MEMORY STATS
MEMORY DOCTOR
MODULE LIST ---> yüklenmiş modülleri listeler.
ACL CAT -- ACL kategorilerini listeler.
ACL LOG ---> ACL kuralları dolayısıyla oluşturulan güvenlik olaylarını listeler.
ACL USERS
ACL WHOAMI
ACL DELUSER userName
COMMAND COUNT
COMMAND LIST
ROLE

replicaof master-ip master-port

slaveof no one
info replication

ach genpass
ach genpass 5

COMMAND COUNT
DBSIZE
ROLE
BGSAVE -----> verliere dicke yazar

PUBLISH
SUBSCRIBE
PSUBSCRIBE
PUNSUBSCRIBE

pubsub Channels
pubsub numsub

pubsub numpat
Komut çalıştırıldığında, Redis size **aktif desen aboneliklerinin** sayısını gösterir. Eğer hiçbir istemci `PSUBSCRIBE` komutuyla herhangi bir desene abone olmamışsa, çıktı 0 our

XADD mystream * temp 20.5 pressure 1013

1. Temel Veri Okuma (Bir Stream'den)

```
XREAD STREAMS mystream 0
```
Bu komut, `mystream` adlı stream'den, ID'si `0`'dan (yani stream'in en başından) itibaren tüm verileri okur. Eğer yeni veri eklenmişse, bu veriler de komut çalıştırıldığında döndürülecektir.

2. `BLOCK` Kullanarak Veri Okuma (Bekleme Süresi Belirterek)

```
XREAD BLOCK 5000 STREAMS mystream 0
```
Bu komut, `mystream` stream'inden veri okumaya başlar ve 5 saniye boyunca (5000 milisaniye) yeni veri olup olmadığını kontrol eder. Eğer 5 saniye içinde veri eklenirse, bu veriler döndürülür. Veri eklenmezse, komut zaman aşımına uğrar ve boş bir sonuç döner.

3. Birden Fazla Stream'den Veri Okuma

```
XREAD STREAMS mystream1 mystream2 0 0
```
Bu komut, hem `mystream1` hem de `mystream2` stream'lerinden veri okumak için kullanılır. Her iki stream'den de veriler, ID `0`'dan itibaren okunur.

4. `COUNT` ile Belirli Sayıda Mesaj Okuma

```
XREAD COUNT 5 STREAMS mystream 0
```
Bu komut, `mystream` stream'inden yalnızca ilk 5 mesajı okur. Eğer stream'de 5'ten fazla mesaj varsa, yalnızca ilk 5 mesaj alınır.

5. Yeni Eklenen Mesajları Okuma

```
XREAD STREAMS mystream >
```
Bu komut, `mystream` adlı stream'den yalnızca **yeni eklenen** mesajları okur. Bu komutun anlamı şudur: En son okunan ID'den sonra eklenen mesajlar getirilecektir. Eğer yeni bir mesaj eklenmemişse, komut hiçbir şey döndürmez.

6. `BLOCK` ve `COUNT` Birlikte Kullanımı

```
XREAD BLOCK 5000 COUNT 10 STREAMS mystream 0
```
Bu komut, `mystream` stream'inde en az 1 yeni mesaj olana kadar 5 saniye boyunca bekler. Eğer 5 saniye içinde yeni mesajlar eklenirse, bu mesajları döndüren komut 10 mesaj kadar alır (eğer varsa).

Örnek Kullanımlar

1. Tek Bir Mesajı Silmek

`XDEL mystream 1622500000000-0`
Bu komut, `mystream` adlı stream'den, ID'si `1622500000000-0` olan mesajı siler.

2. Birden Fazla Mesajı Silmek

`XDEL mystream 1622500000000-1 1622500000000-2`
Bu komut, `mystream` adlı stream'den, ID'leri sırasıyla `1622500000000-1` ve `1622500000000-2` olan iki mesajı siler.

3. Bir Stream'in Sonunda Eklenen Mesajları Silmek

`XDEL mystream 1622500000000-3`
Bu komut, `mystream` adlı stream'den, son eklenen mesaj (ID: `1622500000000-3`) siler.

