetDatabase();

SubscribeSample(db);

1 reference
void SubscribeSample(IDatabase db)
{
  var subscriber=  redis.GetSubscriber();
    subscriber.Subscribe("channel2", (channel, message) =>
    {
        Console.WriteLine($"message recieved = {message}");
    });

    Console.ReadLine();
}
```

## 🚀 Redis Modules Overview

### 1️⃣ RedisSearch

- A full-text search engine built into Redis.
- Enables **fast search and filtering** on structured/unstructured data.
- Supports **fuzzy search, autocomplete, and ranking**.

### 2️⃣ RedisJson

- Allows storing and manipulating **JSON data** natively in Redis.
- Supports **nested queries** and **partial updates**.

### 3️⃣ RedisTimeSeries

- Optimized for storing and analyzing **time-series data**.
- Ideal for **IoT, monitoring, and real-time analytics**.

### 4️⃣ RedisGraph

- A **graph database** built on top of Redis.
- Uses **nodes and edges** to represent relationships (as shown in the diagram).
- Supports **Cypher query language**.

### 5️⃣ RedisBloom

- Implements **probabilistic data structures** such as:
  - **Bloom filters** (fast membership testing).
  - **Cuckoo filters** (alternative to bloom filters).
  - **Top-K** (tracking most frequent elements).
  - **Count-min sketch** (approximate counting).

### 6️⃣ RedisAI

- Integrates Redis with **AI/ML models**.
- Supports **TensorFlow, PyTorch, ONNX**.
- Ideal for **real-time AI inference**.

### 7️⃣ RedisGears

- A **serverless engine** for event-driven data processing.
- Enables **custom scripts** for **data transformation and automation**.

---

### 🛠️ Use Cases

- **RedisSearch** → Searching blog articles.
- **RedisJson** → Storing user profiles.
- **RedisTimeSeries** → Monitoring server CPU usage.
- **RedisGraph** → Social network relationships.
- **RedisBloom** → Preventing duplicate registrations.

- **RedisAI** → Fraud detection in real-time.
- **RedisGears** → Automating data pipelines.

## 🔶 1. What is caching?

**Definition:**
Caching is the process of storing frequently accessed data in a temporary storage (called cache) so that future requests for that data can be served faster.

**In simple words:**
Instead of fetching data every time from a slow source (like a database), we store it in a fast memory (like Redis or in-memory cache), so the app becomes faster and more efficient.

**Example:**
When you visit a website, your browser stores static files (like images or CSS) in cache — next time, it loads instantly without re-downloading.

---

## 🔶 2. Benefits of caching

✅ **Improves Performance**

Fetching from cache is much faster than from a database or API.

✅ **Reduces Server Load**

Fewer calls to the backend/database.

✅ **Enhances Scalability**

Can serve more users with fewer resources.

✅ **Improves User Experience**

Faster response = happier users!

✅ **Reduces Cost**

Less pressure on expensive backend resources.

---

## 🔶 3. Which data can be cached?

You can cache **any data that doesn't change frequently** or is **read more than written**.

🔷 **Examples of cachable data:**

- Product catalog
- Homepage content
- User profile (read-only parts)
- Search results
- Authentication tokens
- Configuration data

🔷 **Data you should NOT cache:**

- Frequently changing data (like stock prices)
- User-specific sensitive data (unless well-handled)
- Data that must always be up to date

---

## 💡 Summary

| Concept | Meaning |
| --- | --- |
| Caching | Storing data in fast memory for quicker access |
| Why cache? | Speed, performance, scalability, cost-saving |

| What to cache? | Static/frequently-read data, avoid fast-changing data |
| --- | --- |

### ✅ Without caching:

- Every time the client makes a request, the application server has to fetch data from the database.
- This creates **more load** on the database and **slower response times**.

### ✅ With caching:

- The application server stores frequently accessed data temporarily (in a cache).
- When the client makes a request, the server can respond **immediately from the cache**, **without hitting the database**.
- This improves **performance** and **reduces database load**.

## Caching Use Cases

1. **HTTP Application Cache** (on the website):
   - Stores data directly on the server.
   - Prevents repeated generation or fetching of the same data.
   - Fastest access for future requests.
2. **HTTP Network Cache** (in the network, like a CDN or proxy):
   - Stores frequently accessed responses closer to the client.
   - Reduces the need to reach the website every time.
   - Improves speed for geographically distributed users.
3. **Web Browser Cache** (on the user's device):
   - Saves static files (like images, CSS, JS) locally.
   - Prevents repeated downloads.
   - Makes page load almost instant for the user on repeated visits.

4. **Service Cache**

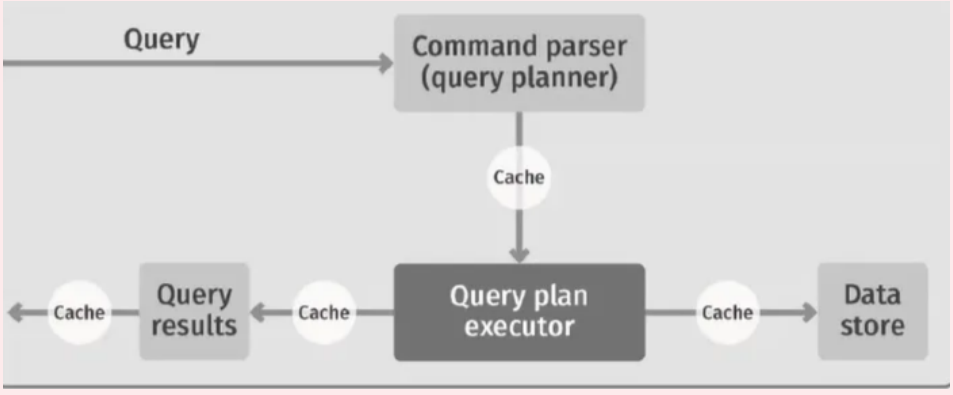
- Used to store the results of expensive service operations (like heavy computations or external service calls).
- Improves performance by avoiding repeated processing for the same request.

5. **Application Programming Interface (API) Cache**

- Stores responses from API calls.
- Reduces the load on the server and decreases response time by reusing previous API responses.

6. **Database Cache**

- Speeds up database operations by caching:
  - Query execution plans
  - Query results
  - Frequently accessed data




## Caching Use Cases (Hardware/System-Level)

### 1. CPU Cache

- A small, fast memory located inside or very close to the CPU.
- Stores frequently accessed instructions and data.
- **Purpose**: Speeds up processing by reducing the need to access slower main memory (RAM).

### 2. Memory Cache (RAM Cache)

- A cache that stores data in system memory (RAM) to serve faster than reloading from disk.
- Often used by applications to avoid recomputing or reloading data.
- **Purpose**: Improves application performance by keeping frequently accessed data in RAM.

### 3. Disk Cache

- A portion of memory (RAM or dedicated storage) that stores recently read or written data from disk (HDD/SSD).
- Used by operating systems and disk controllers.
- **Purpose**: Reduces disk access times, speeding up file read/write operations.

---

## When to Use Caching (Conditions):

1. 🔄 **Slow Operation**
   - The operation (e.g., database query, API call) takes time to complete.
   - Caching avoids repeated slow calls.
2. ⚡ **Faster Results Using Fewer Resources**
   - Instead of recalculating or re-fetching, serve the result from cache.
   - Saves CPU, memory, and I/O.
3. 🔁 **Data That Doesn't Change Frequently**
   - Ideal for **static or rarely changing data** (like product lists, config values).
   - If the result stays the same across requests, it's a good cache candidate.
4. 🧪 **No Side-Effects**
   - The operation should not cause any changes (no writing to DB, no sending emails, etc.).
   - Pure functions (same input gives same output) are ideal for caching.
5. 🔁 **Data Is Needed Multiple Times**
   - If the same data is used more than once (even across users), cache it.
   - Avoid repeated expensive computation or loading.

## 🔄 Static vs Dynamic Caches

### 🔷 Static Cache (Read-only)

- Data is **not updated** in the cache.
- When an update is required, it **bypasses the cache** and goes directly to the **persistent data store**.
- Cache stays the same until it's manually refreshed or invalidated.
- Used when data doesn’t change frequently or consistency is critical.

🧠 Example use: CDN for static files (images, CSS, etc.)

---

### 🔶 Dynamic Cache (Read/Write)

- Cache supports **both read and write** operations.
- Updates happen in the **cache first**, then are pushed to the **data store**.
- Provides **faster writes** and **quicker reads**.
- Suitable when data updates are frequent and immediate performance is key.

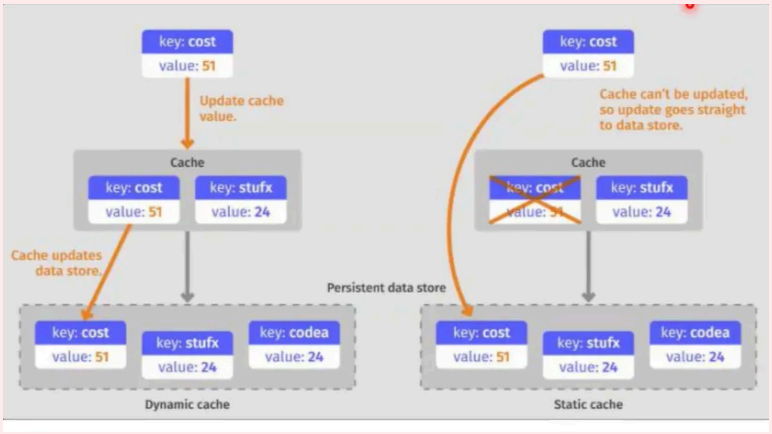
🧠 Example use: In-memory caches like Redis for session state or user profiles.

---

### 🧩 Summary (from the diagram):

- **Dynamic Cache**: Writes go to cache → cache updates the DB.
- **Static Cache**: Cache doesn’t allow writes → writes go straight to DB.

---



## ✅ Problems Solved by Caching

### 1. Performance Improvement

- Reduces latency by serving data faster.
- Avoids expensive recalculations (like in the example: `MUL 3 4 = 12` is directly returned).

### 2. Scaling

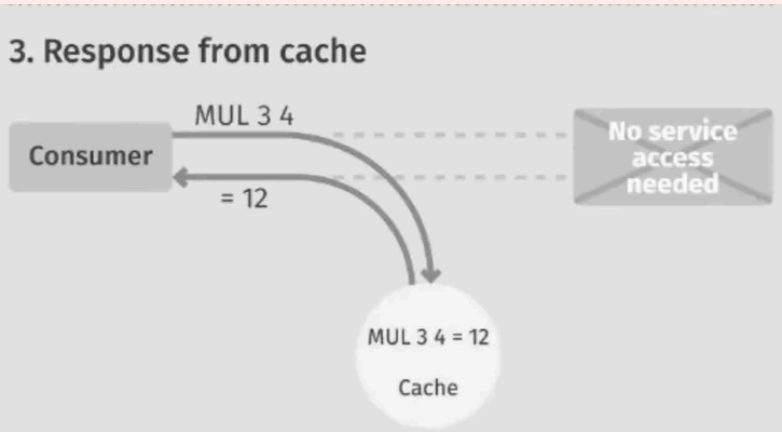
- Reduces load on backend services and databases.
- Helps handle high volumes of traffic more efficiently.

### 3. Resource Optimization

- Saves computational power and network bandwidth.
- Reuses previously fetched or calculated data.

### 4. Availability

- If the backend service is down, cached data can still serve user requests.
- Enhances system reliability.



## 🧠 Cache Concerns (Made Simple)

### 1. Side Effects

- Some actions **shouldn't be cached** (like making a payment or submitting a form).
- If they are cached, the system might **repeat the same action** without meaning to. That can cause problems.

### 2. Inconsistent Data

- The cache might show **old data**.
- If the real data changes and the cache doesn't update, users might see **wrong info**.

### 3. Poor Cache Performance

- If your cache **rarely gets used**, it's not helping.
- You're using memory and time for something that **isn't giving good results**.

---

## 🔵 Cold Cache

- Means: ❌ No data is saved in the cache yet.
- Example: Like an **empty fridge** — nothing inside yet.

---

## 🟢 Warm Cache

- Means: ✅ Data is already in the cache and ready to use.
- Example: Like a **fridge full of food** — ready to eat!

---

## 🔄 Warming the Cache

- Means: Adding important data to the cache **before it's needed**.
- Example: Like putting food in the fridge so it's **ready when someone is hungry**.

---

### 🟠 Cache Persistence

- Means: Saving cache data to **permanent storage** (like a hard disk).
- Purpose: So that **even if the power goes off**, the cache is not lost.
- Think of it like: Saving a game so you can load it later.

---

### 🔵 Rehydration

- Means: Taking the saved cache from **persistent storage** and putting it **back into RAM** (temporary memory).
- Purpose: To **restore** the cache when the system restarts.
- Think of it like: **Loading your saved game** after turning your device back on.

---

### 💡 In Short:

- Cache persistence = **Save the cache**.
- Rehydration = **Load the cache**.

## 📌 Redis Persistence Types

Redis provides different ways to persist data so that it can survive a restart. This slide lists the main persistence options:

1. **RDB (Redis Database):**
   - Takes **snapshots** of your dataset at specified intervals.
   - Saves them to disk.
   - It's fast and efficient but may lose recent data if Redis crashes.
2. **AOF (Append-Only File):**
   - Logs **every write operation** received by the server.
   - Writes are appended to a file.
   - Provides better durability than RDB, but it can be slower.
3. **No Persistence:**
   - Data is stored only in memory.
   - If Redis restarts or crashes, **all data is lost**.
   - Suitable when using Redis purely as a **cache**.
4. **RDB + AOF (Both):**
   - Combines the advantages of both.
   - RDB offers faster startup.
   - AOF ensures minimal data loss.

---

## 💡 Analogy:

Imagine you're writing a document:

- **RDB** = Auto-save every 10 minutes.
- **AOF** = Save every time you type a character.
- **No Persistence** = Never save.
- **RDB + AOF** = Save every 10 minutes **and** save every keystroke.

## 🟧 Redis Persistence – RDB (Redis Database)

### ✅ Advantages:

- **RDB files are perfect for backups**

  You can easily store them and move them to remote storage for safety.

- **Disaster recovery**

  If your server crashes, you can recover data from the most recent snapshot.

- **Faster than AOF**

  Since RDB saves the entire dataset at once (not every operation), it has better performance during normal operations.

### ❌ Disadvantages:

- **Data loss**
  Since RDB snapshots are taken at intervals, any data added between the last snapshot and a crash **will be lost**.

## 🟦 Redis Persistence – AOF (Append-Only File)

### ✅ Advantages:

- **More durable**

  Redis is more reliable with AOF, as it logs every write operation.

- **Background rewrite**

  AOF can be rewritten in the background to reduce file size and improve performance.

  by create in background new file when file size is high

- **Operation-by-operation logging**

  It logs **each command** in order, so it can replay them to restore the data.

  this help when you do a `flushall` can cancel this by go to command and remove it and restart redis

### ❌ Disadvantages:

- **Larger file size**

  AOF files are usually bigger than RDB files for the same data.

- **Potentially slower**

  AOF can be slower than RDB depending on the `fsync` policy (how often the data is written to disk).

- **High memory usage during rewrite**

  If writes happen during a rewrite, it can use a lot of memory.

## 📸 Snapshotting (RDB Persistence)

Redis supports snapshotting as one of its persistence methods using the **RDB (Redis Database Backup)** format. There are two main commands used:

➤ `SAVE`

- Creates a snapshot **synchronously**.
- Blocks the Redis server until the save is complete.
- **Used rarely in production** because it blocks clients during the process.
- saving acts in file in folder data & file name : `dump.rdb`

➤ `BGSAVE`

- Creates a snapshot **asynchronously**.
- Forks a child process to handle the saving.
- **Recommended** in production environments because it allows Redis to continue serving clients during the save.
- replace the `dump.rdb` with new after finishing backup

```
# redis-cli
127.0.0.1:6379> save
OK
127.0.0.1:6379> BGsave
Background saving started
127.0.0.1:6379>
```

## Append-only file (AOF)

This is another persistence mechanism Redis offers. Unlike snapshotting (RDB), AOF logs **every write operation** received by the server. It's more durable and allows **replaying** the commands to rebuild the dataset.

### Config: `appendonly yes`

- Enables AOF persistence.
- Configurable in the `redis.conf` file.

### Base file

- The starting point of the AOF rewrite.
- Contains a full snapshot of data up to a certain point.

### Increment files

- Store changes after the base file.
- Represent additional commands that happened after the snapshot.

### Manifest file

- Keeps track of the AOF base and increment files.
- Helps Redis know the correct order to reconstruct the dataset during recovery.

---

To apply AOF

create file `redis.conf` and writein it : `(appendonly yes)`

and use this command in powershell

```
docker run -d --name redis-AOF -p 6379:6379 -v (your location of : redis.conf ):/redis.conf redis:latest redis-server /redis.conf
```

### Storing Log File in Redis (AOF Mode)

The `appendfsync` setting controls **how often Redis writes to the disk** when using the Append Only File (AOF) feature.

There are three options:

1. `appendfsync no`

   Redis does **not** explicitly sync data to disk. The operating system handles flushing the data eventually.

   🔶 *Faster performance but risk of data loss if the system crashes.*

   ✅ Command:

   ```
   config set appendfsync no
   ```

2. `appendfsync everysec`

   Redis syncs data to disk **every second**.

   🔶 *A good balance between performance and safety. Only ~1 second of data might be lost on crash.*

   ✅ Command:

   ```
   config set appendfsync everysec
   ```

3. `appendfsync always`

   Redis syncs data to disk **after every write operation**.

   🔶 *Very safe but slow performance.*

   ✅ Command:

   ```
   config set appendfsync always
   ```

## 🔒 Redis Persistence (Saving Data)

Redis provides two main ways to persist data (i.e., save it so it's not lost after a restart):

### 1. AOF – Append Only File

- **Configuration Command:**

  ```
  config set appendonly yes
  ```

  This enables AOF persistence, where Redis logs every write operation it receives.

- **Write Behavior Command:**

  ```
  config set appendfsync always
  ```

  This ensures Redis writes every operation to the AOF file **immediately** after it happens.

  ⚠️ This is the **safest** but **slowest** option because it flushes to disk constantly.

### 2. RDB – Redis Database Backup (Snapshotting)

- **Command:**

  ```
  save
  ```

  Creates a snapshot of the dataset **synchronously** (blocking operation).

- **Command:**