`XDEL` Komutunun Çıktısı

Komut başarılı bir şekilde çalıştıktan sonra, Redis size silinen mesaj sayısını döndürecektir.

Örnek çıktı:

`(integer) 1`
Bu çıktı, belirtilen ID'ye sahip bir mesajın başarıyla silindiğini gösterir.

Eğer belirtilen ID'ye sahip bir mesaj stream'de bulunmazsa, Redis bu durumu göz ardı eder ve 0 döndürür.

XLEN mystream

XRANGE mystream - +

XRANGE mystream - + COUNT 2

//en fazla 2 sonuç getirir.

1. Basit Bir Tüketici Grubu Oluşturma

Bir stream'den veri okumaya başlamak için önce bir tüketici grubu oluşturmanız gerekir. Örneğin, `mystream` adlı stream için bir tüketici grubu oluşturmak için:

```
XGROUP CREATE mystream mygroup 0
```

Bu komut, `mystream` adlı stream için `mygroup` adlı bir tüketici grubu oluşturur ve gruptaki tüketiciler, stream'in başındaki tüm mesajları okumaya başlar.

2. Yeni Mesajlardan Başlayarak Tüketici Grubu Oluşturma

Eğer tüketici grubunun yalnızca yeni mesajlardan okumaya başlamasını istiyorsanız, ID parametresini `$` olarak belirtebilirsiniz:

```
XGROUP CREATE mystream mygroup $
```

Bu komut, `mygroup` adlı tüketici grubunu oluşturur ve gruptaki tüketiciler, sadece stream'e **yeni** eklenen mesajları okumaya başlar.

3. Stream Olmayan Bir Adreste Tüketici Grubu Oluşturma

Eğer `mystream` adlı stream mevcut değilse ve yine de bir tüketici grubu oluşturmak istiyorsanız, `MKSTREAM` opsiyonunu kullanabilirsiniz:

```
XGROUP CREATE mystream mygroup 0 MKSTREAM
```

Bu komut, `mystream` adlı stream mevcut değilse, Redis'in bu stream'i oluşturmasını sağlar ve `mygroup` adlı bir tüketici grubu oluşturur. Tüketici grubu, stream'in başından itibaren mesajları okumaya başlar.

`XGROUP CREATE` Komutunun Çıktısı

Başarılı bir şekilde bir tüketici grubu oluşturduğunuzda, Redis size bir onay döndürür. Genellikle şu şekilde bir çıktı alırsınız:

```
OK
```

Eğer belirtilen stream zaten bir tüketici grubuna sahipse, Redis hata döndürecektir. Örneğin:

```
BUSYGROUP Consumer Group name already exists
```

```
XGROUP DESTROY mystream mygroup
```

`XREADGROUP` Örneği

Bir tüketici grubunun nasıl çalıştığını ve `XREADGROUP` komutunun nasıl kullanıldığını göstermek için bir örnek üzerinden gidelim.

```
XADD mystream * field1 value1
XADD mystream * field2 value2
```
Bir tüketici grubu (`mygroup`) ve bir tüketici (`consumer1`) oluşturalım:

```
XGROUP CREATE mystream mygroup $ MKSTREAM
```
- `mystream`: Stream adı.
- `mygroup`: Grubun adı.
- `$`: Sonraki mesajlardan başlamak.
- `MKSTREAM`: Eğer stream henüz yoksa, bu komut stream'i oluşturur.
-

Mesajları okuma (Tüketici Grubu Kullanarak):

Tüketici grubu `mygroup` ve tüketici `consumer1` ile mesaj okumak için:

```
XREADGROUP GROUP mygroup consumer1 BLOCK 1000 COUNT 2 STREAMS mystream >
```
Bu komutun işleyişi şudur:

- **`GROUP mygroup consumer1`**: `mygroup` tüketici grubundan `consumer1` adlı tüketiciyi belirtir.
- **`BLOCK 1000`**: Eğer yeni mesaj yoksa, 1 saniye boyunca (1000 milisaniye) bekler.
- **`COUNT 2`**: En fazla 2 mesaj okunur.
- **`STREAMS mystream >`**: `mystream` adlı stream'den, son okunmuş mesajdan sonra gelen mesajları (`'>'` ile yeni mesajlar) okur.