  ```
  bgsave
  ```

  Creates a snapshot of the dataset **asynchronously** (runs in the background, non-blocking).

### 📝 Summary:

- **AOF** is better for durability (data safety).
- **RDB** is better for performance (faster but risk of data loss if crash happens before next snapshot).
- You can use **both together** for the best of both worlds.

## Cache Scaling

### ✅ Cache Advantages:

- Improves **performance** (faster data access)
- Reduces usage of **backend resources** (CPU, database queries, etc.)

---

### ⚠️ Limits of Caching:

1. **Storage Limits:**
   - The cache has limited memory or space.
   - When it’s full, older or less-used items may be removed (evicted).
2. **Resource Limits:**
   - Caching consumes system resources like memory and processing power.
   - More cache = more resource usage.

---

In short:

While caching greatly boosts speed and efficiency, it has limitations in storage and resource consumption. To handle this, **scaling** (adding nodes or using distributed caching) may be necessary.

Let me know if you want a summary in Arabic too!

## Improving Cache Scalability

To make caching more scalable, we focus on:

### ⬆️ Vertical Scaling (Scaling Up & Down):

- This means **upgrading a single server** by giving it:
  - **More RAM**
  - **More CPU**
  - **More Bandwidth**
- Makes one machine more powerful to handle more cache.
- 🟢 Easy to implement but has physical limits.

### 🔁 Horizontal Scaling (Scaling Out & In):

- This means **adding more servers** (nodes).
- The cache is **spread across multiple machines**.
- 🟢 Increases capacity and availability.
- 💡 Requires coordination between nodes.

---

### ✅ Goal:

**Increase storage & resource limits** so the cache can handle more data and serve more users efficiently.

## Horizontal Scaling Techniques: Read Replicas

### 📌 What's happening here?

- The **application** sends:
  - **Write requests** → to the **Redis master**
  - **Read requests** → to **read replicas** through a **load balancer**

---

### 📫 Redis Master:

- Handles all **write operations**
- Updates are automatically shared with all **read replicas**

### 📫 Read Replicas:

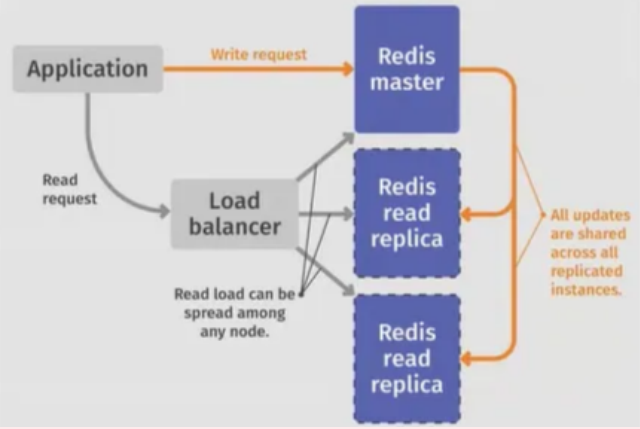
- Handle **read operations only**
- Help **reduce the load** on the master
- Can **scale out** by adding more replicas

---

### ✅ Benefits:

- Improves **performance** and **availability**
- Reduces **latency** for read-heavy applications
- Enables **horizontal scaling**

---




### ⚖️ Load Balancer:

- Distributes **read traffic** evenly across the replicas

### What does the Load Balancer do here?

It **distributes read requests** from the application to different **Redis read replicas**.

### Why?

- To **reduce the load** on the Redis master.
- To **improve performance** by using multiple replicas to answer read requests.
- To **make responses faster** (lower latency).

### Example:

If there are 3 read replicas and 1000 read requests, the load balancer might send:

- 330 requests to the first replica
- 330 to the second
- 340 to the third

So instead of one server handling everything, the work is shared.

## 🔷 Horizontal Scaling Technique – Sharding

**What is Sharding?**

Sharding is a method of **splitting data** across multiple databases (called **shards**) so that each shard holds only a **portion** of the total data.

### 🔶 How it works (as shown in the image):

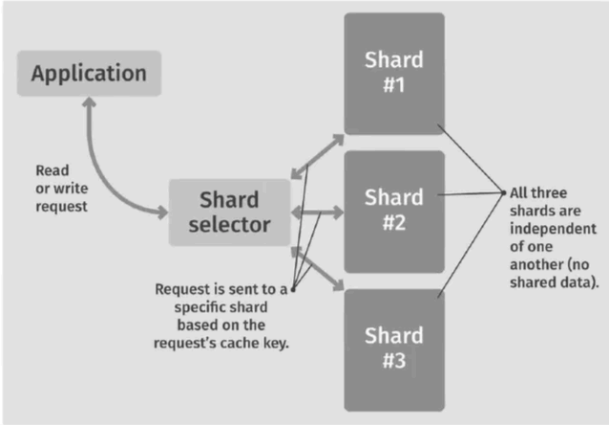
1. The **application** wants to read or write data.
2. It sends the request to a **Shard Selector**.
3. The **Shard Selector** decides **which shard** should handle the request.
   ➤ It uses a
   **cache key** or some rule (like user ID or data type) to choose the correct shard.
4. The request goes to the chosen shard (Shard #1, #2, or #3).
5. That shard processes the request.

### 🔶 Important Notes:

- Each **shard is independent** – it stores different data from the others.
- There is **no shared data** between shards.
- Sharding helps systems handle **more users and more data** without slowing down.
- It is a popular technique used in **big systems** like Facebook, Twitter, or databases like MongoDB and MySQL.

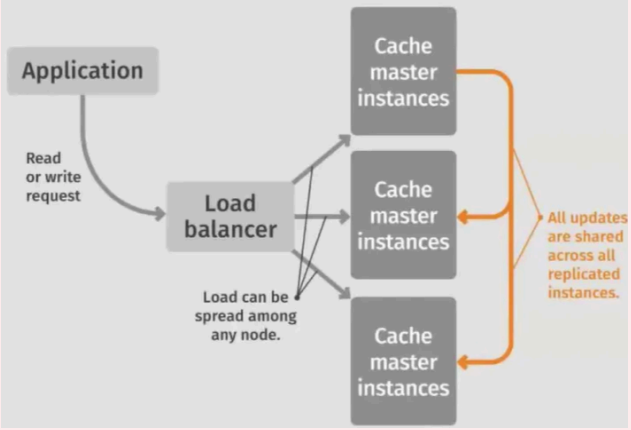
### ✅ Why use Sharding?

- To **improve performance** by spreading the load.
- To **scale horizontally** (add more machines as needed).
- To **reduce database bottlenecks**.



## 🔷 Horizontal Scaling Techniques – Active-Active (Multi-Master)

This is a **horizontal scaling method** where **multiple master instances** are active at the same time and handle both **read** and **write** requests.




### 🔶 How it works (based on the image):

1. The **Application** sends a **read or write request**.
2. The **Load Balancer** receives the request and distributes it to one of the available **Cache Master Instances**.
   - The load balancer can send traffic to **any instance**.
3. Each **Cache Master Instance** can **read or write** data.
4. When a write happens, the **update is shared across all other master instances** to keep them in sync.

### 🔶 Challenges:

- **Write conflict**:

  Since multiple masters can write at the same time, **conflicts** can happen if two nodes try to change the same data.

- **Data lag**:

  It takes time to **synchronize updates** between all instances, which can cause a **delay** in getting the most up-to-date data.

### 🔶 Use Case:

- This approach is common in **enterprise systems** that use **Active-Active geo-distributed architecture**, where different data centers in different regions are all active and handling traffic.

### ✅ Why use this approach?

- Supports **high availability** – if one node goes down, others still work.
- Good for **distributed systems** across regions.
- Can **scale horizontally** by adding more master nodes.

📒 ممكن يبانوا شبه بعض، لكن في فرق واضح وبسيط بينهم. خليني أوضح **Shared** والـ **Active-Active** ـ

### 🔷 أولاً: Active-Active (Multi-Master)

- **كل Node (أو سيرفر)** بيقدر يقرأ ويكتب (Read & Write).
- **البيانات بتتكرر وتتزامن** بين كل الـ nodes.
- لو فيه **تحديث (write)** على Node 1، لازم يوصل نفس التحديث لـ Node 2 و Node 3.
- يحاول يكتب على نفس البيانات في نفس الوقت Node لما أكتر من **(Write Conflict)** يحصل أحيانًا **تضارب في الكتابة**.

🧠 **تخيل** إن عندك 3 موظفين عندهم نسخة من نفس الملف، وكل واحد يقدر يعدل عليه. لو اتنين عدلوا نفس السطر، يحصل تضارب ولازم نحلّه.

### 🔶 مثال توضيحي:

- عندك 3 قواعد بيانات في أماكن مختلفة.
- كل واحدة منهم نشطة وبتتعامل مع المستخدمين.
- التحديثات لازم تتزامن بينهم كلهم.

### 🔷 ثانيًا: Shared (Shared Storage أو Shared Database)

- لكن كلهم **بيشاركوا قاعدة بيانات واحدة فقط**، **(nodes)** فيه **سيرفرات متعددة**.
- **الكتابة والقراءة بتتم من نفس المكان**.
- مفيش تضارب في البيانات لأن كله بيكتب ويقرأ من نفس المصدر.
- **النقطة المشتركة الوحيدة** بينهم هي قاعدة البيانات أو الـ storage.

🧠 **تخيل** إن عندك 3 موظفين بيشتغلوا على نفس ملف Word على Google Drive. كل التعديلات بتروح للمكان نفسه، فمفيش نسخ مختلفة ولا تضارب.

### ✅ خلاصة الفرق ببساطة:

| | Active-Active | Shared Storage |
|---|---|---|
| **الكتابة** | كل Node يقدر يكتب | الكل بيكتب على نفس المكان |
| **البيانات** | nodes بتتكرر وتتزامن بين الـ | في مكان واحد مشترك |
| **التضارب** | Write Conflict ممكن يحصل | مفيش تضارب (كلهم على نفس المصدر) |
| **المرونة الجغرافية** | مناسبة للتوزيع الجغرافي (Geo) | أقل مرونة للتوزيع الجغرافي |

## ✅ What is cache consistency?

**Cache consistency** means that the data in the **cache** is the same (or synchronized) with the data in the **main database/storage**.

In other words:

- If someone updates data in the database, the cache should also reflect that update.
- The goal is to make sure **users are not reading old or outdated data** from the cache.

---

## ❌ How does a cache become inconsistent?

A **cache becomes inconsistent** when the data in the cache is **different from** the data in the main database.

This can happen when:

- The **database is updated**, but the cache is **not updated or invalidated**.
- There are **delays or failures** in syncing the cache after data changes.
- Multiple caches (in different regions or servers) are not synchronized with each other.

## 🔷 In-Memory Caching

In-memory caching stores data in the **RAM of the server**, allowing faster access compared to fetching it from a database every time.

It works best for:

- A **single server**, or
- **Multiple servers** using something called **sticky sessions**.

---

## 📌 Difference Between:

## ❌ Without Sticky Session

- The client sends requests, but they go to **different servers** (e.g., Server X for request 1, Server Y for request 2).
- Each server has its **own memory**, so:
  - Data cached on Server X won't be available on Server Y.
  - This can lead to **cache misses** or inconsistent data.

## ✅ With Sticky Session

- The client's requests are always routed to the **same server** (e.g., Server X).
- Since all requests go to Server X:
  - The cached data remains available throughout the session.
  - This ensures better performance and **consistent caching**.

---

## 💡 Summary:

- **Sticky session** = All requests go to the same server → In-memory cache works effectively.
- **No sticky session** = Requests go to different servers → Cache may not work properly across them.

## 🗃️ In-Memory Caching Implementation

### 🔶 Cache Guidelines:

1. **Don't depend on cached value to be available all the time**
   - Cache can be cleared, expire, or removed—so always have a fallback.
2. **Do not insert external input into the cache**
   - Avoid caching values directly from user input to prevent security issues like cache poisoning.
3. **Use expirations to limit cache growth**
   - Set absolute or sliding expiration to avoid the cache consuming too much memory.
4. **It's up to the developer to limit cache size**
   - .NET's memory cache doesn't enforce strict size limits; developers must manage this responsibly.

## 🧪 .NET Sample

Sure! Below is a **complete example in .NET (ASP.NET Core Razor Pages or MVC)** demonstrating **three ways** to use **in-memory caching**:

- Manually using `TryGetValue` and `Set`
- Using `GetOrCreate`
- Using `RegisterPostEvictionCallback` with `SetPriority`

---

### ✅ Startup Configuration

In `.NET 6+` (`Program.cs`):

```
var builder = WebApplication.CreateBuilder(args);

// Add services
builder.Services.AddMemoryCache(); // Required
builder.Services.AddRazorPages(); // or AddControllersWithViews()

var app = builder.Build();

app.UseRouting();
app.MapRazorPages(); // or MapDefaultControllerRoute()
app.Run();
```

---

### ✅ Model or Service using IMemoryCache

```
using Microsoft.Extensions.Caching.Memory;

public class MyCacheService
{
    private readonly IMemoryCache _cache;

    public MyCacheService(IMemoryCache cache)
```

```
{
    _cache = cache;
}

public string ManualCacheExample()
{
    string key = "manual_key";
    if (!_cache.TryGetValue(key, out string cachedValue))
    {
        cachedValue = $"Manual Cache at {DateTime.Now}";

        var options = new MemoryCacheEntryOptions()
            .SetAbsoluteExpiration(TimeSpan.FromMinutes(2))
            .SetPriority(CacheItemPriority.Normal);

        _cache.Set(key, cachedValue, options);
    }
    return cachedValue;
}

public string GetOrCreateExample()
{
    string key = "getorcreate_key";

    return _cache.GetOrCreate(key, entry =>
    {
        entry.AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(1);
        entry.Priority = CacheItemPriority.High;
        return $"GetOrCreate Cache at {DateTime.Now}";
    });
}

public string WithEvictionCallbackExample()
{
    string key = "eviction_key";

    if (!_cache.TryGetValue(key, out string value))
    {
        value = $"Eviction Callback Cache at {DateTime.Now}";

        var options = new MemoryCacheEntryOptions()
            .SetSlidingExpiration(TimeSpan.FromSeconds(30))
            .SetPriority(CacheItemPriority.Low)
            .RegisterPostEvictionCallback((k, v, reason, state) =>
            {
                Console.WriteLine($"Cache entry '{k}' was evicted because: {reason}");
            });

        _cache.Set(key, value, options);
    }

    return value;
```

```
    }
}
```

### ✅ Using the Service in a Razor Page or Controller

**Example Razor Page ( `Pages/CacheDemo.cshtml.cs` ):**

```
using Microsoft.AspNetCore.Mvc.RazorPages;

public class CacheDemoModel : PageModel
{
    private readonly MyCacheService _cacheService;

    public string ManualResult { get; private set; }
    public string GetOrCreateResult { get; private set; }
    public string EvictionCallbackResult { get; private set; }

    public CacheDemoModel(MyCacheService cacheService)
    {
        _cacheService = cacheService;
    }

    public void OnGet()
    {
        ManualResult = _cacheService.ManualCacheExample();
        GetOrCreateResult = _cacheService.GetOrCreateExample();
        EvictionCallbackResult = _cacheService.WithEvictionCallbackExample();
    }
}
```

**And register the service in `Program.cs` :**

```
builder.Services.AddSingleton<MyCacheService>();
```

### ✅ Sample Output in Razor Page (.cshtml)

```
<h2>Manual Cache: @Model.ManualResult</h2>
<h2>GetOrCreate Cache: @Model.GetOrCreateResult</h2>
<h2>Eviction Callback Cache: @Model.EvictionCallbackResult</h2>
```

## ✅ Distributed Cache - Redis

A **distributed cache** is a type of caching system where the cache is **shared across multiple application servers** instead of being stored locally on a single server. One popular distributed cache system is **Redis**.

### 🔥 Benefits of Using Redis as a Distributed Cache

1. **Performance**

   Redis stores data in memory, which allows for **very fast** read/write operations. This improves the overall performance of your application, especially for frequently accessed data.

2. **Scalability**

   Because the cache is shared and not tied to one server, Redis makes it easier to **scale your app horizontally** (adding more servers).

3. **Consistency**

   Redis helps ensure that all app servers access the **same cached data**, which leads to consistent behavior regardless of which server handles the request.

4. **Survives Server Restarts and App Deployments**

   Since Redis runs independently of your application, the cached data can **persist** through application restarts and new deployments (if configured to do so), unlike in-memory caching.

### 🧪 .NET Example

Here's a **full working .NET 7+ Razor Pages example** using **Redis as a distributed cache**. This setup will:

- Connect your app to Redis.
- Store and retrieve data from the Redis cache.
- Show how caching works on a Razor Page.

## ✅ Step-by-Step Setup: Redis + .NET (Razor Pages)

### 1️⃣ Install Required Package

In your terminal, run:

```
dotnet add package Microsoft.Extensions.Caching.StackExchangeRedis
```

### 2️⃣ Configure Redis in `Program.cs`

```
using Microsoft.Extensions.Caching.Distributed;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddRazorPages();

// Configure Redis Distributed Cache
builder.Services.AddStackExchangeRedisCache(options =>
{
```

```
    options.Configuration = "localhost:6379"; // Use your Redis server's connection string
    options.InstanceName = "MyRedisInstance:";
});

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

app.MapRazorPages();

app.Run();
```

> 🔁 Make sure Redis is running on your machine. If you're using Docker, run:
>
> ```
> docker run -d -p 6379:6379 redis
> ```

---

### 3️⃣ Create a Razor Page `Pages/RedisTest.cshtml.cs`

```
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Extensions.Caching.Distributed;
using System.Text;

namespace YourAppNamespace.Pages
{
    public class RedisTestModel : PageModel
    {
        private readonly IDistributedCache _cache;

        public string CachedMessage { get; set; }

        public RedisTestModel(IDistributedCache cache)
        {
            _cache = cache;
        }

        public async Task OnGetAsync()
        {
            // Try to get from cache
            var cachedValue = await _cache.GetAsync("myMessage");

            if (cachedValue == null)
            {
```

```
            // If not found in cache, set it
            var message = "Hello from Redis at " + DateTime.Now;
            var encodedMessage = Encoding.UTF8.GetBytes(message);

            var options = new DistributedCacheEntryOptions()
                .SetAbsoluteExpiration(TimeSpan.FromMinutes(1));

            await _cache.SetAsync("myMessage", encodedMessage, options);

            CachedMessage = "Cached New Message: " + message;
        }
        else
        {
            // Found in cache
            CachedMessage = "Retrieved from Cache: " + Encoding.UTF8.GetString(cachedValue);
        }
    }
  }
}
```

### 4️⃣ Create Razor View `Pages/RedisTest.cshtml`

```
@page
@model YourAppNamespace.Pages.RedisTestModel
@{
  ViewData["Title"] = "Redis Cache Test";
}

<h1>Redis Cache Example</h1>
<p>@Model.CachedMessage</p>
```

## ✅ Final Result

- Visit `/RedisTest` in your browser.
- On the first load, it caches the message.
- On refresh, it retrieves the same message from Redis until the cache expires (1 min in this example).

## ✅ Distributed Cache – Types (Summary)

These are different types of distributed caching systems used in .NET applications:

1. **Distributed Redis Cache**
   - ⚡ Highly recommended by Microsoft
   - Fast, reliable, and supports cross-platform apps
   - Works great in cloud and microservices environments
2. **Distributed Memory Cache**
   - Stores cache in memory but shared across multiple nodes only if properly configured
   - Not suitable for large-scale distributed systems
   - Good for simple setups or testing
3. **Distributed NCache**
   - A commercial, in-memory distributed caching solution
   - Offers advanced features and high performance
   - Works well in enterprise-level .NET applications
4. **Distributed SQL Server Cache**
   - Stores cache data in a SQL Server database
   - Easy to set up in existing SQL-based infrastructures
   - Slower than Redis or NCache but reliable for shared caching in small to medium applications

## 🔄 What is Cache Eviction?

When your system stores data in a **cache** (like temporary memory), it has limited space.

So, when it's full and new data needs to come in, the system must **remove some old data** to make space.

This process is called **Eviction**, and there are different **strategies (rules)** for deciding *which* data to remove.

## 📋 Types of Eviction Strategies (With Simple Examples):

### 1. Least Recently Used (LRU)

👉 *"Remove the one we haven't used for the longest time."*

Example:

Imagine your phone keeps 3 apps open. You used:

- Facebook (5 mins ago)
- WhatsApp (2 mins ago)
- Instagram (Just now)

If you now open a 4th app (YouTube), and memory is full, the phone will **close Facebook** because it was used the longest time ago.

### 2. Least Frequently Used (LFU)

👉 *"Remove the one we use the least often."*

Example:

You open these apps today:

- Instagram (5 times)
- WhatsApp (10 times)
- Twitter (1 time)

When you need to make space, the phone will remove **Twitter**, since you used it the least.

### 3. Window TinyLFU (W-TinyLFU)

👉 *A smart combination of LRU and LFU.*

It looks at **recent** usage *and* **frequent** usage to decide what to remove.

This strategy gives better performance in many systems.

### 4. Time To Live (TTL)

👉 *"Remove it after a certain time, even if it's used a lot."*

Example:

Your cache says: "Store data for only 5 minutes."

Even if a file is used again and again, it will be deleted after 5 minutes.

## ⚙️ Approximation Algorithm

👉 These are **faster but less accurate** ways to apply these strategies.

Instead of tracking every detail, they make **quick guesses** to save time and system resources.

## 🔥 What are Eviction Policies in Redis?

Redis stores data in memory (RAM).

But RAM has a limit — when Redis uses all the memory you allowed it (like 100 MB), it has to **remove some old data** to make space for new data.

This is called **"eviction"**, and the way Redis chooses what to remove is called the **eviction policy**.

## 🛠️ Example of Setting Memory Limit

In the Redis config file ( `redis.conf` ), you can say:

```
maxmemory 100mb
```

That tells Redis: "You're allowed to use up to 100 MB."

## 🤔 What happens when Redis reaches 100 MB?

Redis looks at the **eviction policy** you set and decides **what to remove** to free space.

## 📜 Types of Eviction Policies

Let's break down each one **in simple words**:

### 1. `allkeys-lru`

- Removes **any key** (with or without expiration).
- Chooses the one that was **used the least recently**.
- Think of it like: "I haven't used this in a while, let's remove it."

### 2. `volatile-lru`

- Only removes keys that have an **expiration time (TTL)**.
- Also chooses the **least recently used** key among them.

### 3. `allkeys-lfu`

- Removes **any key**, just like `allkeys-lru` .
- But chooses the one that is **used the least often**.
- LFU = Least Frequently Used.

### 4. `volatile-lfu`

- Same as above, but only for **keys with TTL**.
- Removes the one **used the least often**.

### 5. `volatile-ttl`

- Removes the key that will **expire soonest**.
- Only looks at keys with TTL.
- Example: If one key will expire in 1 second and another in 1 hour, it will remove the one expiring in 1 second.

### 6. noeviction

- Redis **does not remove anything**.
- If memory is full and you try to add more data, Redis gives you an **error**.
- Use this if you want full control and **never want Redis to delete data automatically**.

---

## 🧠 Summary Table (Super Simple)

| Policy | Removes Any Key? | Uses TTL? | Chooses Key Based On |
|---|---|---|---|
| allkeys-lru | ✅ Yes | ❌ No | Least recently used |
| volatile-lru | ❌ No | ✅ Yes | Least recently used |
| allkeys-lfu | ✅ Yes | ❌ No | Least frequently used |
| volatile-lfu | ❌ No | ✅ Yes | Least frequently used |
| volatile-ttl | ❌ No | ✅ Yes | Soonest to expire |
| noeviction | ❌ No | ❌ No | ❌ No deletion (error) |

---

## 🚀 What is RediSearch?

RediSearch is a tool that adds **advanced search features** to **Redis** (which is a fast data store).

### 🧠 Think of it like this:

Normally, Redis is super fast for saving and getting data by key. But what if you want to **search inside the data** — like search for words, phrases, or similar text? Redis alone can't do that.

That's where **RediSearch** comes in!

---

### 💡 What can RediSearch do?

- ✅ **Let you search your data** using full text, just like Google search.
- ✅ You can **create indexes** to make searching super fast.
- ✅ You can use its **own query language** to ask questions like:
  - "Find items where the name includes 'apple'"
- ✅ It uses **compressed, inverted indexes** → meaning: searches are fast and don't use much memory.
- ✅ Supports:
  - 🔍 **Exact phrase search**: matches exact words like `"red apple"`
  - 🤖 **Fuzzy search**: finds results even with spelling mistakes (`appl` → `apple`)
  - ✨ **Autocomplete**: shows suggestions as you type (like Google search box)
- ✅ For **C# developers**, you can use the **NRediSearch** package to easily work with it in your code.

---

### 🎯 Why use RediSearch?

If you want to **search your Redis data like a search engine**, RediSearch is the best and fastest way.

## "Installing RediSearch using Docker"

And it shows this Docker command:

```
docker run -d --name redis-Search -p 6379:6379 redis/redis-stack:latest
```

---

# 📂 Creating Index in Redis (RediSearch)

## 🔷 Basic Redis Commands:

- `HSET`, `HGET`, `HMGET` — Used to insert and retrieve hash keys/fields.

## 🔷 When creating an index, you define:

1. **Which data** to index (e.g., all hashes with a key starting with `movie:`).
2. **Which fields** to index within those hashes (schema definition).

## 🔷 Create Index Command:

```
FT.CREATE idx:movie ON hash PREFIX 1 "movie:" SCHEMA title TEXT SORTABLE
release_year NUMERIC SORTABLE rating NUMERIC SORTABLE  genre TAG SORTABLE
```

## 🧾 Explanation:

- `idx:movie` → name of the index.
- `ON hash` → index is on Redis hashes.
- `PREFIX 1 "movie:"` → only hashes with keys starting with `"movie:"`.
- `SCHEMA` → defines the fields and their types:
  - `title` → `TEXT`
  - `release_year` and `rating` → `NUMERIC`
  - `genre` → `TAG`
- `SORTABLE` allows you to sort by this field in search queries.

## 🔎 To View Index Info:

```
FT.INFO idx:movie
```

This command gives metadata about the created index.