Tüketici Grubu ve Mesaj İşleme

- Her tüketici, stream üzerinde yalnızca **henüz işlenmemiş** mesajları alır. Mesajlar her bir tüketici tarafından alındıktan sonra, "acknowledged" (onaylanmış) olarak işaretlenir.

- Tüketiciler, bir mesajı aldıktan sonra, mesajı işlediklerinde `XACK` komutunu kullanarak bu mesajı "acknowledge" (onaylama) ederler. Bu, mesajın grubun geri kalan üyeleri tarafından işlenmeyeceği anlamına gelir.

Mesaj Onaylama (Acknowledge)

Mesajın onaylanması için şu komutu kullanırız:

```
XACK mystream mygroup <message_id>
```
Bu komut, belirtilen mesajın grubun bir tüketicisi tarafından işlendiğini ve artık başka tüketiciler tarafından alınmayacağını belirtir.

1. Stream ve Tüketici Grubu Oluşturma

Öncelikle bir stream oluşturduğumuzu varsayalım:

```
XADD mystream * field1 value1
XADD mystream * field2 value2
```
Ardından, bir **tüketici grubu** oluşturuyoruz:

```
XGROUP CREATE mystream mygroup $ MKSTREAM
```
Bu komut:

- `mystream`: Stream adı.
- `mygroup`: Tüketici grubunun adı.
- `$`: Başlangıç olarak yeni mesajlardan itibaren okuma.
- `MKSTREAM`: Eğer `mystream` adlı stream yoksa, bu komut stream'i oluşturur.

2. Yeni Bir Tüketici Ekleme

`XGROUP CREATECONSUMER` komutunu kullanarak, `mygroup` adlı gruba `consumer1` adlı bir tüketici ekliyoruz:

```
XGROUP CREATECONSUMER mystream mygroup consumer1
```
- `mystream`: Stream adı.
- `mygroup`: Tüketici grubunun adı.
- `consumer1`: Yeni ekleyeceğiniz tüketicinin adı.

Bu komut başarılı olduğunda, `consumer1` artık `mygroup` grubunun bir parçası olarak, `mystream` stream'inden mesajları okuyabilir.

3. Tüketici Grubundan Mesaj Okuma

Şimdi, `consumer1` adlı tüketici, `mygroup` grubunun bir parçası olarak mesajları okumaya başlayabilir:

```
XREADGROUP GROUP mygroup consumer1 STREAMS mystream >
```
Bu komut, `mygroup` grubundaki `consumer1` tüketicisinin, `mystream` stream'inden yeni gelen mesajları okumasını sağlar. `'>'` işareti, yalnızca daha önce işlenmemiş mesajların okunacağını belirtir.

`XGROUP CREATECONSUMER` Komutunun Amacı

`XGROUP CREATECONSUMER` komutunun amacı, bir tüketici grubuna yeni bir tüketici eklemektir. Redis Streams ile çoklu tüketici grupları arasında iş yükünü paylaşmak ve mesajları daha verimli bir şekilde işlemek için bu komut kullanılır.

`XGROUP CREATE` ile Farkı

- **`XGROUP CREATE`**: Bu komut, bir tüketici grubunu oluşturur. Ancak, bu komut sadece grubu oluşturur, herhangi bir tüketici eklemez. Tüketici grubunun oluşturulmasından sonra, gruba mesajları işlemek için bir tüketici eklenmesi gerekir.

- **`XGROUP CREATECONSUMER`**: Bu komut, gruba **tüketici ekler**. Tüketici grubuna bir tüketici eklemek için kullanılır. Eğer grup zaten var ise ve gruba yeni bir tüketici eklemek isteniyorsa, bu komut gekeilter.

XGROUP DELCONSUMER mystream mygroup consumer1

`XINFO CONSUMERS <stream_name>`
Bu komut, belirli bir akıştaki tüm tüketiciler hakkında bilgi verir. Dönen sonuç, her bir tüketici hakkında aşağıdaki bilgileri içerir:

- **name**: Tüketicinin adı.
- **pending**: Tüketicinin işlemeyi bekleyen mesajların sayısı.
- **idle**: Tüketicinin en son mesajı aldığı zamandan itibaren geçen süre (ms cinsinden).
- **last delivered**: Tüketicinin en son aldığı mesajın ID'si.
-

Örnek:

`XINFO CONSUMERS mystream group1`

`XINFO GROUPS <stream_name>`
Burada, `<stream_name>` sorgulamak istediğiniz stream'in adıdır.

Dönüş Verileri:

Komut başarılı bir şekilde çalıştığında aşağıdaki bilgileri döndürebilir:

1. **name**: Tüketici grubunun adı.
2. **consumers**: Gruba ait aktif tüketici sayısı.
3. **pending**: Grubun işleme bekleyen toplam mesaj sayısı.
4. **last-delivered-id**: Grubun en son işlediği mesajın ID'si.
5. **last-delivered-time**: En son işlenen mesajın zaman damgası.
6. **lag**: Tüketici grubunun işleme bekleyen mesaj sayısı.

Örnek:

`XINFO GROUPS mystream`
Bu komut, `mystream` adlı stream'deki tüm tüketici gruplarının durumunu gösterir.

Örnek:

`XINFO STREAM mystream`

Örnek Kullanımlar:

1. **Tüketici Grubunun Bekleyen Mesajları Gösterme:**

```
XPENDING mystream mygroup
```

Bu komut, `mystream` stream'indeki `mygroup` adlı tüketici grubunun işlenmemiş (pending) mesajlarını gösterir.

2. **Belirli Bir Mesaj Aralığını Gösterme:**

```
XPENDING mystream mygroup 1683775792464-0
1683775890000-0
```

Bu komut, mystream stream'indeki mygroup grubunun 1683775792464-0 ile 1683775890000-0 arasındaki ID'lere sahip işlenmemiş mesajları gösterir.

3. **Belirli Bir Tüketiciye Ait Bekleyen Mesajları Gösterme:**

```
XPENDING mystream mygroup 0 1000 10 consumer1
```

Bu komut, `mystream` stream'indeki `mygroup` grubundaki `consumer1` adlı tüketiciye ait 10 bekleyen mesajı görüntüler.

XCLAIM Örnek Kullanımı:

Diyelim ki bir "mystream" adlı bir stream var ve bu stream'in içinde bazı mesajlar var. Bu mesajlar bir tüketici grubu tarafından işleniyor, ancak bazı mesajlar zamanında işlenmemiş. Bu durumda, bu mesajları başka bir tüketiciye devredebiliriz.

Adımlar:

Stream'e mesaj ekleyelim (bu adım daha önce yapılmış olabilir):

```
XADD mystream * key1 value1

XADD mystream * key2 value2
```

Bir tüketici grubu oluşturalım ve bir tüketici mesajları almaya başlasın:

```
XGROUP CREATE mystream mygroup $ MKSTREAM
```

Mesajı işleyen bir tüketici oluşturup (tüketici1) birkaç mesaj alalım:

```
XREADGROUP GROUP mygroup consumer1 COUNT 2 BLOCK 0 STREAMS mystream >
```

Tüketici1 mesajları işleyemiyor veya gecikiyor. Bu durumda, başka bir tüketici (tüketici2) mesajları devralabilir:

```
XCLAIM mystream mygroup consumer2 60000 1234567890
```

Bu komut şu anlamı taşır:

mystream: Mesajların bulunduğu stream.
mygroup: Mesajların işlendiği tüketici grubu.
consumer2: Yeni tüketici (tüketici2) bu mesajı devralacak.
60000: Son işleme zamanından itibaren 60 saniyelik bir süre (min-idle-time).
1234567890: Devredilecek mesajın ID'si.

Bu komut, 60 saniyedir işlenmemiş mesajı tüketici2'ye devreder.