```
# redis-cli
127.0.0.1:6379> HSET movie:11002 title "Star Wars: Episode V - The Empire Strikes Back" plot "After the Rebel
s are brutally overpowered by the Empire on the ice planet Hoth, Luke Skywalker begins Jedi training with Yod
a, while his friends are pursued by Darth Vader and a bounty hunter named Boba Fett all over the galaxy." rele
ase_year 1980 genre "Action" rating 8.7 votes 1127635 imdb_id tt0080684
(integer) 7
127.0.0.1:6379>
127.0.0.1:6379> HSET movie:11003 title "The Godfather" plot "The aging patriarch of an organized crime dyna
sty transfers control of his clandestine empire to his reluctant son." release_year 1972 genre "Drama" rating 9.
2 votes 1563839 imdb_id tt0068646
(integer) 7
127.0.0.1:6379>
127.0.0.1:6379> HSET movie:11004 title "Heat" plot "A group of professional bank robbers start to feel the heat
from police when they unknowingly leave a clue at their latest heist." release_year 1995 genre "Thriller" rating
8.2 votes 559490 imdb_id tt0113277
```

```
(integer) 7
127.0.0.1:6379>
127.0.0.1:6379> HSET movie:11005 title "Star Wars: Episode VI - Return of the Jedi" plot "The Rebels dispatch
to Endor to destroy the second Empire's Death Star." release_year 1983 genre "Action" rating 8.3 votes 90626
0 imdb_id tt0086190
(integer) 7
127.0.0.1:6379> FT.CREATE idx:movie ON hash PREFIX 1 "movie:" SCHEMA title TEXT SORTABLE release_year
NUMERIC SORTABLE  rating NUMERIC SORTABLE genre TAG SORTABLE
OK
127.0.0.1:6379> ft.info idx:movie
 1) index_name
 2) idx:movie
 3) index_options
 4) (empty array)
 5) index_definition
 6) 1) key_type
    2) HASH
    3) prefixes
    4) 1) movie:
    5) default_score
    6) "1"
 7) attributes
 8) 1) 1) identifier
       2) title
       3) attribute
       4) title
       5) type
       6) TEXT
       7) WEIGHT
       8) "1"
       9) SORTABLE
    2) 1) identifier
       2) release_year
       3) attribute
       4) release_year
       5) type
       6) NUMERIC
       7) SORTABLE
       8) UNF
    3) 1) identifier
       2) rating
       3) attribute
       4) rating
       5) type
       6) NUMERIC
       7) SORTABLE
       8) UNF
    4) 1) identifier
       2) genre
       3) attribute
       4) genre
       5) type
       6) TAG
       7) SEPARATOR
```

```
      8) ,
      9) SORTABLE
 9) num_docs
10) (integer) 4
11) max_doc_id
12) (integer) 4
13) num_terms
14) (integer) 17
15) num_records
16) (integer) 34
17) inverted_sz_mb
18) "0.0021753311157226563"
19) vector_index_sz_mb
20) "0"
21) total_inverted_index_blocks
22) (integer) 22
23) offset_vectors_sz_mb
24) "2.09808349609375e-5"
25) doc_table_size_mb
26) "2.93731689453125e-4"
27) sortable_values_size_mb
28) "4.901885986328125e-4"
29) key_table_size_mb
30) "1.3828277587890625e-4"
31) tag_overhead_sz_mb
32) "8.296966552734375e-5"
33) text_overhead_sz_mb
34) "5.674362182617188e-4"
35) total_index_memory_sz_mb
36) "0.0038547515869140625"
37) geoshapes_sz_mb
38) "0"
39) records_per_doc_avg
40) "8.5"
41) bytes_per_record_avg
42) "67.0882339477539"
43) offsets_per_term_avg
44) "0.6470588445663452"
45) offset_bits_per_record_avg
46) "8"
47) hash_indexing_failures
48) (integer) 0
49) total_indexing_time
50) "0.6240000128746033"
51) indexing
52) (integer) 0
53) percent_indexed
54) "1"
55) number_of_uses
56) (integer) 1
57) cleaning
58) (integer) 0
59) gc_stats
60)  1) bytes_collected
```

```
    2) "0"
    3) total_ms_run
    4) "0"
    5) total_cycles
    6) "0"
    7) average_cycle_time_ms
    8) "nan"
    9) last_run_time_ms
   10) "0"
   11) gc_numeric_trees_missed
   12) "0"
   13) gc_blocks_denied
   14) "0"
61) cursor_stats
62) 1) global_idle
    2) (integer) 0
    3) global_total
    4) (integer) 0
    5) index_capacity
    6) (integer) 128
    7) index_total
    8) (integer) 0
63) dialect_stats
64) 1) dialect_1
    2) (integer) 0
    3) dialect_2
    4) (integer) 0
    5) dialect_3
    6) (integer) 0
    7) dialect_4
    8) (integer) 0
65) Index Errors
66) 1) indexing failures
    2) (integer) 0
    3) last indexing error
    4) N/A
    5) last indexing error key
    6) "N/A"
67) field statistics
68) 1) 1) identifier
       2) title
       3) attribute
       4) title
       5) Index Errors
       6) 1) indexing failures
          2) (integer) 0
          3) last indexing error
          4) N/A
          5) last indexing error key
          6) "N/A"
    2) 1) identifier
       2) release_year
       3) attribute
       4) release_year
```

```
      5) Index Errors
      6) 1) indexing failures
         2) (integer) 0
         3) last indexing error
         4) N/A
         5) last indexing error key
         6) "N/A"
   3) 1) identifier
      2) rating
      3) attribute
      4) rating
      5) Index Errors
      6) 1) indexing failures
         2) (integer) 0
         3) last indexing error
         4) N/A
         5) last indexing error key
         6) "N/A"
   4) 1) identifier
      2) genre
      3) attribute
      4) genre
      5) Index Errors
      6) 1) indexing failures
         2) (integer) 0
         3) last indexing error
         4) N/A
         5) last indexing error key
         6) "N/A"
127.0.0.1:6379>
```

### 🔍 General Purpose:

These queries are used to **search and filter data** stored in Redis using the **RediSearch module**, specifically from an index called `idx:movie`.

---

### ✅ Query Breakdown with Explanation:

1. `FT.SEARCH idx:movie "war"`
   - Searches for the keyword `"war"` **anywhere** in indexed fields.
   - Returns all matching documents.
2. `FT.SEARCH idx:movie "war" RETURN 2 title release_year`
   - Same as above, **but only returns** the `title` and `release_year` fields.
   - `RETURN 2` means you're returning **2 fields**.
3. `FT.SEARCH idx:movie "@title:war" RETURN 2 title release_year`
   - Searches for `"war"` **only in the** `title` **field**.
4. `FT.SEARCH idx:movie 'war -jedi' RETURN 2 title release_year`
   - Searches for documents that contain `"war"` **but NOT** `"jedi"`.
5. `FT.SEARCH idx:movie "%gdfather%" RETURN 2 title release_year`
   - This one is trying to use a wildcard `%`, **but this is invalid in RediSearch**.
   - RediSearch doesn't support `%` like SQL — instead, use full-text matching or  for **prefix matching**.
6. `FT.SEARCH idx:movie "@genre:{Thriller}" RETURN 2 title release_year`
   - Searches for movies with **exact genre = Thriller**.
   - Curly braces `{}` are used for **tag fields**.
7. `FT.SEARCH idx:movie "@genre:{Thriller|Action}" RETURN 3 title release_year genre`
   - Searches for movies where genre is **either Thriller OR Action**.
   - `RETURN 3` returns `title`, `release_year`, and `genre`.
8. `FT.SEARCH idx:movie * FILTER release_year 1970 1980 RETURN 2 title release_year`
   - Finds all movies where `release_year` is **between 1970 and 1980** (inclusive).
   -  means search for all documents, and `FILTER` applies a numeric filter.
9. `FT.SEARCH idx:movie "@release_year:[1970 1980]" RETURN 2 title release_year`
   - Another way to filter by **numeric range**, using square brackets.
   - Searches where `release_year` is **between 1970 and 1980**.
10. `FT.SEARCH idx:movie "@release_year:[1970 (1980)" RETURN 2 title release_year`

- Similar to the one above, but `(` means **exclude** the upper bound (1980 not included).
- So it searches for years **>= 1970 and < 1980**.

---

### 🔑 Summary (Key Learnings):

| Feature | Syntax Example | Explanation |
|---|---|---|
| Simple Search | `"war"` | Matches "war" in any field |
| Field Search | `"@title:war"` | Matches "war" only in `title` |
| Exclusion | `"war -jedi"` | Includes "war", excludes "jedi" |

| | | |
|---|---|---|
| Tag Filter | `"@genre:{Thriller}"` | Exact match for tags |
| OR in Tags | `"@genre:{Thriller Action}"` | |
| Numeric Filter | `FILTER release_year 1970 1980` or `@release_year:[1970 1980]` | Range filter for numbers |
| Return Fields | `RETURN 2 title release_year` | Limit output fields |

Commands:

```
# redis-cli
127.0.0.1:6379>  HSET movie:11002 title "Star Wars: Episode V - The Empire Strikes Back" plot "After the Rebels are brutally overpowered by the Empire on the ice planet Hoth, Luke Skywalker begins Jedi training with Yoda, while his friends are pursued by Darth Vader and a bounty hunter named Boba Fett all over the galaxy." release_year 1980 genre "Action" rating 8.7 votes 1127635 imdb_id tt0080684
(integer) 7
127.0.0.1:6379> HSET movie:11003 title "The Godfather" plot "The aging patriarch of an organized crime dynasty transfers control of his clandestine empire to his reluctant son." release_year 1972 genre "Drama" rating 9.2 votes 1563839 imdb_id tt0068646
(integer) 7
127.0.0.1:6379> HSET movie:11004 title "Heat" plot "A group of professional bank robbers start to feel the heat from police when they unknowingly leave a clue at their latest heist." release_year 1995 genre "Thriller" rating 8.2 votes 559490 imdb_id tt0113277
(integer) 7
127.0.0.1:6379>  HSET movie:11005 title "Star Wars: Episode VI - Return of the Jedi" plot "The Rebels dispatch to Endor to destroy the second Empire's Death Star." release_year 1983 genre "Action" rating 8.3 votes 906260 imdb_id tt0086190
(integer) 7
127.0.0.1:6379>  FT.CREATE idx:movie ON hash PREFIX 1 "movie:" SCHEMA title TEXT SORTABLE release_year NUMERIC SORTABLE  rating NUMERIC SORTABLE genre TAG SORTABLE
OK
127.0.0.1:6379> ft.search idx:movie "war"
1) (integer) 2
2) "movie:11002"
3)  1) "genre"
    2) "Action"
    3) "rating"
    4) "8.7"
    5) "release_year"
    6) "1980"
    7) "plot"
    8) "After the Rebels are brutally overpowered by the Empire on the ice planet Hoth, Luke Skywalker begins Jedi training with Yoda, while his friends are pursued by Darth Vader and a bounty hunter named Boba Fett all over the galaxy."
    9) "imdb_id"
   10) "tt0080684"
   11) "title"
   12) "Star Wars: Episode V - The Empire Strikes Back"
   13) "votes"
   14) "1127635"
4) "movie:11005"
5)  1) "genre"
    2) "Action"
    3) "rating"
    4) "8.3"
    5) "release_year"
```

```
   6) "1983"
   7) "plot"
   8) "The Rebels dispatch to Endor to destroy the second Empire's Death Star."
   9) "imdb_id"
  10) "tt0086190"
  11) "title"
  12) "Star Wars: Episode VI - Return of the Jedi"
  13) "votes"
  14) "906260"
127.0.0.1:6379> ft.search idx:movie "war" return 2 title rlease_yearr
1) (integer) 2
2) "movie:11002"
3) 1) "title"
   2) "Star Wars: Episode V - The Empire Strikes Back"
4) "movie:11005"
5) 1) "title"
   2) "Star Wars: Episode VI - Return of the Jedi"
127.0.0.1:6379> ft.search idx:movie "war" return 2 title release_year
1) (integer) 2
2) "movie:11002"
3) 1) "title"
   2) "Star Wars: Episode V - The Empire Strikes Back"
   3) "release_year"
   4) "1980"
4) "movie:11005"
5) 1) "title"
   2) "Star Wars: Episode VI - Return of the Jedi"
   3) "release_year"
   4) "1983"
127.0.0.1:6379> ft.search idx:movie "@title:war" return 2 title release_year
1) (integer) 2
2) "movie:11002"
3) 1) "title"
   2) "Star Wars: Episode V - The Empire Strikes Back"
   3) "release_year"
   4) "1980"
4) "movie:11005"
5) 1) "title"
   2) "Star Wars: Episode VI - Return of the Jedi"
   3) "release_year"
   4) "1983"
127.0.0.1:6379> ft.search idx:movie ":war-jedi" return 2 title release_year
(error) Syntax error at offset 0 near
127.0.0.1:6379> ft.search idx:movie "war-jedi" return 2 title release_year
1) (integer) 1
2) "movie:11002"
3) 1) "title"
   2) "Star Wars: Episode V - The Empire Strikes Back"
   3) "release_year"
   4) "1980"
127.0.0.1:6379> ft.search idx:movie "gdfather" return 2 title release_year
1) (integer) 0
127.0.0.1:6379> ft.search idx:movie "godfather" return 2 title release_year
1) (integer) 1
```

```
2) "movie:11003"
3) 1) "title"
   2) "The Godfather"
   3) "release_year"
   4) "1972"
127.0.0.1:6379> ft.search idx:movie "%gdfather%" return 2 title release_year
1) (integer) 1
2) "movie:11003"
3) 1) "title"
   2) "The Godfather"
   3) "release_year"
   4) "1972"
127.0.0.1:6379> ft.search idx:movie "genre:{thriller}" return 2 title release_year
(error) Syntax error at offset 5 near genre
127.0.0.1:6379> ft.search idx:movie "@genre:{thriller}" return 2 title release_year
1) (integer) 1
2) "movie:11004"
3) 1) "title"
   2) "Heat"
   3) "release_year"
   4) "1995"
127.0.0.1:6379> ft.search idx:movie "@genre:{thriller}" return 2 title release_year genre
(error) Unknown argument `genre` at position 5 for <main>
127.0.0.1:6379> ft.search idx:movie "@genre:{thriller}" return 3 title release_year genre
1) (integer) 1
2) "movie:11004"
3) 1) "title"
   2) "Heat"
   3) "release_year"
   4) "1995"
   5) "genre"
   6) "Thriller"
127.0.0.1:6379> ft.search idx:movie "@genre:{thriller|Action}" return 3 title release_year genre
1) (integer) 3
2) "movie:11004"
3) 1) "title"
   2) "Heat"
   3) "release_year"
   4) "1995"
   5) "genre"
   6) "Thriller"
4) "movie:11002"
5) 1) "title"
   2) "Star Wars: Episode V - The Empire Strikes Back"
   3) "release_year"
   4) "1980"
   5) "genre"
   6) "Action"
6) "movie:11005"
7) 1) "title"
   2) "Star Wars: Episode VI - Return of the Jedi"
   3) "release_year"
   4) "1983"
   5) "genre"
```

```
   6) "Action"
127.0.0.1:6379> ft.search idx:movie *filter release_year 1970 1980 return 3 title release_year genre
(error) Unknown argument `release_year` at position 1 for <main>
127.0.0.1:6379> ft.search idx:movie*filter release_year 1970 1980 return 3 title release_year genre
(error) Unknown argument `1970` at position 1 for <main>
127.0.0.1:6379> ft.search idx:movie*filterrelease_year 1970 1980 return 3 title release_year genre
(error) Unknown argument `1980` at position 1 for <main>
127.0.0.1:6379> ft.search idx:movie * filter release_year 1970 1980 return 3 title release_year genre
1) (integer) 2
2) "movie:11003"
3) 1) "title"
   2) "The Godfather"
   3) "release_year"
   4) "1972"
   5) "genre"
   6) "Drama"
4) "movie:11002"
5) 1) "title"
   2) "Star Wars: Episode V - The Empire Strikes Back"
   3) "release_year"
   4) "1980"
   5) "genre"
   6) "Action"
127.0.0.1:6379> ft.search idx:movie @release_year[1970:1980] return 3 title release_year genre
(error) Syntax error at offset 13 near release_year
127.0.0.1:6379> ft.search idx:movie @release_year[1970 1980] return 3 title release_year genre
(error) Unknown argument `1980]` at position 1 for <main>
127.0.0.1:6379> ft.search idx:movie @release_year:[1970 1980] return 3 title release_year genre
(error) Unknown argument `1980]` at position 1 for <main>
127.0.0.1:6379> ft.search idx:movie "@release_year:[1970 1980]" return 3 title release_year genre
1) (integer) 2
2) "movie:11003"
3) 1) "title"
   2) "The Godfather"
   3) "release_year"
   4) "1972"
   5) "genre"
   6) "Drama"
4) "movie:11002"
5) 1) "title"
   2) "Star Wars: Episode V - The Empire Strikes Back"
   3) "release_year"
   4) "1980"
   5) "genre"
   6) "Action"
127.0.0.1:6379> ft.search idx:movie "@release_year:[1970 (1980]" return 3 title release_year genre
1) (integer) 1
2) "movie:11003"
3) 1) "title"
   2) "The Godfather"
   3) "release_year"
   4) "1972"
   5) "genre"
```

```
   6) "Drama"
127.0.0.1:6379>
```

## 🛠️ Manage Index in RediSearch

### 🔶 FT._LIST

- Lists **all existing indexes**.
- Use this to see which indexes have been created.

---

### 🔶 FT.INFO "idx:movie"

- Returns **metadata** and **schema details** of the index `idx:movie`.
- Useful to inspect:
  - Fields
  - Document count
  - Memory usage
  - Indexing options

---

### 🔶 FT.ALTER idx:movie SCHEMA ADD plot TEXT WEIGHT 0.5

- **Modifies the schema** of an existing index by adding a new field:
  - Field name: `plot`
  - Field type: `TEXT`
  - Weight: `0.5` (used in ranking search results — lower means less importance)
- You can use this to **dynamically update your index schema** without recreating it.

---

### 🔶 FT.DROPINDEX idx:movie

- **Deletes the index** `idx:movie`.
- ⚠️ By default, this **does not delete the underlying documents**.
- To delete the documents too, use:

  `DD` stands for "Drop Documents".

```
FT.DROPINDEX idx:movie DD
```

---

## ✅ Summary Table

| Command | Purpose |
|---|---|
| `FT._LIST` | Show all indexes |
| `FT.INFO "idx:movie"` | Get details about a specific index |
| `FT.ALTER` | Add fields to an existing index schema |
| `FT.DROPINDEX` | Remove an index (and optionally its documents) |

---

```
127.0.0.1:6379> ft._list
1) idx:movie
127.0.0.1:6379> ft.info "idx:movie"
 1) index_name
```

```
 2) idx:movie
 3) index_options
 4) (empty array)
 5) index_definition
 6) 1) key_type
    2) HASH
    3) prefixes
    4) 1) movie:
    5) default_score
    6) "1"
 7) attributes
 8) 1) 1) identifier
       2) title
       3) attribute
       4) title
       5) type
       6) TEXT
       7) WEIGHT
       8) "1"
       9) SORTABLE
    2) 1) identifier
       2) release_year
       3) attribute
       4) release_year
       5) type
       6) NUMERIC
       7) SORTABLE
       8) UNF
    3) 1) identifier
       2) rating
       3) attribute
       4) rating
       5) type
       6) NUMERIC
       7) SORTABLE
       8) UNF
    4) 1) identifier
       2) genre
       3) attribute
       4) genre
       5) type
       6) TAG
       7) SEPARATOR
       8) ,
       9) SORTABLE
 9) num_docs
10) (integer) 4
11) max_doc_id
12) (integer) 4
13) num_terms
14) (integer) 17
15) num_records
16) (integer) 34
17) inverted_sz_mb
```

```
18) "0.0021753311157226563"
19) vector_index_sz_mb
20) "0"
21) total_inverted_index_blocks
22) (integer) 22
23) offset_vectors_sz_mb
24) "2.09808349609375e-5"
25) doc_table_size_mb
26) "2.93731689453125e-4"
27) sortable_values_size_mb
28) "4.901885986328125e-4"
29) key_table_size_mb
30) "1.3828277587890625e-4"
31) tag_overhead_sz_mb
32) "8.296966552734375e-5"
33) text_overhead_sz_mb
34) "5.674362182617188e-4"
35) total_index_memory_sz_mb
36) "0.0038547515869140625"
37) geoshapes_sz_mb
38) "0"
39) records_per_doc_avg
40) "8.5"
41) bytes_per_record_avg
42) "67.0882339477539"
43) offsets_per_term_avg
44) "0.6470588445663452"
45) offset_bits_per_record_avg
46) "8"
47) hash_indexing_failures
48) (integer) 0
49) total_indexing_time
50) "0.7689999938011169"
51) indexing
52) (integer) 0
53) percent_indexed
54) "1"
55) number_of_uses
56) (integer) 18
57) cleaning
58) (integer) 0
59) gc_stats
60)  1) bytes_collected
     2) "0"
     3) total_ms_run
     4) "0"
     5) total_cycles
     6) "0"
     7) average_cycle_time_ms
     8) "nan"
     9) last_run_time_ms
    10) "0"
    11) gc_numeric_trees_missed
    12) "0"
```

```
   13) gc_blocks_denied
   14) "0"
61) cursor_stats
62) 1) global_idle
   2) (integer) 0
   3) global_total
   4) (integer) 0
   5) index_capacity
   6) (integer) 128
   7) index_total
   8) (integer) 0
63) dialect_stats
64) 1) dialect_1
   2) (integer) 1
   3) dialect_2
   4) (integer) 0
   5) dialect_3
   6) (integer) 0
   7) dialect_4
   8) (integer) 0
65) Index Errors
66) 1) indexing failures
   2) (integer) 0
   3) last indexing error
   4) N/A
   5) last indexing error key
   6) "N/A"
67) field statistics
68) 1) 1) identifier
      2) title
      3) attribute
      4) title
      5) Index Errors
      6) 1) indexing failures
         2) (integer) 0
         3) last indexing error
         4) N/A
         5) last indexing error key
         6) "N/A"
   2) 1) identifier
      2) release_year
      3) attribute
      4) release_year
      5) Index Errors
      6) 1) indexing failures
         2) (integer) 0
         3) last indexing error
         4) N/A
         5) last indexing error key
         6) "N/A"
   3) 1) identifier
      2) rating
      3) attribute
      4) rating
```

```
      5) Index Errors
      6) 1) indexing failures
        2) (integer) 0
        3) last indexing error
        4) N/A
        5) last indexing error key
        6) "N/A"
   4) 1) identifier
      2) genre
      3) attribute
      4) genre
      5) Index Errors
      6) 1) indexing failures
        2) (integer) 0
        3) last indexing error
        4) N/A
        5) last indexing error key
        6) "N/A"
127.0.0.1:6379> ft.alter idx:movie schema add plot text weight 0.5
OK
127.0.0.1:6379> ft.dropindex idx:movie
OK
127.0.0.1:6379> ft.info idx:movie
(error) Unknown index name
127.0.0.1:6379>
```

### Definition of Aggregation:

**Aggregation** in RedisSearch refers to the process of computing **summaries or grouped information** from a set of documents based on specified fields. It allows you to perform operations like **counting, averaging, summing**, and **grouping** results — similar to SQL's `GROUP BY` and aggregate functions.