XCLAIM ile Mesaj Devretmenin Faydaları:

Hata Toleransı: Bir tüketici mesajları işleyemezse, başka bir tüketici devralabilir.
Zaman Aşımı Yönetimi: Belirli bir süre içinde işlenmeyen mesajları devrederek sistemin verimli çalışmasını sağlarsınız.
Daha Esnek Tüketici Grupları: Aynı mesajı birden fazla tüketiciye işletebilir ve her biri farklı zamanlarda devralabilir.
Önemli Notlar:

XCLAIM, sadece **consumer group**'ları ve **stream** veritabanı yapısını destekler. Eğer bir tüketici mesajı alıp işlediyse, o mesajın ID'si bir daha başka bir tüketiciye devredilemez.
Komutun etkinliği, **min-idle-time** parametresine bağlıdır. Bu değer, bir mesajın ne kadar süre boyunca işlenmeden beklediğini belirtir.

Bu komut, Redis Streams ile çalışan tüketici gruplarında zaman aşımı nedeniyle kaybedilen mesajları diğer tüketicilere devretmek için kullanışlıdır.

XAUTOCLAIM Örnek Kullanımı:

Örnek 1: Basit XAUTOCLAIM Kullanımı

Diyelim ki bir "mystream" adında bir stream var ve "mygroup" adlı bir tüketici grubuna bağlı bazı tüketiciler bu mesajları işlemeye çalışıyorlar. Ancak, bazı mesajlar işlemeyi başaramayan bir tüketici nedeniyle zaman aşımına uğramış.

Tüm işlenmemiş mesajları **consumer2**'ye devretmek için şu komutu kullanabiliriz:

```
XAUTOCLAIM mystream mygroup consumer2 60000 0
```

Bu komut şu şekilde çalışır:

- **mystream**: Mesajların bulunduğu stream.
- **mygroup**: Mesajların işlendiği tüketici grubu.
- **consumer2**: Yeni tüketici (consumer2) mesajları devralacak.
- **60000**: İşlenmeyen mesajların devredilmesi için minimum bekleme süresi (60 saniye).
- **0**: Başlangıç ID'si. "0", stream'deki ilk mesajdan başlanacağını belirtir.

Bu komut, 60 saniye boyunca işlenmemiş tüm mesajları "consumer2" adlı yeni tüketiciye devredecektir.

Örnek 2: COUNT Parametresi ile Mesaj Sayısını Sınırlama

Eğer bir kerede devredilecek mesaj sayısını sınırlamak istiyorsanız, `COUNT` parametresini kullanabilirsiniz. Örneğin, son 100 mesajdan yalnızca 10 tanesini devretmek isterseniz:

```
XAUTOCLAIM mystream mygroup consumer2 60000 0 COUNT 10
```

Bu komut, 60 saniye boyunca işlenmeyen 10 mesajı "consumer2" adlı tüketiciye devreder.

Örnek 3: JUSTID Seçeneği ile Yalnızca ID Döndürme

Eğer sadece mesaj ID'lerini almak isterseniz, ancak mesajların içeriğini almanıza gerek yoksa `JUSTID` seçeneğini kullanabilirsiniz:

```
XAUTOCLAIM mystream mygroup consumer2 60000 0 COUNT 5 JUSTID
```

Bu komut, işlenmemiş 5 mesajın yalnızca ID'lerini döndürecektir. Mesaj içeriği geri döndürülmeyecektir.

XAUTOCLAIM'in Faydaları:

- **Verimli mesaj devri:** XAUTOCLAIM, zaman aşımına uğramış mesajları belirli bir tüketiciye devretmek için daha verimli bir yol sağlar.
- **Kolay otomasyon:** Tüketicinin mesajları işlemeyi başaramadığı durumlar için otomatik mesaj devri sağlayarak manuel müdahale gerekliliğini ortadan kaldırır.
- **Mesaj kaybı önleme:** Eğer bir tüketici zamanında bir mesajı işlemezse, o mesaj kaybolmaz. Başka bir tüketici devralır ve işlemi tamamlar.

Önemli Notlar:

- **XAUTOCLAIM** komutunun etkin çalışabilmesi için tüketici grubunun doğru şekilde yapılandırılmış olması gerekir.
- Bu komut, yalnızca **consumer group** yapısında ve **stream** üzerinde çalışır.
- **min-idle-time** parametresi, mesajın en son hangi zaman diliminde işlendiğini baz alır, bu nedenle zaman aşımı durumlarını yönetmek için dikkatlice ayarlanmalıdır.