```
# redis-cli
127.0.0.1:6379> HSET "User:5978" "first_name" "Bel" "last_name" "Sowood" "email" "bsowoodp@about.me" "gender" "Female" "ip_address" "16.139.224.29" "country" "Indonesia" "country_code" "ID" "city" "Baturunggit-Kaja" "location" "115.5631,-8.2464" "last_login" "1569423153"
 "country_code" "ID" "city" "Pab(integer) 10
127.0.0.1:6379> HSET "User:5979" "first_name" "Franky" "last_name" "Le-Brum" "email" "flebruma@nasa.gov" "gender" "Male" "ip_address" "36.219.51.207" "country" "Germany" "country_code" "DE" "city" "Offenbach" "location" "8.81077219,50.080932" "last_login" "1582795277"
(integer) 10
127.0.0.1:6379> HSET "User:5980" "first_name" "Keith" "last_name" "Khan" "email" "khansanabr@google.cn" "gender" "Male" "ip_address" "97.11.152.180" "country" "Sweden" "country_code" "SE" "city" "Hägersten" "location" "17.99824435,59.287397" "last_login" "1575274044"
(integer) 10
127.0.0.1:6379> HSET "User:5981" "first_name" "Randle" "last_name" "Hallibone" "email" "rhalliboner@edublogs.org" "gender" "Female" "ip_address" "125.209.213.193" "country" "Colombia" "country_code" "CO" "city" "Manzanares" "location" "-75.1538099,5.392073" "last_login" "1590082825"
(integer) 10
127.0.0.1:6379> HSET "User:5982" "first_name" "Clevie" "last_name" "MacKeever" "email" "cmackeevernd@wunderground.com" "gender" "Female" "ip_address" "200.179.72.98" "country" "Iran" "country_code" "IR" "city" "Behbahān" "location" "50.234745,30.591158" "last_login" "1583990962"
(integer) 10
127.0.0.1:6379> HSET "User:5983" "first_name" "Bibb" "last_name" "Davidovic" "email" "bdavidovicire@mapquest.com" "gender" "Female" "ip_address" "92.203.243.239" "country" "Indonesia" "country_code" "ID" "city" "Rengasdengklok" "location" "106.7181959,-6.0847273" "last_login" "1583997170"
(integer) 10
127.0.0.1:6379> HSET "User:5984" "first_name" "Xena" "last_name" "Cortesi" "email" "xcortesisrfd@diigo.com" "gender" "Male" "ip_address" "152.12.18.131" "country" "Brazil" "country_code" "BR" "city" "Três-Lagoas" "location" "-51.8287077,-20.6121259" "last_login" "1595139856"
(integer) 10
127.0.0.1:6379> HSET "User:5985" "first_name" "Herron" "last_name" "Searton" "email" "hseartonng@mysql.com" "gender" "Male" "ip_address" "36.94.143.10" "country" "Poland" "country_code" "PL" "city" "Kiserionazu" "location" "19.0864556,52.255245" "last_login" "1577310658"
(integer) 10
127.0.0.1:6379> HSET "User:5986" "first_name" "Alice" "last_name" "Ivins" "email" "aivinsah@harvard.edu" "gender" "Female" "ip_address" "20.51.141.251" "country" "China" "country_code" "CN" "city" "Beimei" "location" "112.539988,34.78777" "last_login" "1575997415"
(integer) 10
127.0.0.1:6379> HSET "User:5987" "first_name" "Rosalynd" "last_name" "Clifft" "email" "rclifftri@nih.gov" "gender" "Female" "ip_address" "155.100.152.213" "country" "Estonia" "country_code" "EE" "city" "Tallinn" "location" "24.753547,59.437968" "last_login" "1592261044"
(integer) 10
127.0.0.1:6379> HSET "User:5988" "first_name" "Sterling" "last_name" "Plaister" "email" "splaisterrn@upenn.edu" "gender" "Male" "ip_address" "34.222.165.192" "country" "China" "country_code" "CN" "city" "Honghuabu" "location" "119.374297,27.139753" "last_login" "1592261044"
(integer) 10
```

```
127.0.0.1:6379> HSET "User:5989" "first_name" "Lilly" "last_name" "Sey" "email" "lseykr@myspace.com" "ge
nder" "Male" "ip_address" "208.109.135.29" "country" "Iran" "country_code" "IR" "city" "Oshnavīyeh" "locatio
n" "45.089217,37.036455" "last_login" "1588973256"
(integer) 10
127.0.0.1:6379> HSET "User:5990" "first_name" "Ezrah" "last_name" "Etolani" "email" "etolanilr@loyalsite.co
m" "gender" "Female" "ip_address" "183.148.228.183" "country" "Sweden" "country_code" "SE" "city" "Kharis
tān" "location" "44.126184,56.1938777" "last_login" "1588203777"
(integer) 10
127.0.0.1:6379> HSET "User:5991" "first_name" "Abdul" "last_name" "Ciccone" "email" "acicconemib@in.co
m" "gender" "Male" "ip_address" "131.76.34.252" "country" "Philippines" "country_code" "PH" "city" "Manas
a" "location" "120.2422184,15.5854665" "last_login" "1584366261"
(integer) 10
127.0.0.1:6379> HSET "User:5992" "first_name" "Zaria" "last_name" "Telsey" "email" "ztelseynr@bizjournals.c
om" "gender" "Female" "ip_address" "40.84.110.107" "country" "Indonesia" "country_code" "ID" "city" "Kadug
ede" "location" "108.0429974,-7.0297974" "last_login" "1589040771"
(integer) 10
127.0.0.1:6379> HSET "User:5993" "first_name" "Angela" "last_name" "Rawdales" "email" "arawdalesros@sci
entificamerican.com" "gender" "Female" "ip_address" "220.125.223.167" "country" "Indonesia" "country_cod
e" "ID" "city" "Bumbulan" "location" "122.0072825,0.5271852" "last_login" "1594583295"
(integer) 10
127.0.0.1:6379> HSET "User:5994" "first_name" "Bastian" "last_name" "Mathevon" "email" "bmathevonsup@u
sda.gov" "gender" "Male" "ip_address" "20.118.181.166" "country" "Indonesia" "country_code" "ID" "city" "Pab
uaruan" "location" "106.802879,6.4544294" "last_login" "1579064666"
(integer) 10
127.0.0.1:6379> HSET "User:5995" "first_name" "Cory" "last_name" "Eisentraut" "email" "ceisentrautqq@va.g
ov" "gender" "Male" "ip_address" "198.170.234.126" "country" "China" "country_code" "CN" "city" "Houmen"
"location" "119.25652,27.246947" "last_login" "1586283622"
(integer) 10
127.0.0.1:6379> HSET "User:5996" "first_name" "Marillin" "last_name" "Auger" "email" "maugerum@bing.co
m" "gender" "Female" "ip_address" "71.77.24.124" "country" "Philippines" "country_code" "PH" "city" "Mati"
"location" "126.3020575,6.7833779" "last_login" "1600007536"
(integer) 10
127.0.0.1:6379> PING "5996 users inserted"
"5996 users inserted"
127.0.0.1:6379> HSET "theater:99" name "The Secret Theatre" address "44-02 23rd St." city "Long Island Cit
y" zip "11101" phone "718 392-0722" url "http://www.secrettheatre.com/" location "-73.94482132078857,40.7
4809599748924"
(integer) 7
127.0.0.1:6379> HSET "theater:100" name "The Upright Citizens Brigade Theatre" address "307 W 26th Stree
t" city "New York" zip "10001" phone "212 366-9176" url "http://www.ucbtheatre.com/" location "-73.9973874
129749,40.747653594271554"
(integer) 7
127.0.0.1:6379> HSET "theater:101" name "The Walter Reade Theater at Lincoln Center" address "70 Lincoln C
enter Plaza" city "New York" zip "10023" phone "212 875-5600" url "http://www.filmlinc.com/wrt/wrt.html" lo
cation "-73.984238656361,40.774102169778494"
(integer) 7
127.0.0.1:6379> HSET "theater:102" name "The Zipper Theater" address "336 W 37th St" city "New York" zip
"10018" phone "212 563-0480" url "http://www.yelp.com/biz/the-zipper-theater-new-york" location "-73.994
03864237084,40.75467479483833"
(integer) 7
127.0.0.1:6379> HSET "theater:103" name "Theater For The New City" address "155 1st Avenue" city "New Yor
k" zip "10003" phone "212 254-1109" url "http://www.theaterforthenewcity.net/" location "-73.984980346087
3,40.728027870282"
(integer) 7
```

```
127.0.0.1:6379> HSET "theater:104" name "Theatre@ St. Clement's" address "423 West 46th Street" city "Ne
w York" zip "10036" phone "212 246-7277" url "http://www.stclementsnyc.org/" location "-73.9915711499504
9,40.76156669740397"
(integer) 7
127.0.0.1:6379> HSET "theater:105" name "Theatre 80 St Marks" address "80 Saint Marks Pl" city "New York"
zip "10003" phone "212 353-3321" url "http://www.theatermania.com/new-york/theaters/theater-80_3446/" l
ocation "-73.98575178980837,40.72762096909825"
(integer) 7
127.0.0.1:6379> HSET "theater:106" name "Theatre Row" address "410 W 42nd Street" city "New York" zip "10
036" phone "212 714-2442" url "http://www.theatrerow.org/" location "-73.9928423164134,40.75872836920
871"
(integer) 7
127.0.0.1:6379> HSET "theater:107" name "Theater St. Marks" address "94 St. Marks Place" city "New York" z
ip "10009" phone "212 777-6088" url "http://www.httheater.org/rentersUNDER.html" location "-73.98484849
05892,40.7271077482072"
(integer) 7
127.0.0.1:6379> HSET "theater:108" name "Union Square Theater" address "100 E 17th St" city "New York" zip
"10003" phone "212 505-0700" url "http://www.nytheatre.com/nytheatre/union.htm" location "-73.98786048
910895,40.7362666166269"
(integer) 7
127.0.0.1:6379> HSET "theater:109" name "Village Theater" address "158 Bleecker St" city "New York" zip "10
012" phone "212 253-0623" url "http://www.villagetheatre.org/" location "-74.00082225661741,40.728511024
1655"
(integer) 7
127.0.0.1:6379> HSET "theater:110" name "Vineyard Theater" address "108 E 15th St" city "New York" zip "100
03" phone "212 353-3366" url "http://www.vineyardtheatre.org/" location "-73.9880197165548,40.73469974
288967"
(integer) 7
127.0.0.1:6379> HSET "theater:111" name "Walkerspace Theater" address "46 Walker Street" city "New York" z
ip "10013" phone "212 868-4444" url "http://www.theateronline.com/venuebook.xzc?PK=238&view=hist" loc
ation "-74.00351520714296,40.7191031490749"
(integer) 7
127.0.0.1:6379> HSET "theater:112" name "WaMu Theater formerly Madison Square Garden" address "4 Penn
Plaza" city "New York" zip "10001" phone "212 465-6741" url "http://www.theateratmsg.com/" location "-73.9
9339518921358,40.75094725240502"
(integer) 7
127.0.0.1:6379> HSET "theater:113" name "Westside Theater" address "407 W 43rd St" city "New York" zip "1
0036" phone "212 315-2244" url "http://www.westsidetheatre.com/" location "-73.99525324238019,40.7591
634161395"
(integer) 7
127.0.0.1:6379> HSET "theater:114" name "Wings Theatre" address "154 Christopher St" city "New York" zip
"10014" phone "212 627-2960" url "http://www.wingstheatre.com/" location "-74.00889265661446,40.73299
33762212"
(integer) 7
127.0.0.1:6379> HSET "theater:115" name "Winter Garden Theatre" address "1634 Broadway" city "New York"
zip "10019" phone "212 944-3700" url "http://www.shubertorganization.com/theatres/winter_garden.asp" loc
ation "-73.98348163057112,40.76152464628083"
(integer) 7
127.0.0.1:6379> HSET "theater:116" name "York Theatre" address "619 Lexington Ave" city "New York" zip "10
022" phone "212 935-5820" url "http://w
127.0.0.1:6379> HSET "theater:117" name "Delacorte Theater" address "Central Park – Mid-Park at 80th Stree
t" city "New York" zip "0" phone "212 861-7277" url "http://www.centralpark.com/pages/attractions/delacorte
-theatre.html" location "-73.9688248050205,40.78017565353063"
(integer) 7
```

```
127.0.0.1:6379> HSET "movie:1124" title "Rome Wants Another Caesar" genre "Drama" votes 17 rating 7.0 relea
se_year 1974 plot "N/A" poster "N/A" imdb_id "tt0072094"
 Mi(integer) 8
127.0.0.1:6379> HSET "movie:1125" title "From Rome to Rollerball: The Full Circle" genre "Short" votes 24 ratin
g 5.3 release_year 1975 plot "N/A" poster "N/A" imdb_id "tt0322340"
V1(integer) 8
127.0.0.1:6379> HSET "movie:1126" title "Rome: The Other Side of Violence" genre "Action" votes 129 rating 6.
2 release_year 1976 plot "Four bandits kidnap a villa: the maid calls the police and Commissioner Ferreri imme
diately warns Commissioner Baldi, who orders the patrols to chase the four." poster "https://m.media-amazon.
com/images/M/MV5BYTBvK0tYOUTo9TGE4M2E0OS0Y2ML0iIGQtHmThIYlRhyN2QkKEYxKFcqGdeQXVyNzQ
zNDEy0Q@@._V1_SX300_.jpg" imdb_id "tt0200052"
(integer) 8
127.0.0.1:6379> HSET "movie:1127" title "Messalina, Empress and Whore" genre "Adventure" votes 156 rating
4.4 release_year 1977 plot "N/A" poster "https://m.media-amazon.com/images/M/MV5BNjk4NjM4NTgtM5B1L
5BaBNKx8rFZtTzLwKjZtNY0@@._V1_SY264_CR30,178,264,_.jpg" imdb_id "tt0078195"
(integer) 8
127.0.0.1:6379> HSET "movie:1128" title "Rome '78" genre "Drama" votes 163 rating 6.9 release_year 1978 plot
"N/A" poster "N/A" imdb_id "tt0102643"
TkzZW(integer) 8
127.0.0.1:6379> HSET "movie:1132" title "The Emperor of Rome" genre "Drama" votes 106 rating 7.1 release_ye
ar 1988 plot "N/A" poster "https://m.media-amazon.com/images/M/MV5BMGM2M2YzYGTnE25MEC0QkGxL0i
V0UzNTE2NzItC1kyKEYxKFcqGdeQXVyMzU0NzkwM@@._V1_SX300_.jpg" imdb_id "tt0210750"
(integer) 8
127.0.0.1:6379> HSET "movie:1133" title "Bastards" genre "Drama" votes 78 rating 6.2 release_year 1999 plot
"Two teenaged American-Asian brothers leave Vietnam and head for the USA to find their father and pursue t
he American dream." poster "https://m.media-amazon.com/images/M/MV5BMTA1M2EzN3F5BM1L5BaBNKx8r
FZtTCwMT0YDIYMQ@@._V1_UX182_CR0,0,182,268_AL_.jpg" imdb_id "tt0139882"
127.0.0.1:6379> HSET "movie:1134" title "The Insider" genre "Biography" votes 154624 rating 7.8 release_year
1999 plot "A research chemist comes under personal and professional attack when he decides to expose a 60
Minutes exposé on Big Tobacco." poster "https://m.media-amazon.com/images/M/MV5BODgyjA2N2QtODGfK
mNlY0EyKzZlTgNzYTIyYjU2WuTM2ZDDkL2ltYwlxKEYxKFcqGdeQXVyMTQxMzNDl0._V1_UY268_CR2,0,182,26
8_AL_.jpg" imdb_id "tt0140352"
(integer) 8
127.0.0.1:6379> HSET "movie:1135" title "Pulp Fiction" genre "Crime" votes 1743680 rating 8.9 release_year 19
94 plot "The lives of two mob hitmen, a boxer, a gangster and his wife, and a pair of diner bandits intertwine i
n four tales of violence and redemption." poster "https://m.media-ama
127.0.0.1:6379> HSET "movie:1137" title "Léon: The Professional" genre "Action" votes 984763 rating 8.5 relea
se_year 1994 plot "Mathilda, a 12-year-old girl, is reluctantly taken in by Léon, a professional assassin, after he
r family is murdered. An unusual relationship forms as she becomes his protégée and learns the assassin's tra
de." poster "https://m.media-amazon.com/images/M/MV5BZDAlMTY1NTgtOS00ZDNlLTNiYTYtZTk2Y2NkMTI
hYzY2XkEyXkFqcGdeQXVyNDYyMDk4NzM@._V1_UY268_CR0,0,182,268_AL_.jpg" imdb_id "tt0110413"
(integer) 8
127.0.0.1:6379> HSET "movie:1138" title "The Mask" genre "Comedy" votes 330393 rating 6.9 release_year 19
94 plot "Bank clerk Stanley Ipkiss is transformed into a manic superhero when he wears a mysterious mask."
poster "https://m.media-amazon.com/images/M/MV5BOWM2OTAxZWItNWNlNi00N2QzLWI4YzItMzk2YzM1O
WRiNmE2XkEyXkFqcGdeQXVyMTQxMzNDNl0._V1_UX182_CR0,0,182,268_AL_.jpg" imdb_id "tt0110475"
(integer) 8
127.0.0.1:6379> HSET "movie:1139" title "The Lord of the Rings: The Fellowship of the Ring" genre "Fantasy" v
otes 1500000 rating 8.8 release_year 2001 plot "A meek hobbit from the Shire and eight companions set out o
n a journey to destroy the powerful One Ring and save Middle-earth from the Dark Lord Sauron." poster "http
s://m.media-amazon.com/images/M/MV5BMGE2ZjY3N2MtMzU0ZGMtYWI0ZTktYzAyM2E4ZDYwZDJhXkEyX
kFqcGdeQXVyNzkwMjQ5NzM@._V1_UX182_CR0,0,182,268_AL_.jpg" imdb_id "tt0120737"
(integer) 8
127.0.0.1:6379> HSET "movie:1140" title "1917" genre "Drama" votes 209811 rating 7.9 release_year 2019 plot
```

```
"April 6th, 1917. As a regiment assembles to wage war deep in enemy territory, two soldiers are assigned to ra
ce against time and deliver a message that will stop 1600 men from walking straight into a deadly trap." poster
"https://m.media-amazon.com/images/M/MV5BOTdmNThjNDExY0E0Zjk3LkEtTGFjMTMtN2MzMTZhNzCwZjk
zKEYxKFcqGdeQXVyNDIzMzc1NjA@._V1_UY268_CR0,0,182,268_AL_.jpg" imdb_id "tt8579674"
(integer) 8
127.0.0.1:6379> HSET "movie:1141" title "Heat" genre "Drama" votes 506287 rating 8.3 release_year 1995 plot
"A group of professional bank robbers start to feel the heat from police when they unknowingly leave a clue at
their latest heist." poster "https://m.media-amazon.com/images/M/MV5BDM3JiWE5MTEtMDk2Mi00ZjczLTkz
ZWItYjA2NTMzZThhNzc0XkEyXkFqcGdeQXVyNDIzMzc1NjA@._V1_UY268_CR12,0,182,268_AL_.jpg" imdb_id
"tt0113277"
(integer) 8
127.0.0.1:6379> FT.CREATE idx:movie ON hash PREFIX 1 "movie:" SCHEMA title TEXT SORTABLE plot TEXT W
EIGHT 0.5 release_year NUMERIC SORTABLE rating NUMERIC SORTABLE votes NUMERIC SORTABLE genre T
AG SORTABLE
OK
127.0.0.1:6379> FT.CREATE idx:theater ON hash PREFIX 1 "theater:" SCHEMA name TEXT SORTABLE location
GEO
OK
127.0.0.1:6379> FT.CREATE idx:user ON hash PREFIX 1 "user:" SCHEMA gender TAG country TAG SORTABLE l
ast_login NUMERIC SORTABLE location GEO
OK
127.0.0.1:6379> FT.AGGREGATE "idx:movie" "*" GROUPBY 1 @release_year REDUCE COUNT 0 AS nb_of_movi
es
 1) (integer) 11
 2) 1) "release_year"
    2) "1977"
    3) "nb_of_movies"
    4) "1"
 3) 1) "release_year"
    2) "2019"
    3) "nb_of_movies"
    4) "1"
 4) 1) "release_year"
    2) "1995"
    3) "nb_of_movies"
    4) "1"
 5) 1) "release_year"
    2) "1978"
    3) "nb_of_movies"
    4) "1"
 6) 1) "release_year"
    2) "1974"
    3) "nb_of_movies"
    4) "1"
 7) 1) "release_year"
    2) "1999"
    3) "nb_of_movies"
    4) "2"
 8) 1) "release_year"
    2) "1988"
    3) "nb_of_movies"
    4) "1"
 9) 1) "release_year"
    2) "1975"
```