Sonuç:

XAUTOCLAIM, Redis Streams ile çalışan sistemlerde, işlenmeyen mesajları başka bir tüketiciye devretmek için çok kullanışlı bir araçtır. Bu komut, sistemin verimli çalışmasını ve mesajların kaybolmamasını sağlar.

Kullanım Senaryoları:

Tüketici Grubunun Okuma Konumunu Sıfırlama:

Eğer bir tüketici grubunun tüm mesajları yeniden işlemesini istiyorsanız, bu komutu kullanarak başlangıç ID'sini '**0**' olarak ayarlayabilirsiniz.

Örnek:

```
XGROUP SETID mystream mygroup 0
```

Bu komut, "mygroup" adlı tüketici grubunun, "mystream" adlı stream'deki tüm mesajları yeniden okumasını sağlar.

Okuma Konumunu Belirli Bir Mesaj ID'sine Ayarlama:

Eğer belirli bir mesajdan itibaren okumaya başlamak istiyorsanız, o mesajın ID'sini kullanarak okuma konumunu ayarlayabilirsiniz.

Örnek:

```
XGROUP SETID mystream mygroup 1623844394082-0
```

Bu komut, "mygroup" adlı tüketici grubunun, "mystream" adlı stream'deki `1623844394082-0` ID'sinden itibaren okumaya başlamasını sağlar.

Tüketici Grubunu Daha Sonra Gelen Mesajlarla Başlatma:

Eğer bir gruptaki tüketicilerin yalnızca yeni gelen mesajları almasını istiyorsanız, ID parametresini `$` olarak belirtebilirsiniz. Bu, grubu yalnızca **sonraki gelen mesajları** okumaya başlatır.

Örnek:

```
XGROUP SETID mystream mygroup $
```

Bu komut, "mygroup" adlı tüketici grubunun, "mystream" stream'inde yalnızca **sonraki gelen mesajlardan** başlamasını sağlar. Yani, bu komut, mevcut mesajları atlar ve grubu yalnızca **yeni mesajları** almaya yönlendirir.

Örnek Senaryo:

Bir sistemde, bir grup tüketici (consumer group) mesajları alıyor ve her tüketici farklı bir mesaj üzerinde çalışıyor. Fakat, bazı sebeplerle (örneğin, sistem yeniden başlatması veya bir hata) tüketici grubunun başlangıç mesajı yanlış ayarlanmış olabilir ve bir noktada geri dönmek gerekebilir.

Tüketici grubunun son okuduğu ID'yi bilmeden önceki mesajlardan itibaren okumaya başlamasını istiyorsanız:

```
XGROUP SETID mystream mygroup 0
```

Bu, tüm eski mesajları tekrar işler.
Tüketici grubunun yalnızca yeni gelen mesajları almasını istiyorsanız:

```
XGROUP SETID mystream mygroup $
```

XREVRANGE Komutunun Örnek Kullanımları:

1. Tüm Mesajları Ters Sırayla Almak

Bu komut, "mystream" adlı stream'deki tüm mesajları ters sırayla almak için kullanılabilir.

Kodu kopyala
```
XREVRANGE mystream + -
```
- **+**: En son mesaj.
- **-**: En eski mesaj.

Bu komut, "mystream" stream'inin tüm mesajlarını **en yeni mesajdan başlayarak** ters sırayla alır.

2. Son N Mesajı Almak

Örneğin, "mystream" stream'inde son 5 mesajı almak için şu komutu kullanabilirsiniz:

Kodu kopyala
```
XREVRANGE mystream + - COUNT 5
```
Bu komut, "mystream" stream'inde son 5 mesajı ters sırayla döndürecektir. Yani, son eklenen mesajdan başlayarak en eski 5 mesajı alırsınız.

3. Belirli Bir Aralıktaki Mesajları Ters Sırayla Almak

Diyelim ki "mystream" adlı stream'deki **1623844394082-0** ile **1623844400000-0** arasındaki mesajları ters sırayla almak istiyorsunuz:

Kodu kopyala
```
XREVRANGE mystream 1623844400000-0 1623844394082-0
```
Bu komut, **1623844400000-0** ID'sinden **1623844394082-0** ID'sine kadar olan mesajları **ters sırayla** döndürür. Bu da demektir ki, en yeni mesajdan başlayarak verilen aralıktaki mesajları alırsınız.

Örnek 1: Tüketici Grubunun Okuma Konumunu Sıfırlama

Diyelim ki bir stream olan `mystream` üzerinde bir **consumer group** olan `mygroup` var. Bu grup, geçmişteki mesajları işledi ancak bir nedenle yeniden başlamak istiyorsunuz. Eğer bu grubu stream'in başından itibaren tekrar okumaya başlatmak istiyorsanız, **XSETID** komutunu şu şekilde kullanabilirsiniz:

```
XSETID mystream mygroup 0
```

Bu komut şu anlamı taşır:

- **mystream**: Mesajların bulunduğu stream.
- **mygroup**: Tüketici grubunun adı.
- **0**: Başlangıç ID'si olarak "0" verilmiştir. Bu, grup okuma işlemini stream'deki tüm mesajlardan başlatacaktır.

Bu durumda, **mygroup** adlı grup tüm mesajları baştan itibaren alacaktır.

Örnek 2: Belirli Bir Mesaj ID'sinden Başlamak

Bir başka senaryoda, grubu belirli bir mesajdan itibaren okumaya başlatmak isteyebilirsiniz. Örneğin, `mystream`'deki mesaj ID'si `1623844394082-0`'dan itibaren okumaya başlamak için şu komutu kullanabilirsiniz:

```
XSETID mystream mygroup 1623844394082-0
```

Bu komut, **mygroup** adlı tüketici grubunun, **mystream** adlı stream'deki `1623844394082-0` ID'li mesajdan itibaren okumaya başlamasını sağlar. Bu, geçmişteki mesajları atlar ve yalnızca bu ID'den sonraki mesajları alır.

Örnek 3: Yalnızca Yeni Gelen Mesajları Okumak

Eğer bir grubu yalnızca yeni mesajlardan itibaren okumaya başlatmak istiyorsanız, **XSETID** komutunda **$** (dolar işareti) kullanılabilir. Bu, grubu yalnızca stream'e yeni eklenen mesajlardan okumaya yönlendirir. Örneğin:

```
XSETID mystream mygroup $
```

Bu komut, **mygroup** grubunu yalnızca **mystream**'e eklenen yeni mesajlardan itibaren okumaya başlatır. Yani, geçmişteki tüm mesajlar göz ardı edilir ve yalnızca **$** işaretinden sonra gelen yeni mesajlar okunur.

XSETID Komutunun Faydaları:

- **Esneklik:** Okuma başlangıç noktasını değiştirerek, tüketici gruplarının hangi noktadan itibaren veri almaya başlayacağını kontrol edebilirsiniz.
- **Zaman Aşımına Uğramış Mesajları Atlamak:** Eğer bir tüketici grubu belirli mesajları zamanında işlemediyse, **XSETID** komutuyla okuma konumunu değiştirebilir ve yalnızca yeni gelen mesajlara odaklanabilirsiniz.
- **Veri Yönetimi:** İhtiyaca göre grubu geçmiş mesajlardan başlatabilir ya da sadece yeni gelen mesajlarla sınırlı tutabilirsiniz.

Sonuç

XSETID komutu, bir **consumer group**'un okuma başlangıç noktasını değiştirerek, veri akışını esnek bir şekilde yönetmenizi sağlar. Bu komut sayesinde, geçmişteki mesajları yeniden işlemek veya yalnızca yeni gelen mesajları almak mümkün olur. Bu, özellikle yüksek hacimli veri işleme sistemlerinde veya hata toleransı sağlamak için yararlı olabilir.

Örnek Kullanımlar:

1. Stream Boyutunu Maksimum Mesaj Sayısına Göre Sınırlamak

Bir stream'in yalnızca son 100 mesajı tutmasını istiyorsanız, şu komutu kullanabilirsiniz:

```
XTRIM mystream MAXLEN 100
```

Bu komut, "mystream" adlı stream'deki mesaj sayısı 100'ü geçtiğinde, en eski mesajları siler ve stream'i 100 mesajla sınırlar.