```
   3) "nb_of_movies"
   4) "1"
10) 1) "release_year"
   2) "2001"
   3) "nb_of_movies"
   4) "1"
11) 1) "release_year"
   2) "1976"
   3) "nb_of_movies"
   4) "1"
12) 1) "release_year"
   2) "1994"
   3) "nb_of_movies"
   4) "4"
127.0.0.1:6379> FT.AGGREGATE "idx:movie" "*" GROUPBY 1 @release_year REDUCE COUNT 0 AS nb_of_movi
es SORTBY 2 @release_year DESC
 1) (integer) 11
 2) 1) "release_year"
   2) "2019"
   3) "nb_of_movies"
   4) "1"
 3) 1) "release_year"
   2) "2001"
   3) "nb_of_movies"
   4) "1"
 4) 1) "release_year"
   2) "1999"
   3) "nb_of_movies"
   4) "2"
 5) 1) "release_year"
   2) "1995"
   3) "nb_of_movies"
   4) "1"
 6) 1) "release_year"
   2) "1994"
   3) "nb_of_movies"
   4) "4"
 7) 1) "release_year"
   2) "1988"
   3) "nb_of_movies"
   4) "1"
 8) 1) "release_year"
   2) "1978"
   3) "nb_of_movies"
   4) "1"
 9) 1) "release_year"
   2) "1977"
   3) "nb_of_movies"
   4) "1"
10) 1) "release_year"
   2) "1976"
   3) "nb_of_movies"
   4) "1"
11) 1) "release_year"
```

```
   2) "1975"
   3) "nb_of_movies"
   4) "1"
127.0.0.1:6379> FT.AGGREGATE idx:movie "*" GROUPBY 1 @genre REDUCE COUNT 0 AS nb_of_movies REDU
CE SUM 1 votes AS nb_of_votes REDUCE AVG 1 rating AS avg_rating SORTBY 4 @avg_rating DESC @nb_of_vo
tes DESC
1) (integer) 8
2) 1) "genre"
   2) "crime"
   3) "nb_of_movies"
   4) "1"
   5) "nb_of_votes"
   6) "1743680"
   7) "avg_rating"
   8) "8.9"
3) 1) "genre"
   2) "fantasy"
   3) "nb_of_movies"
   4) "1"
   5) "nb_of_votes"
   6) "1500000"
   7) "avg_rating"
   8) "8.8"
4) 1) "genre"
   2) "biography"
   3) "nb_of_movies"
   4) "1"
   5) "nb_of_votes"
   6) "154624"
   7) "avg_rating"
   8) "7.8"
5) 1) "genre"
   2) "drama"
   3) "nb_of_movies"
   4) "7"
   5) "nb_of_votes"
   6) "2458142"
   7) "avg_rating"
   8) "7.45714285714"
6) 1) "genre"
   2) "action"
   3) "nb_of_movies"
   4) "2"
   5) "nb_of_votes"
   6) "984892"
   7) "avg_rating"
   8) "7.35"
7) 1) "genre"
   2) "comedy"
   3) "nb_of_movies"
   4) "1"
   5) "nb_of_votes"
   6) "330393"
   7) "avg_rating"
```

```
   8) "6.9"
8) 1) "genre"
   2) "short"
   3) "nb_of_movies"
   4) "1"
   5) "nb_of_votes"
   6) "24"
   7) "avg_rating"
   8) "5.3"
9) 1) "genre"
   2) "adventure"
   3) "nb_of_movies"
   4) "1"
   5) "nb_of_votes"
   6) "156"
   7) "avg_rating"
   8) "4.4"
127.0.0.1:6379>
```

### 🧾 Functions Used

- `Year()` — extracts the **year** from a datetime.
- `MonthOfYear()` — extracts the **month number**.
- `DayOfWeek()` — extracts the **weekday** (e.g., Sunday = 1).

All are applied on the field `@last_login`.

---

### 📌 Commands Breakdown

1. `FT.Aggregate idx:user * APPLY year(@last_login) AS loginYear`
   - Extracts the **year** from `last_login`.
   - Names it `loginYear`.
2. `FT.Aggregate idx:user * APPLY MonthOfYear(@last_login) AS LoginMonth`
   - Extracts **month number** from `last_login`.
   - Names it `LoginMonth`.
3. `FT.Aggregate idx:user * APPLY DayOfWeek(@last_login) AS LoginDay`
   - Extracts **day of the week**.
   - Names it `LoginDay`.
4. `FT.AGGREGATE idx:user * APPLY year(@last_login) AS year APPLY "monthofyear(@last_login) + 1" AS month GROUPBY 2 @year @month REDUCE count 0 AS num_login SORTBY 4 @year ASC @month ASC`
   - **Extracts both year and month** from `last_login`.
   - Adds 1 to the month (possibly due to RedisSearch using 0-based months).
   - Groups by year and month.
   - Counts logins in each group and names it `num_login`.
   - Sorts results ascending by year and month.

---

### ✅ Summary

The `APPLY` keyword allows you to **create new fields dynamically** using functions applied to existing fields. These derived values can then be used in `GROUPBY`, `SORTBY`, or displayed in results.

### ❓ Question:

In the command

```
SORTBY 4 @year ASC @month ASC
```

why is the number **4** written after `SORTBY`?

---

### ✅ Answer:

The number **4** after `SORTBY` tells Redis how many arguments come after it.

Each sorting field needs **two arguments**:

- the field name (e.g., `@year`)
- the sort direction (`ASC` or `DESC`)

So in this case:

- `@year ASC` = 2 arguments
- `@month ASC` = 2 arguments
  Total =
  **4 arguments**

That's why you write `SORTBY 4`.

---

### 🔍 What does this Redis command do?

```
FT.AGGREGATE idx:user "@gender:{female}"
GROUPBY 1 @country
REDUCE COUNT 0 AS nb_of_users
FILTER "@country!='china' && @nb_of_users > 100"
SORTBY 2 @nb_of_users DESC
```

---

### 🧠 Step-by-step Explanation:

1. **Search female users only**

   `@gender:{female}` → Match only users who are female.

2. **Group the results by country**

   `GROUPBY 1 @country` → Group users by the `country` field.

3. **Count users in each country**

   `REDUCE COUNT 0 AS nb_of_users` → Count how many users per country, and name it `nb_of_users`.

4. **Filter the grouped results**

   `FILTER "@country!='china' && @nb_of_users > 100"`

   → Only include countries that are **not China** and have **more than 100 female users**.

5. **Sort the final result**

   `SORTBY 2 @nb_of_users DESC`

   → Sort by the number of users (`nb_of_users`) in **descending order** (largest first).

   The `2` means there are **2 arguments**: `@nb_of_users` and `DESC`.

---

### ✅ Summary in one line:

> Show all countries (except China) that have more than 100 female users, and sort them from most to least.

Let me know if you want this in Arabic too!

Examples in C# :

```
using NetTopologySuite.Index.HPRtree;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using NRedisStack;
using NRedisStack.RedisStackCommands;
using NRedisStack.Search;
using NRedisStack.Search.Aggregation;
using NRedisStack.Search.Literals.Enums;
using StackExchange.Redis;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace Redis_tester
{

    internal class Program
    {

        static void Main(string[] args)
        {
            ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost,allowAdmin=true");

            var endpoints=redis.GetEndPoints();

            var server = redis.GetServer(endpoints.FirstOrDefault());

            server.FlushAllDatabases();


            var db = redis.GetDatabase();

            var FT = db.FT();

            var json=db.JSON();

            filldata(db,FT,json);

            SearchbyName_age(db,FT);
            SearchbyName_city(db,FT);
            Searchbycity(db,FT);

            Console.ReadKey();
        }

        private static void filldata(IDatabase db, SearchCommands fT, JsonCommands json)
        {
            // Test data
            var user1 = new
            {
                name = "ahmed mohamed",
                email = "ahmed.mohamed@example.com",
```

```
            age = 42,
            city = "cairo"
        };

        var user2 = new
        {
            name = "ali mohamed",
            email = "ali.mohamed@example.com",
            age = 29,
            city = "alex"
        };

        var user3 = new
        {
            name = "hany ali",
            email = "hany.ali@example.com",
            age = 35,
            city = "alex"
        };

        var schema = new Schema()
        .AddTextField(new FieldName("$.name", "name"))
        .AddTagField(new FieldName("$.city", "city"))
        .AddNumericField(new FieldName("$.age", "age"));


        fT.Create(
            "idx:users",
            new FTCreateParams().On(IndexDataType.JSON).Prefix("user:")
            , schema
            );


        json.Set("user:1", "$", user1);
        json.Set("user:2", "$", user2);
        json.Set("user:3", "$", user3);


    }

    static void SearchbyName_age(IDatabase db, SearchCommands fT)
    {
        Console.WriteLine("searching by name & age");
        var result = fT.Search("idx:users", new Query("hany @age:[30 40]")).Documents.Select(x => x["jso
n"]);
        Console.WriteLine(string.Join("\n",result));
    }

    static void SearchbyName_city(IDatabase db, SearchCommands fT)
    {
        Console.WriteLine("searching by name & city");
        var result = fT.Search("idx:users", new Query("hany").ReturnFields(new FieldName("$.city", "city")))
            .Documents.Select(x => x["city"]);
        Console.WriteLine(string.Join("\n", result));
```

```
        }


        static void Searchbycity(IDatabase db, SearchCommands fT)
        {
            Console.WriteLine("searching by city ");
            var request = new AggregationRequest("*").GroupBy("@city", Reducers.Count().As("count"));

            var result=fT.Aggregate("idx:users",request);

            for (int i = 0; i < result.TotalResults; i++)
            {
                var row = result.GetRow(1);
                Console.WriteLine($"{row["city"]} - {row["count"]}");
            }
        }
    }
}
```

> output
>
> searching by name & age
> {"name":"hany
> ali","email":"hany.ali@example.com","age":35,"city":"alex"}
> searching by name & city
> alex
> searching by city
> cairo - 1
> cairo - 1

## 🔶 JSON Support in Redis

Redis can now work with **JSON objects** using a module called **RedisJSON**. Here's what that means:

### ✅ What you can do:

- **Store** full JSON objects in Redis.
- **Update** specific parts of those JSON objects.
- **Retrieve** entire objects or just specific fields.

### ✅ Follows JSON standard:

- Redis stores data in valid **JSON format** (e.g., `{"name": "Ali", "age": 30}`).

### ✅ Supports JSONPath:

- You can use **JSONPath**, a query language like XPath but for JSON, to get or update nested values.
  - Example: `$.user.name` gets the name inside a nested "user" object.

### ✅ Stored as binary data:

- Internally, Redis stores JSON as **binary**, which makes it faster and more efficient.

---

If you're coding, you'll use commands like:

- `JSON.SET` → to store a JSON object.
- `JSON.GET` → to retrieve it.
- `JSON.DEL` → to delete fields.
- `JSON.ARRAPPEND` → to add to arrays inside your JSON.

Some `json` Commands:

```
127.0.0.1:6379> json.set name $ '"ahmed"'
OK
127.0.0.1:6379> json.get name $
"[\"ahmed\"]"
127.0.0.1:6379> json.type name $
1) "string"
127.0.0.1:6379> json.strlen name $
1) (integer) 5
127.0.0.1:6379> json.strappend $ '" hany"'
(error) ERR could not perform this operation on a key that doesn't exist
127.0.0.1:6379> json.strappend name  $ '" hany"'
1) (integer) 10
127.0.0.1:6379> json.get name $
"[\"ahmed hany\"]"
127.0.0.1:6379> json.set n $ 0
OK
127.0.0.1:6379> json.numincrby $ 1
(error) ERR wrong number of arguments for 'json.numincrby' command
127.0.0.1:6379> json.numincrby n $ 1
"[1]"
127.0.0.1:6379> json.numincrby n $ 1.5
"[2.5]"
127.0.0.1:6379> json.numincrby n $ -0.75
"[1.75]"
127.0.0.1:6379> json.numincrby n $ 24
"[25.75]"
127.0.0.1:6379> json.set num $ 5
OK
127.0.0.1:6379> json.nummultby num  $ 6
"[30]"
127.0.0.1:6379>
```

## json arrays

```
# redis-cli
127.0.0.1:6379> json.set arr $[true,{"age":20},null]
Invalid argument(s)
127.0.0.1:6379> json.set arr $'[true,{"age":20},null]'
(error) ERR wrong number of arguments for 'json.set' command
127.0.0.1:6379> json.set arr $ '[true,{"age":20},null]'
OK
127.0.0.1:6379> json.get arr $
"[[true,{\"age\":20},null]]"
127.0.0.1:6379> json.get arr $[0]
"[true]"
127.0.0.1:6379> json.get arr $[2]
"[null]"
127.0.0.1:6379> json.get arr $[1].age
"[20]"
127.0.0.1:6379> json.del $arr[-1]
(integer) 0
127.0.0.1:6379> json.get arr $
"[[true,{\"age\":20},null]]"
127.0.0.1:6379> json.get arr $[-1]
"[null]"
127.0.0.1:6379> json.get arr $
"[[true,{\"age\":20},null]]"
127.0.0.1:6379> json.del arr $[-1]
(integer) 1
127.0.0.1:6379> json.get arr $
"[[true,{\"age\":20}]]"
127.0.0.1:6379> json.set vals $[]
(error) ERR wrong number of arguments for 'json.set' command
127.0.0.1:6379> json.set vals $ []
OK
127.0.0.1:6379> json.arrapend vals $ 0
(error) ERR unknown command 'json.arrapend', with args beginning with: 'vals' '$' '0'
127.0.0.1:6379> json.arrappend vals $ 0
1) (integer) 1
127.0.0.1:6379> json.get vals $
"[[0]]"
127.0.0.1:6379> json.arrappend $ -1 5 4
(error) ERR could not perform this operation on a key that doesn't exist
127.0.0.1:6379> json.arrappend vals  $ -1 5 4
1) (integer) 4
127.0.0.1:6379> json.get vals $
"[[0,-1,5,4]]"
127.0.0.1:6379> json.arrpop vals $
1) "4"
127.0.0.1:6379> json.get vals $
"[[0,-1,5]]"
127.0.0.1:6379>
```

### Commands Json Full:

```
127.0.0.1:6379> json.set obj $ '{"name":"ahmed","age":20,"work":"backend"}'
OK
127.0.0.1:6379> json.get obj $
"[{\"name\":\"ahmed\",\"age\":20,\"work\":\"backend\"}]"
127.0.0.1:6379> json.objlen obj $
1) (integer) 3
127.0.0.1:6379> json objkeys obj $
(error) ERR unknown command 'json', with args beginning with: 'objkeys' 'obj' '$'
127.0.0.1:6379> json.objkeys obj $
1) 1) "name"
   2) "age"
   3) "work"
127.0.0.1:6379>
```

## ⚠️ JSON Limitation

### ❗ Nesting Level Limit: Up to 128

- This means you can only nest JSON objects **up to 128 levels deep**.
- Example:
  - ✅ Valid: `{"a":{"b":{"c":...}}}` (up to 128 nested keys)
  - ❌ Invalid: Anything deeper than 128 levels will cause an error.

> This is a built-in safety feature to avoid performance issues or stack overflows.

---

### 🔷 Path in JSON

- Used to **access specific elements** in a JSON document.

### ✅ Syntax

- **JSONPath syntax**: uses `$` as the root element.
- **Legacy syntax**: older format, not shown in detail here. Ex:

```
127.0.0.1:6379> json.set tes $ '{"name":{"n":25}}'
OK
127.0.0.1:6379> json.get tes $.name.n
"[25]"
```

- **Return type**: always a JSON string (even if it's one object, it's wrapped in an array).

### 💡 Example from the Slide:

```
127.0.0.1:6379> JSON.SET v $ '{"name":"ahmed","age":20}'
OK

127.0.0.1:6379> JSON.GET v $
"[{\"name\":\"ahmed\",\"age\":20}]"
```

- `JSON.SET v $ '{...}'`: sets the key `v` with a JSON object.
- `JSON.GET v $`: retrieves the whole object at the root (`$`) → returns a **stringified JSON array**.

### 📌 Note:

Even if the value is just a single object, Redis returns it in an **array format** when using JSONPath (`$`). That's why you see:

```
[{"name":"ahmed","age":20}]
```

### 🧩 Sample JSON

```
{
  "users": [
    { "name": "Alice", "age": 25, "role": "admin" },
    { "name": "Bob", "age": 32, "role": "user" },
    { "name": "Charlie", "age": 29, "role": "moderator" }
  ],
  "meta": {
    "count": 3
  }
}
```

### ✅ JSONPath Syntax with Examples

| Syntax | Description | Example | Result |
|---|---|---|---|
| `$` | Root | `$` | Entire JSON document |
| `.` or `[]` | Child element | `$.meta.count` or `$['meta']['count']` | `3` |
| `*` | All elements | `$.users[*].name` | `["Alice", "Bob", "Charlie"]` |
| `[,]` | Multiple elements | `$['users', 'meta']` | Contents of `users` and `meta` |
| `[start:end:step]` | Array slice | `$.users[0:2]` | First 2 user objects |
| `?()` | Filter with condition | `$.users[?(@.age > 30)].name` | `["Bob"]` |
| `@` | Current element in filter | `$.users[?(@.role == 'admin')].name` | `["Alice"]` |
| `()` | Script expression (used inside `?()` filter) | `$.users[?((@.age >= 29 && @.role != 'user'))].name` | `["Charlie"]` |

### 💡 Bonus: Combining JSONPath

```
$.users[?(@.age > 25)].role
```

🔍 This will return:

```
["user", "moderator"]
```

## ✅ RedisJSON Use Cases (When to Use RedisJSON):

1. **Access and retrieval of subvalues**

   ➤ You can access **specific parts** (subvalues) of a JSON document directly — without reading the entire object.

   📌 Example: Get just `user.name` instead of fetching the whole user object.

2. **Atomic partial updates**

   ➤ You can **update only part** of a JSON object in a safe, atomic way — meaning no conflict or corruption happens.

   Includes actions like:

   - **Append** (add to a list)
   - **Remove** (delete a field or item)
   - **Increment** (increase a number field)

3. **Indexing and querying**

   ➤ You can create indexes on fields inside your JSON data and **search/query efficiently**.

   📌 Example: Find all users where `user.age > 25`.

---

## 🚀 In short:

**RedisJSON** is useful when you're storing structured JSON data in Redis and you want:

- Fast access to specific fields,
- Safe updates to parts of the object,
- Ability to search inside the JSON.

## 🔑 Redis Keys – What You Need to Know Quickly

- **Unique**: Each key must be different.
- **Binary Safe**: Keys can include any type of data, not just text.
- **Size Limit**: A key can be up to **512 MB** in size.
- **Flat Key Space**: All keys exist in one namespace, like one big list (no folders).
- **Logical Databases**: Redis supports multiple databases, but keys inside each one must still be unique.
- **Case Sensitive**: `Users:1000` ≠ `users:1000` — capital letters matter!

---

### 💡 Examples of Keys

- `Users:1001`
- `Users:1000` (capital U)
- `users:1000` (lowercase u) → This is **different** from the one above!

---

### 🧰 Useful Redis CLI Commands

### 🔍 `KEYS` vs `SCAN`

Both are used to **search keys**, but they are used **differently** and for **different scenarios**.

| Command | What it does | Use When | Pros / Cons |
|---|---|---|---|
| `KEYS pattern` | Finds **all** keys matching a pattern | ⚠️ Use in **development/testing only** | ❌ **Blocks Redis** if there are too many keys |
| `SCAN` | Iterates keys in chunks (non-blocking) | ✅ Use in **production** | ✅ Safe, non-blocking, scalable |

---

### 🧪 Examples:

`KEYS users:1*`

- Finds all keys that start with `users:1`
- ❌ Not good in production (it scans everything at once!)

`SCAN 0 MATCH users:1* COUNT 1000`

- Starts scanning from cursor `0`
- Uses a pattern match ( `MATCH users:1*` )
- `COUNT 1000` tells Redis to return **up to** 1000 keys at a time (not guaranteed)

➡️ Then you repeat the `SCAN` with the cursor it returns until the cursor = 0 again (done).

---

### 🗑️ `DEL` vs `UNLINK`

Both delete keys, but behave differently:

| Command | What it does | Use When | Pros / Cons |
|---|---|---|---|
| `DEL key` | Deletes key **immediately** (blocking) | OK for small data | ❌ May block Redis if key is large |
| `UNLINK key` | Deletes key **asynchronously** | ✅ Best for large keys | ✅ Non-blocking, works in background |

---

### 🧪 Examples:

- `DEL users:1000`
  → Deletes immediately. Use when the key is small.
- `UNLINK users:1001`
  → Deletes in background (good for large values, like lists or big strings)

---

## ❓ `EXISTS key`

- Checks if a key exists in the database.

### 🧪 Example:

- `EXISTS users:1000`
  → Returns
  `1` if key exists, `0` if not.

✅ Use it to **verify before deleting**, or to check if a value is cached.

---

## ✅ Best Practice Summary

| Task | Best Command |
|---|---|
| Search keys in development | `KEYS pattern` |
| Search keys in production | `SCAN` with `MATCH` |
| Delete small keys | `DEL` |
| Delete large keys | `UNLINK` |
| Check if key exists | `EXISTS key` |

---

## 🔀 Difference Between `NX` and `XX` in `SET`

| Option | Meaning | Use When... |
|---|---|---|
| `NX` | Set the key **only if it does NOT exist** | ➤ You want to **create** a key, but not overwrite |
| `XX` | Set the key **only if it already exists** | ➤ You want to **update** a key, but not create new ones |

---

## ✅ Examples:

### 1. `SET mykey "Hello" NX`

- Will **create** `mykey` if it **doesn't exist**
- If `mykey` already exists → ❌ nothing happens

🧠 Useful for: **caching**, or **locking mechanisms**

---

### 2. `SET mykey "Updated" XX`

- Will **update** `mykey` **only if it exists**
- If `mykey` doesn't exist → ❌ nothing happens

🧠 Useful for: **ensuring updates** happen only on existing data

---

## ⚠️ Important Notes:

- You can't use `NX` and `XX` **together** in one command.
- They're **mutually exclusive** — only one of them makes sense at a time.

---

## 🔁 Summary:

| Flag | Key Must Exist? | Action | Example Use Case |
|---|---|---|---|
| NX | ❌ No | Create only | Add a key if it's new |
| XX | ✅ Yes | Update only | Update value if it exists |

Commands :

```
# redis-cli
127.0.0.1:6379> set users:1001 ahmed
OK
127.0.0.1:6379> set users:1002 ali
OK
127.0.0.1:6379> keys users
(empty array)
127.0.0.1:6379> keys users:
(empty array)
127.0.0.1:6379> keys users:1*
1) "users:1001"
2) "users:1002"
127.0.0.1:6379> scan 0 match users:1*
1) "0"
2) 1) "users:1002"
   2) "users:1001"
127.0.0.1:6379> del users:1001
(integer) 1
127.0.0.1:6379> get users:1001
(nil)
127.0.0.1:6379> unlink users:1002
(integer) 1
127.0.0.1:6379> get users:1002
(nil)
127.0.0.1:6379> exists users:1001
(integer) 0
127.0.0.1:6379> exists users:1002
(integer) 0
127.0.0.1:6379> set users:1003 ahmed nx
OK
127.0.0.1:6379> set users:1003 omar nx
(nil)
127.0.0.1:6379> exists users:1003
(integer) 1
127.0.0.1:6379> set users:1001 ali xx
(nil)
127.0.0.1:6379> set users:1003 ali xx
OK
127.0.0.1:6379>
```

## Key Expiration Overview

- **TTL (Time to Live):** This refers to the amount of time a key will remain in the database before it is automatically deleted.

### Commands for Managing Key Expiration

1. **SET [key] [value] EX [seconds]:**
   - Sets a key with a value and specifies its expiration time in seconds.
   - Example: `SET mykey "hello" EX 10` sets the key `mykey` with the value `"hello"` and expires it after 10 seconds.
2. **SET [key] [value] PX [milliseconds]:**
   - Similar to the `EX` option but sets the expiration time in milliseconds.
   - Example: `SET mykey "hello" PX 5000` sets the key `mykey` with the value `"hello"` and expires it after 5000 milliseconds (5 seconds).
3. **EXPIRE [key] [seconds]:**
   - Sets the expiration time for an existing key in seconds.
   - Example: `EXPIRE mykey 20` sets the key `mykey` to expire after 20 seconds.
4. **EXPIREAT [key] [timestamp-seconds]:**
   - Sets the expiration time for a key based on a Unix timestamp (in seconds).
   - Example: `EXPIREAT mykey 1633072800` sets the key `mykey` to expire at the specified Unix timestamp.
5. **PEXPIRE [key] [milliseconds]:**
   - Sets the expiration time for a key in milliseconds.
   - Example: `PEXPIRE mykey 10000` sets the key `mykey` to expire after 10000 milliseconds (10 seconds).
6. **PEXPIREAT [key] [timestamp-milliseconds]:**
   - Sets the expiration time for a key based on a Unix timestamp (in milliseconds).
   - Example: `PEXPIREAT mykey 1633072800000` sets the key `mykey` to expire at the specified Unix timestamp in milliseconds.
7. **TTL [key]:**
   - Returns the remaining time to live of a key in seconds.
   - Example: `TTL mykey` returns the number of seconds left until `mykey` expires.
8. **PTTL [key]:**
   - Returns the remaining time to live of a key in milliseconds.
   - Example: `PTTL mykey` returns the number of milliseconds left until `mykey` expires.
9. **PERSIST [key]:**
   - Removes the expiration from a key, making it persistent.
   - Example: `PERSIST mykey` removes the expiration from `mykey`, so it won't be automatically deleted.

Commands:

```
# redis-cli
127.0.0.1:6379> set user:1 ahmed EX 120
OK
127.0.0.1:6379> TTL user:1
(integer) 106
```

```
127.0.0.1:6379> get user:1
"ahmed"
127.0.0.1:6379> pttl user:1
(integer) 75359
127.0.0.1:6379> set user:2 ali px 120000
OK
127.0.0.1:6379> TTL user:2
(integer) 92
127.0.0.1:6379> expire user:2 300000
(integer) 1
127.0.0.1:6379> expire user:2 300
(integer) 1
127.0.0.1:6379> pexpire user:2 10000000
(integer) 1
127.0.0.1:6379> persist user:2
(integer) 1
127.0.0.1:6379> ttl user:2
(integer) -1 ：not have expitration time it will exists every time
127.0.0.1:6379> ttl user:1
(integer) -2 ：it removed because time ended
127.0.0.1:6379> get user:1
(nil)
127.0.0.1:6379> get user:2
"ali"
```

## Strings Overview

- **Store strings, numerical values, serialized JSON, and binary:**
  - The Strings data type can hold various types of data, including plain text, numbers, JSON objects (in string format), and binary data.
- **Used for caching:**
  - Strings are commonly used for caching purposes, where frequently accessed data is stored temporarily to improve performance.
- **Manipulate numerical values:**
  - You can perform arithmetic operations on numerical values stored as strings, such as incrementing or decrementing them.
- **Maximum of 512 MB:**
  - The maximum size of a String value is 512 MB, which is quite large and allows for storing substantial amounts of data.

### Commands for Manipulating Numerical Values

1. **INCRBYFLOAT command for float values:**
   - This command increments the numeric value of a string key by a floating-point number.
   - Example: `INCRBYFLOAT mykey 0.5` will increase the value of `mykey` by 0.5 if it holds a numeric value.
2. **INCRBY command for only integer values:**
   - This command increments the numeric value of a string key by an integer.
   - Example: `INCRBY mykey 5` will increase the value of `mykey` by 5 if it holds an integer value.

Commands:

```
127.0.0.1:6379> set count 5
OK
127.0.0.1:6379> incrby count 2
(integer) 7
127.0.0.1:6379> incrby count 2.4
(error) ERR value is not an integer or out of range
127.0.0.1:6379> incrbyfloat count 2.4
"9.4"
127.0.0.1:6379>
```

## Hashes Overview

- **Hashes** are used to store multiple string fields and their associated values in a single key-value pair. They are useful for representing objects with multiple attributes.

### Commands for Managing Hashes

1. **HSET [key] [field] [value]:**
   - Sets the value of a field in a hash.
   - Example: `HSET user:1 name "John"` sets the `name` field of the `user:1` hash to `"John"`.
2. **HDEL [key] [field1] [field2] ...:**
   - Deletes one or more fields from a hash.
   - Example: `HDEL user:1 name age` deletes the `name` and `age` fields from the `user:1` hash.
3. **HINCRBY [key] [field] [increment]:**
   - Increments the numeric value of a field in a hash by a given amount.
   - Example: `HINCRBY user:1 age 1` increments the `age` field of the `user:1` hash by 1.
4. **HEXISTS [key] [field]:**
   - Checks if a field exists in a hash.
   - Example: `HEXISTS user:1 name` returns 1 if the `name` field exists in the `user:1` hash, otherwise returns 0.
5. **HINCRBYFLOAT [key] [field] [increment]:**
   - Increments the numeric value of a field in a hash by a floating-point number.
   - Example: `HINCRBYFLOAT user:1 score 0.5` increments the `score` field of the `user:1` hash by 0.5.
6. **HKEYS [key]:**
   - Returns all field names in a hash.
   - Example: `HKEYS user:1` returns all field names in the `user:1` hash.
7. **HVALS [key]:**
   - Returns all field values in a hash.
   - Example: `HVALS user:1` returns all field values in the `user:1` hash.

### Additional Information

- **Performance (O(1) - O(n)):**
  - Most hash operations have constant time complexity (O(1)), meaning they are very fast regardless of the size of the hash. However, some operations like iterating over all fields may have linear time complexity (O(n)).
- **Fields can be only strings:**
  - The field names in a hash must be strings.
- **Deleting the hash key will delete all hash fields:**
  - If you delete the entire hash key, all its fields and their values will be removed.

```
# redis-cli
127.0.0.1:6379> hset k1 name 'ahmed' age 20
(integer) 2
127.0.0.1:6379> hset k1 status active
```

```
(integer) 1
127.0.0.1:6379> hgetall k1
1) "name"
2) "ahmed"
3) "age"
4) "20"
5) "status"
6) "active"
127.0.0.1:6379> hdel k1 status
(integer) 1
127.0.0.1:6379> hgetall k1
1) "name"
2) "ahmed"
3) "age"
4) "20"
127.0.0.1:6379> hincrby k1 age 5
(integer) 25
127.0.0.1:6379> hget k1 name
"ahmed"
127.0.0.1:6379> hexists k1 name
(integer) 1
127.0.0.1:6379> hincrbyfloat k1 age 3.5
"28.5"
127.0.0.1:6379> hkeys k1
1) "name"
2) "age"
127.0.0.1:6379> hvals k1
1) "ahmed"
2) "28.5"
127.0.0.1:6379>
```

## Lists Overview

- **Stack and Queue:** Lists can be used to implement both stack (LIFO - Last In First Out) and queue (FIFO - First In First Out) data structures.

### List as Stack

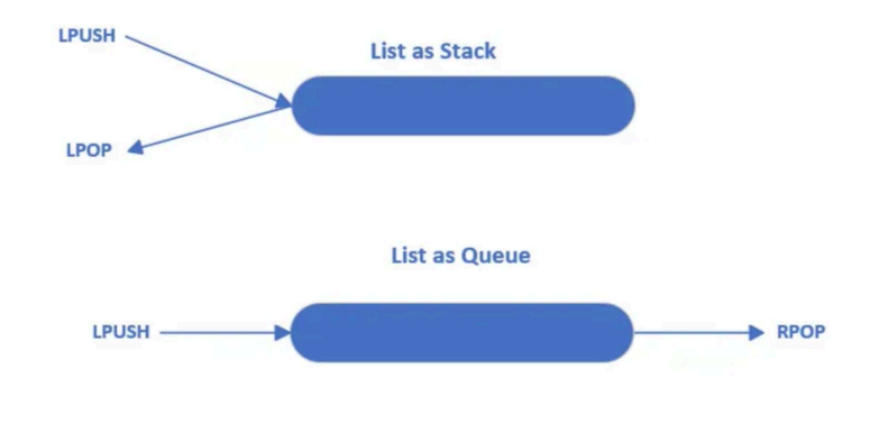
1. **LPUSH [key] [value]:**
   - Adds an element to the head (left side) of the list.
   - Example: `LPUSH mylist "item1"` adds `"item1"` to the beginning of the list `mylist`.
2. **LPOP [key]:**
   - Removes and returns the element from the head (left side) of the list.
   - Example: `LPOP mylist` removes and returns the first item from the list `mylist`.

### List as Queue

1. **LPUSH [key] [value]:**
   - Adds an element to the head (left side) of the list.
   - Example: `LPUSH myqueue "item1"` adds `"item1"` to the beginning of the list `myqueue`.
2. **RPOP [key]:**
   - Removes and returns the element from the tail (right side) of the list.
   - Example: `RPOP myqueue` removes and returns the last item from the list `myqueue`.

### Visual Representation

- **List as Stack:** Elements are added and removed from the same end (head/left side) of the list.
- **List as Queue:** Elements are added at the head (left side) of the list and removed from the tail (right side) of the list.



## Ordered Sets Overview

- **Ordered Sets** are data structures that store unique elements along with a score for each element. The elements are automatically sorted by their scores.

### Commands for Managing Ordered Sets

1. **ZREMRANGEBYRANK [key] [start] [stop]:**
   - Removes all elements in the ordered set within the given rank range.
   - Example: `ZREMRANGEBYRANK myset 0 2` removes the first three elements from the ordered set `myset`.
2. **ZREMRANGEBYSCORE [key] [min] [max]:**
   - Removes all elements in the ordered set with scores within the given range.
   - Example: `ZREMRANGEBYSCORE myset 0 5` removes all elements with scores between 0 and 5 from the ordered set `myset`.
3. **ZSCORE [key] [member]:**
   - Returns the score of a specified member in the ordered set.
   - Example: `ZSCORE myset "item"` returns the score of the member `"item"` in the ordered set `myset`.
4. **ZINTERSTORE [destination] [numkeys] [key1] [key2] ...:**
   - Stores the intersection of multiple ordered sets into a new sorted set.
   - Example: `ZINTERSTORE newset 2 set1 set2` stores the intersection of `set1` and `set2` into a new sorted set `newset`.
5. **ZUNIONSTORE [destination] [numkeys] [key1] [key2] ...:**
   - Stores the union of multiple ordered sets into a new sorted set.
   - Example: `ZUNIONSTORE newset 2 set1 set2` stores the union of `set1` and `set2` into a new sorted set `newset`.

### Visual Representation

- **S1, S2, S3:** Represent different ordered sets.
- **A, B, C:** Represent elements within these sets.
- The intersection (A) represents common elements between S1 and S2.
- The union would include all unique elements from S1 and S2 (A, B, C).

Commands :

```
127.0.0.1:6379> zadd myoset 10 a 20 b 30 c 40 d
(integer) 4
127.0.0.1:6379> zremrangebyrank myoset 0 1
(integer) 2
127.0.0.1:6379> zrange myoset 0 -1
1) "c"
2) "d"
127.0.0.1:6379> zremrangebyscore myoset 30 40
(integer) 2
127.0.0.1:6379> zrange myoset 0 -1
(empty array)
127.0.0.1:6379> zadd oset 1 x 2 y 3 z
(integer) 3
127.0.0.1:6379> zscore oset y
"2"
```

```
127.0.0.1:6379> zadd oset1 1 a 2 b 3 c
(integer) 3
127.0.0.1:6379> zadd oset2 1 a 5 c 3 w
(integer) 3
127.0.0.1:6379> zinterstore oset3 2 oset1 oset2 aggregate sum
(integer) 2
127.0.0.1:6379> zrange oset3 0 -1
1) "a"
2) "c"
127.0.0.1:6379> zrange oset3 0 -1 withscores
1) "a"
2) "2"
3) "c"
4) "8"
127.0.0.1:6379> zrange oset1 0 -1 withscores
1) "a"
2) "1"
3) "b"
4) "2"
5) "c"
6) "3"
127.0.0.1:6379> zrange oset2 0 -1 withscores
1) "a"
2) "1"
3) "w"
4) "3"
5) "c"
6) "5"
127.0.0.1:6379> zunionstore s4 3 oset1 oset2 oset3 aggregate max
(integer) 4
127.0.0.1:6379> zrange s4 0 -1 withscores
1) "a"
2) "2"
3) "b"
4) "2"
5) "w"
6) "3"
7) "c"
8) "8"
127.0.0.1:6379>
```

## Transactions Overview

- **Transactions** allow you to group multiple commands into a single atomic operation, ensuring that either all commands are executed successfully or none at all.

### Commands for Managing Transactions

1. **MULTI:**
   - Marks the start of a transaction block. All commands after `MULTI` are queued and not executed immediately.
   - Example: `MULTI`
2. **EXEC:**
   - Executes all the commands in the transaction block atomically. If any command fails, the entire transaction fails.
   - Example: `EXEC`
3. **DISCARD:**
   - Cancels the transaction and discards all commands in the transaction block.
   - Example: `DISCARD`

### Key Characteristics

- **No Rollback:**
  - Once a transaction is executed using `EXEC`, there is no automatic rollback mechanism. If a command within the transaction fails, the entire transaction fails, and no changes are made.
- **Sequential:**
  - Commands within a transaction are executed sequentially, one after another, in the order they were queued.

### Sample Scenarios

### Sample 1: To Execute Commands

```
MULTI
Command 1
Command 2
Command 2
EXEC
```

- This sample shows how to queue multiple commands and execute them as a single transaction using `EXEC`.

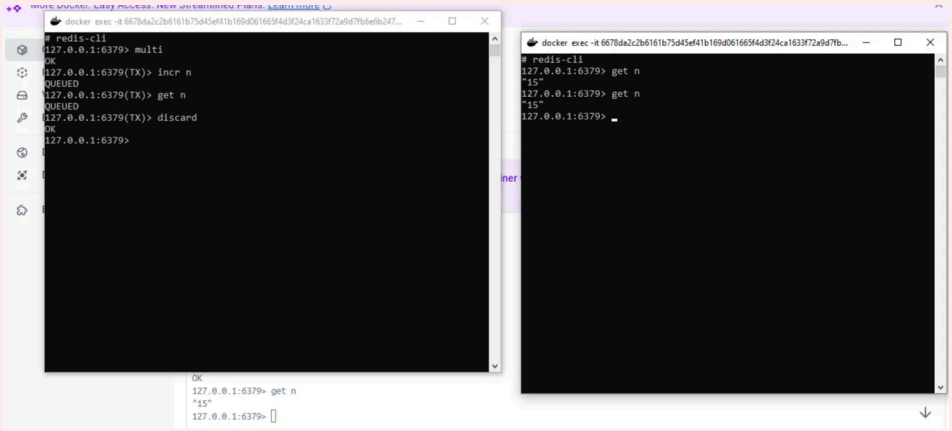
### Sample 2: To Remove All Commands

```
MULTI
Command 1
Command 2
Command 2
DISCARD
```

- This sample shows how to cancel the transaction and discard all queued commands using `DISCARD`.

Commands:

```
# redis-cli
127.0.0.1:6379> multi
OK
127.0.0.1:6379(TX)> set n 10
QUEUED
127.0.0.1:6379(TX)> incrby n 5
QUEUED
127.0.0.1:6379(TX)> get n
QUEUED
127.0.0.1:6379(TX)> exec
1) OK
2) (integer) 15
3) "15"
127.0.0.1:6379> multi
OK
127.0.0.1:6379(TX)> exex
(error) ERR unknown command 'exex', with args beginning with:
127.0.0.1:6379(TX)> discard
OK
127.0.0.1:6379> get n
"15"
127.0.0.1:6379>
```



## Optimistic Locking Overview

- **Optimistic Locking** is a strategy used to manage concurrent updates to data. It assumes that conflicts between transactions are rare and allows transactions to proceed without locking resources. If a conflict is detected during the commit phase, the transaction is aborted.

## Commands for Managing Optimistic Locking

1. **WATCH [key1] [key2] [key3]:**
   - Monitors one or more keys for changes. If any of the watched keys are modified by another transaction before the current transaction is executed, the transaction will be aborted.
   - Example: `WATCH key1 key2 key3` watches the keys `key1`, `key2`, and `key3` for changes.
2. **UNWATCH:**
   - Cancels all the keys being watched by the current transaction. This command is useful if you want to stop monitoring the keys before executing the transaction.
   - Example: `UNWATCH` cancels all watched keys.

## How It Works

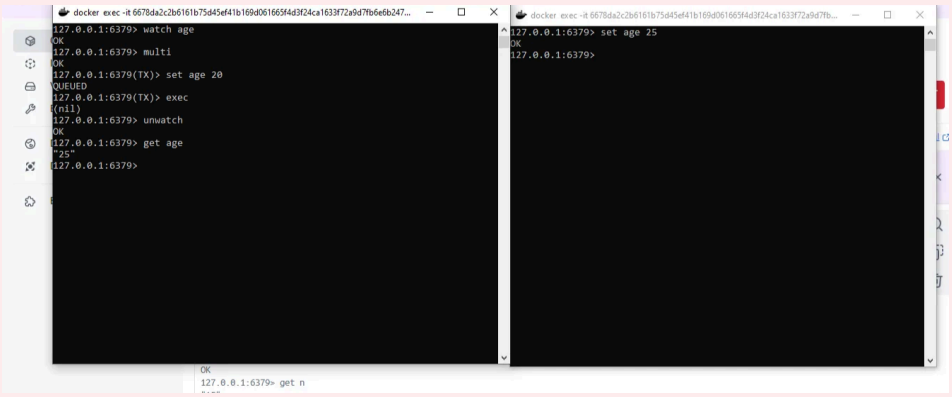
- When you start a transaction using `MULTI` and have previously watched some keys with `WATCH`, the transaction will check if any of the watched keys have been modified by other transactions since they were watched.
- If any watched key has been modified, the transaction will fail when you try to execute it with `EXEC`.
- If no watched keys have been modified, the transaction will proceed as normal.

## Example Scenario

```
WATCH key1 key2 key3
MULTI
Command 1
Command 2
EXEC
```

- In this example, the transaction will only execute successfully if none of the keys `key1`, `key2`, or `key3` have been modified by other transactions since they were watched.

Commands show how can block keys to apply transaction:



### Flattening Relationship in a Single Hash

- **Student + Courses:** The example demonstrates how to flatten a relationship between a student and their courses into a single hash.

### Original JSON Object

```
{
  "Name": "Ahmed",
  "Age": 25,
  "Courses": [
    {
      "CourseName": "Redis",
      "CourseDuration": 4
    },
    {
      "CourseName": "RabbitMQ",
      "CourseDuration": 3
    }
  ]
}
```

### Flattened Hash Representation

```
{
  "Name": "Ahmed",
  "Age": 25,
  "Courses:1:CourseName": "Redis",
  "Courses:1:CourseDuration": 4,
  "Courses:2:CourseName": "RabbitMQ",
  "Courses:2:CourseDuration": 3
}
```

### Explanation

- **Flattening:** The original JSON object has a nested structure where the `Courses` field contains an array of course objects. To flatten this structure into a single hash, each course object is represented as separate fields with unique keys.
- **Unique Keys:** Each course-related field is prefixed with `Courses:<index>:`, where `<index>` is the position of the course in the original array (starting from 1). For example:
  - `Courses:1:CourseName` represents the `CourseName` of the first course.
  - `Courses:1:CourseDuration` represents the `CourseDuration` of the first course.
  - `Courses:2:CourseName` represents the `CourseName` of the second course.
  - `Courses:2:CourseDuration` represents the `CourseDuration` of the second course.

### Benefits of Flattening

- **Simplified Data Structure:** By flattening the relationship, you can store and retrieve data more easily using a single hash.
- **Efficient Operations:** You can perform operations like adding, updating, or deleting course information directly on the hash without needing to manage nested structures.

## Hashes - Complex Objects

### Flattening Relationship in a Single Hash

- This involves representing complex relationships (such as nested objects) within a single hash by flattening them into a simpler structure.

### Advantages (Good)

1. **Encapsulation:**
   - Encapsulation allows you to group related data and operations together, making it easier to manage and understand.
   - By flattening relationships into a single hash, you can encapsulate all relevant information about an object in one place, which can simplify data access and manipulation.
2. **No Transaction:**
   - When working with a single hash, you don't need to use transactions to ensure data consistency across multiple keys.
   - This can improve performance and reduce complexity, as you don't have to worry about coordinating changes across different parts of your data.

### Disadvantages (Bad)

1. **Relationship Maintenance:**
   - Flattening relationships can make it more difficult to maintain and update complex relationships.
   - If the structure of your data changes, you may need to update multiple fields in the hash, which can be error-prone and time-consuming.
2. **Large Objects:**
   - Storing complex objects in a single hash can result in large hash structures.
   - Large hashes can consume more memory and may impact performance, especially when dealing with very large or deeply nested data structures.

### Summary

Using hashes to flatten complex relationships can provide benefits such as better encapsulation and reduced transaction overhead. However, it also comes with challenges like increased difficulty in maintaining relationships and potential issues with large object sizes. Understanding these trade-offs is important when deciding whether to use this approach in your applications.

## Pub/Sub Overview

- **Pub/Sub** allows for a messaging pattern where publishers can send messages to channels, and subscribers can listen to those channels to receive messages. This decouples the sender and receiver, making it useful for various communication scenarios.

### Commands for Managing Pub/Sub

### Pubsub Subcommand <arguments>

1. *PUBSUB CHANNELS :*
   - Lists all active channels that have at least one subscriber.
   - Example: `PUBSUB CHANNELS *` returns a list of all active channels.
2. **PUBSUB NUMSUB ch-1 ch-2:**
   - Returns the number of subscribers for each specified channel.
   - Example: `PUBSUB NUMSUB ch-1 ch-2` returns the number of subscribers for `ch-1` and `ch-2`.
3. **PUBSUB NUMPAT:**
   - Returns the number of subscriptions to patterns.
   - Example: `PUBSUB NUMPAT` returns the total number of pattern subscriptions.

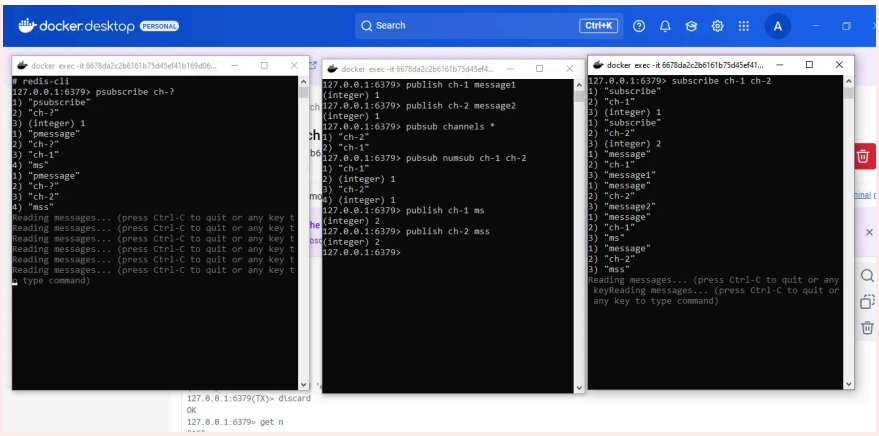
### Psubscribe <pattern>

- The `PSUBSCRIBE` command allows you to subscribe to channels based on patterns. Here are some examples:

1. **PSUBSCRIBE ch-?:**
   - Matches any channel that has a single character after `ch-`.
   - Example: Matches `ch-1`, `ch-2`, `ch-a`, `ch-b`.
2. **PSUBSCRIBE ch-*:**
   - Matches any channel that starts with `ch-` followed by any sequence of characters.
   - Example: Matches `ch-1`, `ch-123`, `ch-a`.
3. **PSUBSCRIBE ch-[ae]:**
   - Matches any channel that has either 'a' or 'e' after `ch-`.
   - Example: Matches only `ch-a`, `ch-e`.
4. **PSUBSCRIBE ch-[^e]:**
   - Matches any channel that has any character except 'e' after `ch-`.
   - Example: Matches `ch-a`, `ch-1`, but not `ch-e`.

### Summary

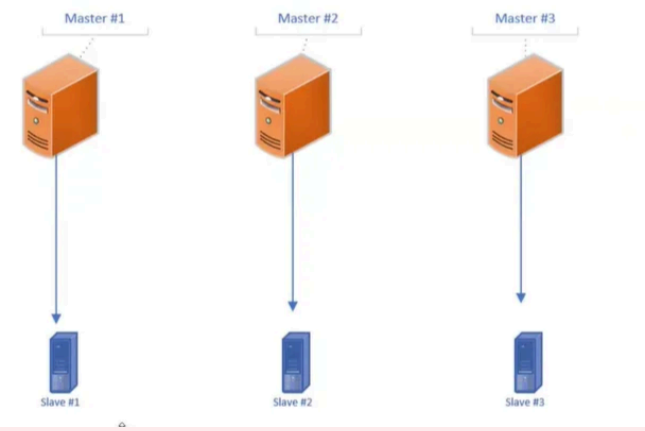
- **Pub/Sub** provides a flexible way to communicate between different parts of an application or between different applications.
- The `PUBSUB` subcommands help manage and monitor channels and subscriptions.
- The `PSUBSCRIBE` command allows for pattern-based subscriptions, making it easier to handle multiple channels with similar naming conventions.


docker desktop
PERSONAL
Search
Ctrl+K
# redis-cli
127.0.0.1:6379> psubscribe ch-?
1) "psubscribe"
2) "ch-?"
3) (integer) 1
1) "pmessage"
2) "ch-?"
3) "ch-1"
4) "ms"
1) "pmessage"
2) "ch-?"
3) "ch-2"
4) "mss"
Reading messages... (press Ctrl-C to quit or any key t
type command)
127.0.0.1:6379> publish ch-1 message1
(integer) 1
127.0.0.1:6379> publish ch-2 message2
(integer) 1
127.0.0.1:6379> pubsub channels *
1) "ch-2"
2) "ch-1"
127.0.0.1:6379> pubsub numsub ch-1 ch-2
1) "ch-1"
2) (integer) 1
3) "ch-2"
4) (integer) 1
127.0.0.1:6379> publish ch-1 ms
(integer) 2
127.0.0.1:6379> publish ch-2 mss
(integer) 2
127.0.0.1:6379>
127.0.0.1:6379> subscribe ch-1 ch-2
1) "subscribe"
2) "ch-1"
3) (integer) 1
1) "subscribe"
2) "ch-2"
3) (integer) 2
1) "message"
2) "ch-1"
3) "message1"
1) "message"
2) "ch-2"
3) "message2"
1) "message"
2) "ch-1"
3) "ms"
1) "message"
2) "ch-2"
3) "mss"
Reading messages... (press Ctrl-C to quit or any
keyReading messages... (press Ctrl-C to quit or
any key to type command)
127.0.0.1:6379(TX)> discard
OK
127.0.0.1:6379> get n

## Clustering

## Clustering Overview

- **Redis scales horizontally with a deployment topology called Redis Cluster:**
  - Redis Cluster allows Redis to scale horizontally by distributing data across multiple nodes (servers). This means that as your data grows, you can add more servers to handle the load.




### Shard: Master with its Slaves

- **Shard:** A shard is a unit of data storage in a Redis Cluster. Each shard consists of a master node and one or more slave nodes.
  - **Master:** The master node is responsible for handling read and write operations for the data it manages.
  - **Slaves:** Slave nodes are replicas of the master node. They hold copies of the data and can take over if the master node fails (failover).

### Redis Cluster TCP Ports

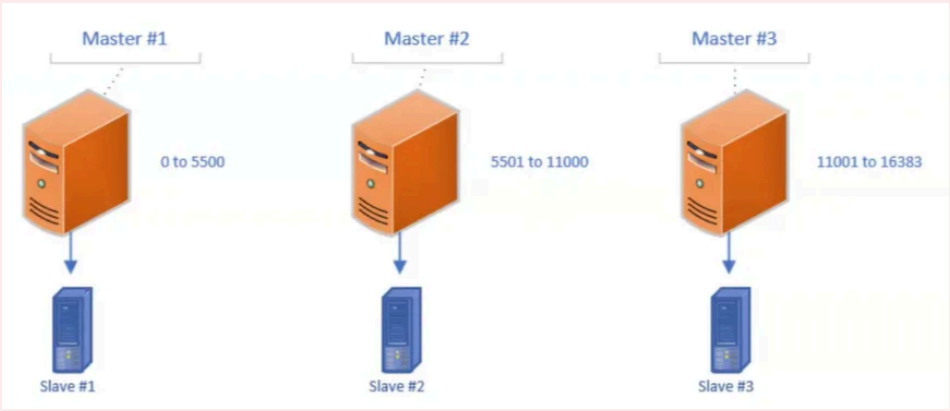
- **Client port:** This is the port used by clients to connect to the Redis Cluster and perform operations.
- **Cluster bus port = Client port + 10000:** The cluster bus port is used for communication between nodes in the cluster. It is calculated by adding 10000 to the client port.

### Visual Representation

- The diagram shows three master nodes (`Master #1`, `Master #2`, `Master #3`) each connected to their respective slave nodes (`Slave #1`, `Slave #2`, `Slave #3`).
- This setup ensures high availability and fault tolerance. If a master node fails, one of its slave nodes can automatically take over its role.

### Summary

- **Horizontal Scaling:** Redis Cluster allows you to distribute data across multiple nodes to handle larger datasets and higher loads.
- **Shards:** Each shard consists of a master node and one or more slave nodes, ensuring data redundancy and failover capabilities.
- **TCP Ports:** The client port is used for client connections, while the cluster bus port (client port + 10000) is used for inter-node communication.




## Clustering Overview

- **Hash slots = 16384:**
  - Redis Cluster uses 16,384 hash slots to distribute keys across multiple nodes. Each node in the cluster is responsible for a subset of these hash slots.
- **CRC16 algorithm, to calculate hash slot:**
  - The CRC16 (Cyclic Redundancy Check 16) algorithm is used to determine which hash slot a given key belongs to. This ensures consistent distribution of keys across the cluster.
- **No downtime while moving slots:**
  - Redis Cluster allows for the migration of hash slots between nodes without causing downtime. This feature enables dynamic scaling and rebalancing of the cluster as needed.

## Multi-key Operations

- **Key 1: user:{123}:profile**
- **Key 2: user:{123}:account**

stored two keys in same hash slote

- When performing multi-key operations, such as transactions or commands that involve multiple keys, it's important that all related keys are stored on the same node. In this example, both `user:{123}:profile` and `user:{123}:account` should be hashed to the same node to ensure they can be accessed together efficiently.

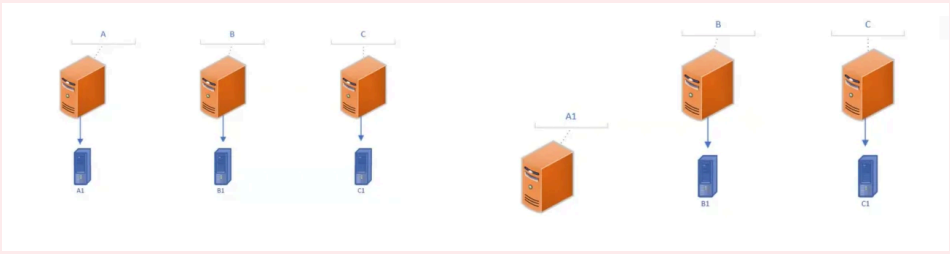
## Visual Representation

- The diagram shows three master nodes (`Master #1`, `Master #2`, `Master #3`) each responsible for a range of hash slots:
  - **Master #1:** Hash slots 0 to 5500
  - **Master #2:** Hash slots 5501 to 11000
  - **Master #3:** Hash slots 11001 to 16383
- Each master node has a corresponding slave node (`Slave #1`, `Slave #2`, `Slave #3`) that acts as a replica. If a master node fails, its slave can take over seamlessly, ensuring high availability and fault tolerance.

## Summary

- **Hash Slots:** Redis Cluster uses 16,384 hash slots to distribute keys across nodes.
- **CRC16 Algorithm:** This algorithm calculates the hash slot for a given key.

- **No Downtime:** Moving hash slots between nodes can be done without downtime.
- **Multi-key Operations:** Related keys should be stored on the same node for efficient access.




## Clustering Overview

- **Master-replica model:** This model involves having one or more master nodes and one or more replica (or slave) nodes for each master. The master nodes handle read and write operations, while the replica nodes hold copies of the data for redundancy and failover purposes.

## Visual Representation

The diagram shows two sets of clusters:

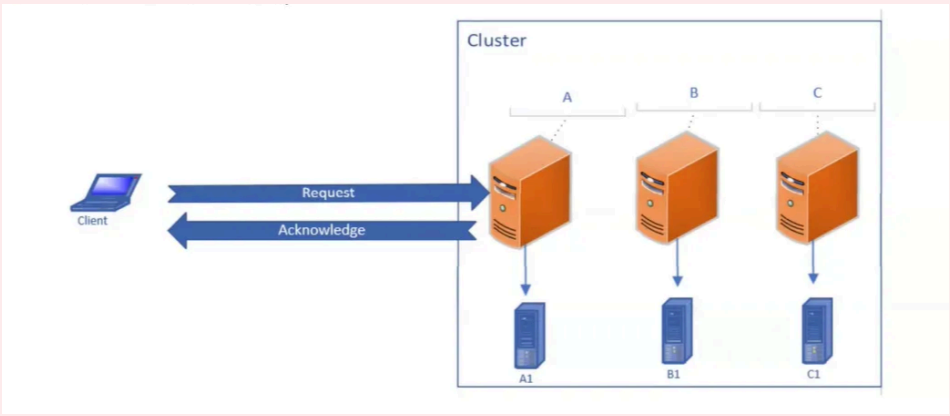
1. **Initial Setup:**
   - **Master A** with its replica **A1**
   - **Master B** with its replica **B1**
   - **Master C** with its replica **C1**
2. **After Failover:**
   - In this scenario, it appears that Master A has failed, and its replica A1 has been promoted to the new Master A.
   - The rest of the setup remains the same:
     - **Master B** with its replica **B1**
     - **Master C** with its replica **C1**

## Key Concepts

- **Master Nodes:** These are the primary nodes responsible for handling read and write operations. Each master node manages a subset of the hash slots in the cluster.
- **Replica Nodes:** These are secondary nodes that hold copies of the data from their respective master nodes. They can take over the role of the master if the master fails, ensuring high availability and fault tolerance.
- **Failover:** When a master node fails, one of its replicas can be automatically promoted to the new master, ensuring that the cluster remains operational without downtime.

## Summary

- The master-replica model in Redis Cluster ensures that data is replicated across multiple nodes, providing redundancy and allowing for seamless failover in case of node failures.
- Understanding this model helps in designing robust and reliable distributed systems using Redis.




### Clustering Overview

- **Redis cluster consistency:** This ensures that data remains consistent and up-to-date across all nodes in the cluster, even when there are multiple replicas.

### Visual Representation

The diagram shows a client interacting with a Redis Cluster consisting of three master nodes (A, B, C) and their respective replica nodes (A1, B1, C1).

### Client Interaction

- The client sends a **Request** to one of the master nodes (in this case, Node A).
- After processing the request, Node A sends an **Acknowledgment** back to the client.

### Data Replication

- The master nodes replicate their data to their respective replicas:
  - Master A replicates its data to Replica A1.
  - Master B replicates its data to Replica B1.
  - Master C replicates its data to Replica C1.

### Key Concepts

- **Consistency:** Ensures that all nodes in the cluster have the same data at any given time. This is crucial for maintaining the integrity of the data.
- **Replication:** The process of copying data from master nodes to replica nodes. This provides redundancy and allows for failover if a master node fails.
- **Client Requests:** Clients can send requests to any master node in the cluster. The master node processes the request and then replicates the changes to its replica.

## Problem:

**if the master node down before it send copy data to replica node the data not found because it destroyed with filler master node and aknowalge send directly after master node get data from client**

### Summary

- Redis Cluster uses replication to ensure data consistency across multiple nodes.

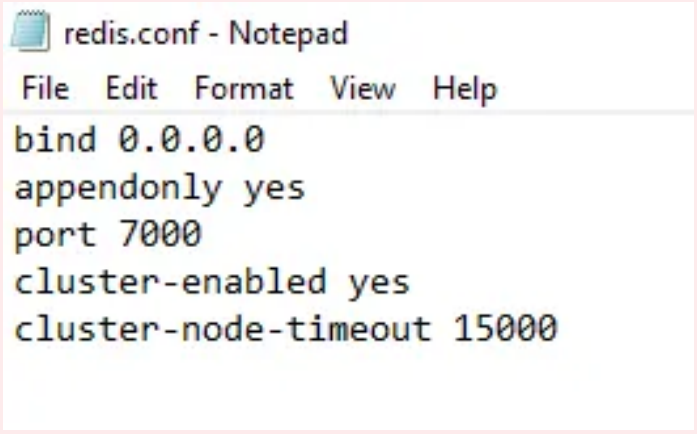
- When a client sends a request to a master node, the master node processes the request and acknowledges it back to the client.
- The master node then replicates the data to its replica, ensuring that the data remains consistent across the cluster.

# Create Cluster

ley's create 6 Nodes and `Redisinsights` for check its

1. create in your device 6 Folders


2. create in every folder a configuration file of every node that contain




1. `bind 0.0.0.0`

   👉 Allows Redis to accept connections from **any IP address** 🌍

2. `appendonly yes`

   ✅ Enables **data persistence** (Redis saves data to disk to recover it later) 💾

3. `port 7000`

   🔌 Redis will run on **port 7000** instead of the default (6379) 📡

4. `cluster-enabled yes`

   🧩 Turns on **Redis Cluster mode** (for scaling Redis across multiple nodes) 📈

5. `cluster-node-timeout 15000`

   ⏱️ Sets **timeout for cluster communication** to 15 seconds (15000 ms) 🔄

3. on `powershell` - create 6 redis instances , cluster enabled

```
docker run -e ALLOW_EMPTY_PASSWORD=yes -d --name redis-01 -p 7000:6379 -v C:\Clustering/01/redis.conf:/redis.conf redis:latest redis-server /redis.conf
docker run -e ALLOW_EMPTY_PASSWORD=yes -d --name redis-02 -p 7001:6379 -v C:\Clustering/02/redis.conf:/redis.conf redis:latest redis-server /redis.conf
docker run -e ALLOW_EMPTY_PASSWORD=yes -d --name redis-03 -p 7002:6379 -v C:\Clustering/03/redis.conf:/redis.conf redis:latest redis-server /redis.conf
docker run -e ALLOW_EMPTY_PASSWORD=yes -d --name redis-01-rep -p 7003:6379 -v C:\Clustering/04/redis.conf:/redis.conf redis:latest redis-server /redis.conf
```

```
docker run -e ALLOW_EMPTY_PASSWORD=yes -d --name redis-02-rep -p 7004:6379 -v C:\Clustering/0
5/redis.conf:/redis.conf redis:latest redis-server /redis.conf
docker run -e ALLOW_EMPTY_PASSWORD=yes -d --name redis-03-rep -p 7005:6379 -v C:\Clustering/0
6/redis.conf:/redis.conf redis:latest redis-server /redis.conf
```

4. on powershell - create network and add all 6 nodes to this network

```
docker network create redis_network
docker network connect redis_network redis-01
docker network connect redis_network redis-02
docker network connect redis_network redis-03
docker network connect redis_network redis-01-rep
docker network connect redis_network redis-02-rep
docker network connect redis_network redis-03-rep
```

5. go to docker - on each container - to run redis server

```
redis-server
```

6. on docker - on first container - open new tab - create cluster with one replica for each master

```
redis-cli --cluster create 172.18.0.2:7000 172.18.0.3:7001 172.18.0.4:7002 172.18.0.5:7003 172.18.0.6:7004 1
72.18.0.7:7005 --cluster-replicas 1
```

7. on docker - on any container - to check node availability

```
redis-cli -h 172.18.0.2 -p 7000 -c
ping
cluster slots
cluster info
```

**PING**

➡️ Checks if Redis is working.

🗣️ Redis replies: **PONG**

---

**CLUSTER SLOTS**

➡️ Shows which Redis server handles which part of the data.

📦 Like: "This range of data goes to that server."

---

**CLUSTER INFO**

➡️ Tells you how the Redis cluster is doing.

📋 Like: "Cluster is healthy, with X nodes, all working fine."

---

8. install Redisinsights from powershell

```
docker run --name redis-insight -v redisinsight:/db -p 8001:8001 redislabs/redisinsight:latest
```

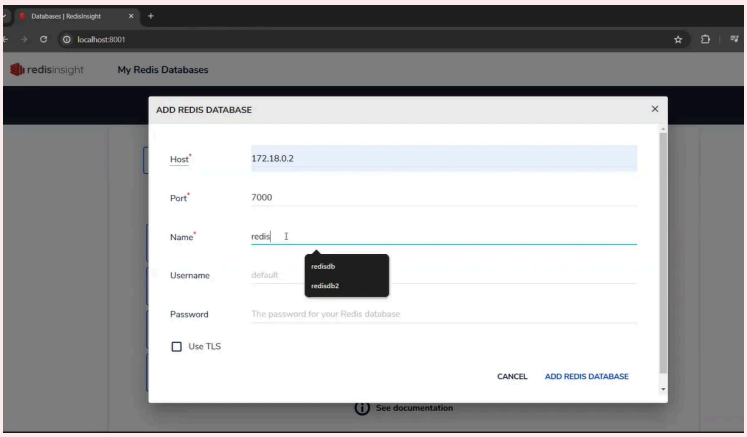
9. add it to same networks that all nodes in it

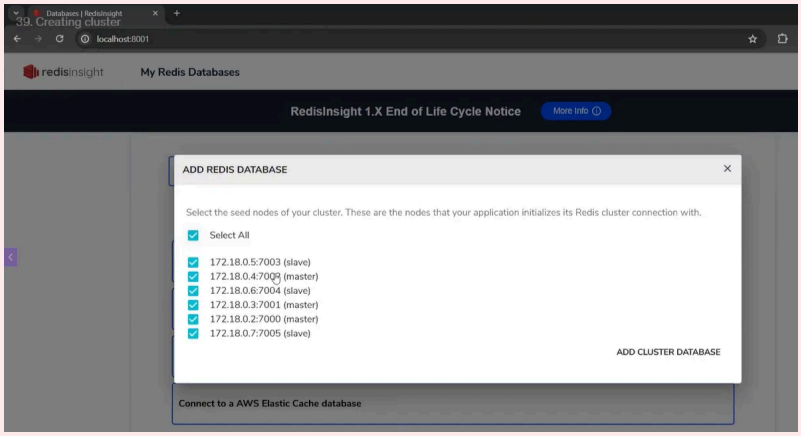
```
docker network connect redis_network redis-insight
```

10. open website

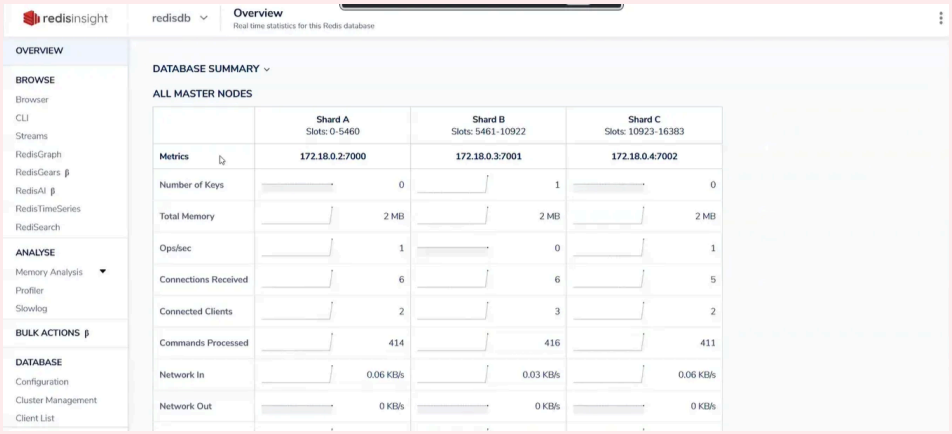
```
http://localhost:8001
```

change you already had Database a nd connect to it and add your host and port of any node and after this add all nodes




Here you connect from Redisinsights to your cluster nodes and can show all shardes with slotes



## 🔧Cluster RESET

### 💡 What is Cluster RESET in Redis?

When you're using a Redis Cluster (a group of Redis nodes working together), sometimes you need to **reset a node** — either to reconfigure it or fix an issue. That's where the `CLUSTER RESET` command comes in.

### 🛠️ Types of Reset:

1. **SOFT Reset** (this is the default):
   - Forgets all other nodes in the cluster.
   - Resets the slot configuration.
   - **Keeps the current node ID**.
   - Useful when you want a light reset without changing the node's identity.
2. **HARD Reset**:
   - Does everything that SOFT does.
   - **Generates a new node ID**.
   - Makes the node look like a brand new one.

### 🧹 Steps to perform a reset:

1. **FLUSHALL**
   - Clears all data stored in the node.
2. **CLUSTER RESET**
   - Resets the node's cluster configuration.

### 🔁 What happens after reset?

- The node **forgets all other nodes** in the cluster.
- The **slots (which determine where data is stored)** are cleared.
- If the node was a **replica**, it becomes an **empty master**.
- If you used **HARD reset**, a **new node ID** is created.
- If you used **SOFT reset**, the **existing node ID** remains the same.

### 🎯 Example Use Case:

Imagine you have 3 nodes in a Redis Cluster, and you want to remove one node and reuse it elsewhere:

- Run `FLUSHALL` to clear its data.
- Run `CLUSTER RESET` to remove all cluster settings.
- The node is now clean and ready to join a new cluster.

### 🔷 1. Re-create Cluster

This means you're starting fresh—creating a new Redis cluster from scratch.

**Example:**
Let's say you have 3 Redis nodes running on:

- 192.168.1.1:7000
- 192.168.1.2:7001
- 192.168.1.3:7002

You start Redis on each node with cluster enabled:

```
redis-server /etc/redis/7000.conf
redis-server /etc/redis/7001.conf
redis-server /etc/redis/7002.conf
```

Then use the `redis-cli` to create the cluster:

```
redis-cli --cluster create 192.168.1.1:7000 192.168.1.2:7001 192.168.1.3:7002 --cluster-replicas 0
```

---

### 🔷 2. Cluster Reset

This command is used when you want to wipe a Redis node's cluster configuration.

There are two types:

- `CLUSTER RESET HARD` – resets everything, including node ID and configuration
- `CLUSTER RESET SOFT` – resets configuration but keeps node ID

**Example:**

```
redis-cli -p 7000 CLUSTER RESET HARD
```

Use this if something went wrong and you want to reset the node.

---

### 🔷 3. Cluster Meet

This command connects one Redis node to another, so they recognize each other and become part of the same cluster.

**Command Format:**

```
CLUSTER MEET <ip> <port>
```

**Example:**
You have a node running at 192.168.1.1:7000 (node A), and you want it to join node B at 192.168.1.2:7001:

```
redis-cli -p 7000 CLUSTER MEET 192.168.1.2 7001
```

This tells node A to meet node B and form a cluster.

In the image, you can see:

- Node A meets Node B
- Node B meets Node C

- Node C meets Node A
  That way, all nodes are aware of each other (forming a ring-like structure).

### 🔷 4. Cluster Replicate

After forming the cluster, you can create replicas (slaves) for fault tolerance.

**Command Format:**

```
CLUSTER REPLICATE <master_node_id>
```

**Example:**
Suppose you have a node (7003) and want it to replicate a master node with ID:

```
b3f5c43e3ebf29e4ffad1c5b7b7b324a1c6615d9
```

Run:

```
redis-cli -p 7003 CLUSTER REPLICATE b3f5c43e3ebf29e4ffad1c5b7b7b324a1c6615d9
```

Now 7003 will act as a replica of that master node.

### 🔷 5. Cluster Fix

This is not an actual Redis command, but usually refers to using the `--cluster fix` option with `redis-cli` to solve configuration issues.

**Example:**
If your cluster has some inconsistencies (slots not properly assigned, etc), run:

```
redis-cli --cluster fix 192.168.1.1:7000
```

It will automatically fix problems like:

- Unassigned slots
- Multiple nodes claiming the same slot
- Nodes not correctly connected

### 🔄 Recap Flow:

1. **Reset all nodes** (optional): `CLUSTER RESET HARD`
2. **Start nodes** with `redis-server`
3. **Meet nodes** using `CLUSTER MEET`
4. **Replicate nodes** using `CLUSTER REPLICATE`
5. If anything goes wrong, use `-cluster fix`

## 🔁 Redis Cluster Failover

### 🔶 1. Automatic Failover

- Handled by the **Redis Cluster itself**.
- If a master node fails and is unreachable, the cluster will:
  - Detect the failure (via gossip protocol and majority voting).
  - Promote one of the **replica nodes** to be the new master.
- No manual intervention is needed.

➡️ **Useful in production** where high availability is critical.

---

### 🔶 2. Manual Failover

### ✅ Command:

```
CLUSTER FAILOVER
```

### ✅ Key Points:

- **Can only be sent to a Redis replica node**.
- It **forces the replica to take over as the new master** of its master.
- Useful during **maintenance** or **controlled role switching**.

### 📌 Example Scenario:

You want to perform maintenance on the master node and want its replica to take over:

1. Run this on the **replica node**:

```
redis-cli -p 7003 CLUSTER FAILOVER
```

1. The replica sends a message to the cluster:
   - Promotes itself to **master**
   - Starts handling the slots previously handled by the original master

---

### ⚠️ Notes:

- The command won't work on a **master node** — only replicas can trigger it.
- Forcing a failover when the master is still healthy might temporarily split the cluster or confuse clients, so use it with caution.

---

# 🔁 Cluster Failover – Manual

## 🔶 Manual Failover

You manually promote a replica to a master using:

```
CLUSTER FAILOVER
```

## 🔶 Options

## ✅ FORCE

```
CLUSTER FAILOVER FORCE
```

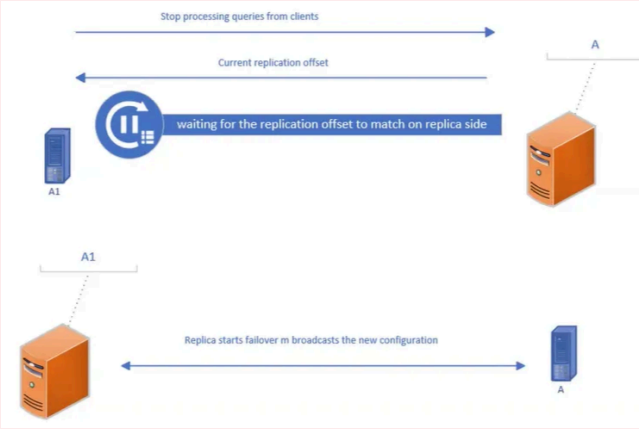
- **Used when the master is still reachable.**
- Forces the replica to become the master **without waiting** for the replication offset to match.
- Use with caution — may lead to **data loss or inconsistencies**.

## ✅ TAKEOVER

```
CLUSTER FAILOVER TAKEOVER
```

- Used **in situations where the master is unreachable** (e.g. crashed).
- Replica immediately takes over **regardless of offset** or master state.
- Common during **emergencies or testing**.

# 🖼️ Diagram Explanation




### Step-by-Step:

1. **Client Query Handling Stops**

   → The replica stops accepting new client requests to maintain consistency.

2. **Replication Offset Sync**
   → The replica waits until its replication offset
   **matches the master's**, meaning it has received all data.

3. **Offset Match Achieved**
   → Once offset is matched, the replica is ready to become the master.
4. **Failover Starts**
   → Replica broadcasts to the cluster that it is taking over.
5. **New Master Announced**
   → All nodes update their configuration to reflect the new master.

---

## 🔄 Cluster Reshard

### 🔶 What is Resharding?

Resharding means **redistributing hash slots** among nodes in a Redis Cluster. This is often done to:

- Balance load,
- Add new nodes,
- Remove old nodes,
- Scale horizontally.

Redis Cluster has **16,384 hash slots** in total, and these slots are divided among the master nodes. When resharding, you're **moving some of these slots from one node to another**.

---

### 🔶 Command

```
redis-cli --cluster reshard <ip:port>
```

- `<ip:port>` refers to **any node** in the cluster (doesn't matter which one).
- This command launches an **interactive wizard** to:
  - Ask how many slots to move,
  - Specify the **target node** (receiving slots),
  - Optionally choose the **source node(s)** (donating slots),
  - Confirm the changes.

---

### 🧠 Example Flow

```
redis-cli --cluster reshard 127.0.0.1:7000
```

Then you'll be prompted for:

1. Number of slots to move.
2. Target node ID.
3. Source node(s).

It will automatically handle:

- Migrating the slot metadata.
- Moving keys between nodes.

---

Would you like a walkthrough or simulation of a real resharding process (maybe with Docker)?

# 🧩 Cluster — Add Node

## ✅ Adding New Node as a Master

1. must create new container of new node and redis-conf file

```
redis-cli --cluster add-node 172.18.0.9:7006 172.18.0.2:7000
```

- `172.18.0.9:7006`: the new node you want to add.
- `172.18.0.2:7000`: an existing node already in the cluster.

This command adds `172.18.0.9:7006` as a **new master** node.

---

## 🧬 Adding New Node as a Replica (Slave)

```
redis-cli --cluster add-node 172.18.0.9:7006 172.18.0.2:7000 --cluster-slave --cluster-master-id <master node id>
```

- `-cluster-slave`: tells Redis to add this node as a **replica**.
- `-cluster-master-id <master node id>`: the ID of the **master node** this replica will be assigned to.

To get the master node ID:

```
redis-cli -c -p 7000 cluster nodes
```

Look for the ID of the master node you want the replica to follow.

---

Would you like a practical example using Docker or Redis CLI to simulate adding a node?

## 🔥 Cluster — Remove Node

### 🧹 Steps to Remove a Node

1. **Delete Node from the Cluster**

```
redis-cli --cluster del-node 172.18.0.2:7000 <node-id>
```

- `172.18.0.2:7000`: address of a cluster node you're contacting.
- `<node-id>`: the **ID of the node** you want to remove.

1. **Force a Node to Forget Another Node**

```
redis-cli --cluster call 172.18.0.2:7000 cluster forget <node-id>
```

- This is used to **tell the cluster** to forget about a node that has been removed.

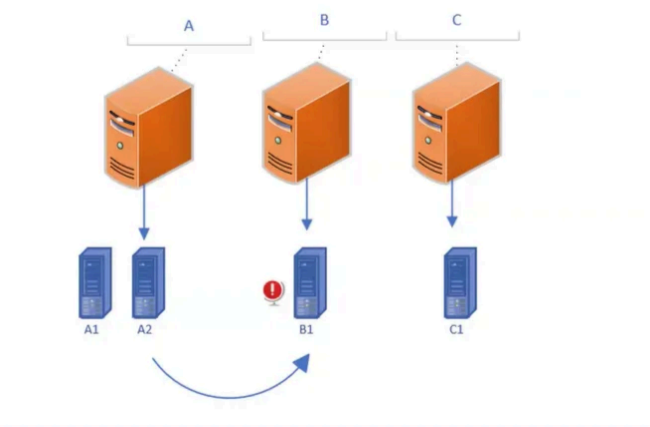
---

### 📝 Notes:

- Before removing a **master node**, you should migrate its slots to other masters using:

```
redis-cli --cluster reshard ...
```

## 🔄 Cluster — Replica Migration




### 🧠 What is Replica Migration?

> Replica migration is the automatic reconfiguration of replicas in a Redis Cluster to improve reliability.

---

### 🧩 Visual Explanation (from the diagram):

- **A, B, C** are three master nodes.
- Each master has one or more **replicas** for high availability.

**Before Migration:**

- Master B has only one replica: `B1`.
- That replica (B1) is marked with a red exclamation mark — maybe it's failed or unreachable.

**Replica Migration Happens:**

- To maintain reliability, the cluster automatically reassigns a replica (e.g., `A2`) to Master B.
- So now B is backed up by a working replica (`A2`) from Master A.

---

### ✅ Why is this important?

- Ensures **failover** and **resilience**.
- Keeps the cluster **balanced** even when a replica node fails.

---

## Working in DotNet 6

## NuGet Package Overview

- **StackExchange.Redis:** This is a popular NuGet package that provides a client library for connecting to and interacting with Redis from .NET applications.

### Key Components

1. **ConnectionMultiplexer:**
   - The `ConnectionMultiplexer` is the primary class used to manage connections to Redis.
   - It handles connection pooling, reconnection logic, and other low-level details, allowing you to focus on using Redis commands.
2. **Interactive Connection:**
   - An interactive connection allows you to execute Redis commands directly and interactively.
   - You can use this to perform operations like setting and getting values, managing keys, and more.
3. **Subscription Connection (Pub/Sub):**
   - The subscription connection is used for implementing the Pub/Sub (Publish/Subscribe) pattern.
   - With this, you can subscribe to channels and receive messages published to those channels, enabling real-time communication between different parts of your application or between different applications.

### Summary

- The `StackExchange.Redis` NuGet package provides essential tools for integrating Redis into .NET applications.
- The `ConnectionMultiplexer` manages connections efficiently.
- Interactive connections allow direct command execution.
- Subscription connections enable Pub/Sub functionality for real-time messaging.

## Connecting to Redis

### Simple Connection String Approach

```
var redis = ConnectionMultiplexer.Connect("localhost:6379,password=foobar");
```

- This line establishes a connection to a Redis server running on `localhost` at port `6379` with the specified password (`foobar`).
- The `ConnectionMultiplexer` class is used to manage the connection, handle reconnections, and provide a high-level API for interacting with Redis.

### Configuration Options Approach

```
var options = new ConfigurationOptions
{
   EndPoints = { "redis-1:6379" },
   Password = "foobar"
};
var redis2 = ConnectionMultiplexer.Connect(options);
```

- This block demonstrates an alternative way to configure the connection using a `ConfigurationOptions` object.
- `EndPoints` specifies the Redis server endpoint. In this case, it connects to a server named `redis-1` at port `6379`.

- `Password` sets the authentication password for the Redis server.
- The `ConnectionMultiplexer.Connect(options)` method then uses these configuration options to establish the connection.

.

### 1. ConnectionMultiplexer

The `ConnectionMultiplexer` is the central object that manages connections to Redis servers. It is designed to be shared and reused throughout your application.

```
using StackExchange.Redis;
using System;

class Program
{
    static void Main()
    {
        // Create a ConnectionMultiplexer
        ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");

        // Use the connection
        IDatabase db = redis.GetDatabase();

        // Set a key
        db.StringSet("mykey", "myvalue");

        // Get the key
        string value = db.StringGet("mykey");
        Console.WriteLine(value); // Output: myvalue

        // Close the connection
        redis.Close();
    }
}
```

### 2. Database

The `IDatabase` interface represents a Redis database. It allows you to execute commands against the Redis database.

```
using StackExchange.Redis;
using System;

class Program
{
    static void Main()
    {
        ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");
        IDatabase db = redis.GetDatabase();

        // String operations
        db.StringSet("greeting", "Hello, Redis!");
        string greeting = db.StringGet("greeting");
        Console.WriteLine(greeting); // Output: Hello, Redis!

        // List operations
        db.ListRightPush("mylist", "item1");
```

```
        db.ListRightPush("mylist", "item2");
        string item = db.ListLeftPop("mylist");
        Console.WriteLine(item); // Output: item1
    }
}
```

### 3. Server

The `IServer` interface provides access to server-specific commands and information.

```
using StackExchange.Redis;
using System;

class Program
{
    static void Main()
    {
        ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");
        IServer server = redis.GetServer("localhost", 6379);

        // Get server info
        var info = server.Info();
        foreach (var section in info)
        {
            Console.WriteLine($"{section.Key}:");
            foreach (var item in section)
            {
                Console.WriteLine($"  {item.Key}: {item.Value}");
            }
        }

        // Flush the database
        server.FlushDatabase();
    }
}
```

### 4. Subscriber

The `ISubscriber` interface allows you to subscribe to channels and receive messages published to those channels.

```
using StackExchange.Redis;
using System;

class Program
{
    static void Main()
    {
        ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");
        ISubscriber sub = redis.GetSubscriber();

        // Subscribe to a channel
        sub.Subscribe("messages", (channel, message) => {
            Console.WriteLine($"Received: {message}");
        });
```

```
        // Publish a message
        sub.Publish("messages", "Hello, subscribers!");

        // Keep the application running to receive messages
        Console.ReadLine();
    }
}
```

### 5. Transaction

The `ITransaction` interface allows you to execute multiple commands as a single atomic operation.

```
using StackExchange.Redis;
using System;

class Program
{
    static void Main()
    {
        ConnectionMultiplexer redis = ConnectionMultiplexer.Connect("localhost");
        IDatabase db = redis.GetDatabase();

        // Create a transaction
        ITransaction transaction = db.CreateTransaction();

        // Add commands to the transaction
        transaction.StringSetAsync("key1", "value1");
        transaction.StringSetAsync("key2", "value2");

        // Execute the transaction
        bool committed = transaction.Execute();
        Console.WriteLine($"Transaction committed: {committed}");
    }
}
```

### Connecting to Redis Cluster

### Configuration Options

```
var options = new ConfigurationOptions
{
    // add and update parameters as needed
    EndPoints = { "redis-1:6379", "redis-2:6379", "redis-3:6379" }
};
```

- This block creates a new instance of `ConfigurationOptions`, which is used to configure the connection to the Redis cluster.
- only add a master nodes
- The `EndPoints` property specifies the list of Redis server endpoints that form the cluster. In this case, it includes three servers: `redis-1:6379`, `redis-2:6379`, and `redis-3:6379`.
- You can add or update other parameters as needed, such as authentication settings, timeouts, and more.

### Establishing the Connection

```
var muxer = ConnectionMultiplexer.Connect(options);
```

- The `ConnectionMultiplexer.Connect(options)` method establishes a connection to the Redis cluster using the specified configuration options.
- The `muxer` variable holds the `ConnectionMultiplexer` instance, which manages the connection and provides a high-level API for interacting with the Redis cluster.

## ✅ store sessions in Redis

### 📦 Step 1: Install the Redis package

In your terminal or Package Manager Console:

```
dotnet add package Microsoft.Extensions.Caching.StackExchangeRedis
```

### ⚙️ Step 2: Configure Redis in `Program.cs`

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.Extensions.Caching.StackExchangeRedis;

var builder = WebApplication.CreateBuilder(args);

// Add Redis distributed cache
builder.Services.AddStackExchangeRedisCache(options =>
{
    options.Configuration = "localhost:6379"; // Change if Redis is hosted remotely
    options.InstanceName = "MyAppSession:";
});

// Add session support
builder.Services.AddSession(options =>
{
    options.Cookie.Name = ".MyApp.Session";
    options.IdleTimeout = TimeSpan.FromMinutes(30);
    options.Cookie.HttpOnly = true;
    options.Cookie.IsEssential = true;
});

// Add Razor Pages
builder.Services.AddRazorPages();

var app = builder.Build();

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

// Enable session before endpoints
app.UseSession();

app.UseAuthorization();

app.MapRazorPages();

app.Run();
```

### 📄 Step 3: Using Session in a Razor Page ( `Pages/Index.cshtml.cs` )

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

public class IndexModel : PageModel
{
    public void OnGet()
    {
        // Store data in session
        HttpContext.Session.SetString("MyKey", "Hello from Redis session!");

        // Example: Incrementing a counter in session
        int count = HttpContext.Session.GetInt32("Counter") ?? 0;
        count++;
        HttpContext.Session.SetInt32("Counter", count);
    }

    public IActionResult OnPostClear()
    {
        HttpContext.Session.Clear();
        return RedirectToPage();
    }

    public string? Message => HttpContext.Session.GetString("MyKey");
    public int Counter => HttpContext.Session.GetInt32("Counter") ?? 0;
}
```

### 🖼️ Step 4: UI for Display & Clearing Session ( `Pages/Index.cshtml` )

```
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<h2>Session State Using Redis</h2>

<p><strong>Message:</strong> @Model.Message</p>
<p><strong>Counter:</strong> @Model.Counter</p>

<form method="post">
    <button type="submit" asp-page-handler="Clear">Clear Session</button>
</form>
```

### ✅ Verify Redis Sessions

To confirm it's working:

1. Run the project.
2. Check Redis keys using a Redis client or CLI:

```
redis-cli
```

```
keys *
```

You should see keys like:

```
MyAppSession:YourSessionIdHere
```