2. Stream Boyutunu Maksimum Mesaj Sayısına Göre ve Limit ile Sınırlamak

Eğer yalnızca son 100 mesajı tutmak istiyorsanız ve silme işlemi sırasında bir limit belirlemek istiyorsanız, şu şekilde kullanabilirsiniz:

```
XTRIM mystream MAXLEN 100 LIMIT 100
```

Bu komut, "mystream" adlı stream'deki maksimum 100 mesajı tutar ve bu limitin üzerindeki eski mesajları siler. Ancak, silme işleminde sadece 100 mesaj silinir.

3. Stream Boyutunu Minimum ID'ye Göre Sınırlamak

Belirli bir ID'den eski mesajları silmek için **MINID** parametresini kullanabilirsiniz. Örneğin, stream'deki `1623844394082-0` ID'sinden eski tüm mesajları silmek için şu komut kullanılabilir:

```
XTRIM mystream MINID 1623844394082-0
```

Bu komut, **1623844394082-0** ID'sinden önceki tüm mesajları siler.

4. Stream Boyutunu APPEND Seçeneği ile Güncellemek

Eğer sadece yeni mesajları eklerken eski mesajları silmek istiyorsanız, **APPEND** seçeneğiyle şu şekilde bir işlem yapılabilir:

```
XTRIM mystream MAXLEN 100 APPEND
```

Bu komut, stream'e yeni mesaj ekledikçe eski mesajları siler ve stream'in boyutunu 100 ile sınırlayarak sadece en yeni 100 mesajı tutar.

Önemli Notlar:

- **XTRIM** komutu, mesajları **stream'den siler**, bu nedenle silinen mesajlar geri alınamaz. Bu komutu kullanırken dikkatli olunmalıdır.
- **MAXLEN** parametresi kullanıldığında, Redis **en eski mesajları siler**. Yani, stream'in başında yer alan eski mesajlar silinirken, en yeni mesajlar korunur.
- **LIMIT** parametresi, genellikle **MAXLEN** ile birlikte kullanılır. Bu parametre, mesaj silme işlemi sırasında sınırlı bir sayıdaki mesajın silinmesini sağlar.
- **APPEND** seçeneği, mesajların eklenmesiyle birlikte eski mesajların silinmesini sağlar. Bu, stream'in boyutunu sabit tutmaya yarar.

Özet

Redis'teki transaction komutları, birden fazla işlemi atomik bir şekilde çalıştırmak için kullanılır. Temel komutlar:

- **`MULTI`**: Transaction başlatır.
- **`EXEC`**: Transaction'ı çalıştırır.
- **`DISCARD`**: Transaction'ı iptal eder.
- **`WATCH`**: Anahtarları izler ve değerleri değişirse transaction'ı iptal eder.
- **UNWATCH**: Anahtarları izlemeyi iptal eder.

Bu özellik, veri tutarlılığını sağlamak ve birden fazla işlemi birleştirerek güvenli bir şekilde yönetmek için çok kullanışlıdır.

```
SAVE --> veritabanını diske senkrot olarak yazaar.

SHUTDOWN --> dbleri senkrot olarak diske yazar ve redis
serverı kapatır.

TIME --> serverdaki zaman bilgisini döndürür.

CLIENT ID

CLIENT LIST

CLIENT INFO

CLIENT SETNAME kenant103

CLIENT GETNAME

CLIENT PAUSE 7000 ALL --> istemciyi 7 saniye bekletir

CLIENT UNPAUSE
```

bf.add myfirstbf item1
bf.card myfirstbf
bf.exists myfirstbf item1
bf.info myfirstbf
bf.info myfirstbf capacity

BF.INSERT filter CAPACITY 10000 ITEMS hello
BF.INSERT filter ITEMS foo bar baz
BF.INSERT filter NOCREATE ITEMS foo bar

Daha önce BF.SAVECHUNK komutuyla dışarıya aktardığınız bir Bloom Filter'ı yüklemek için şu komutu kullanabilirsiniz:
BF.LOADCHUNK myfilter <binary_chunk_data> MAXITEMS 2000

BF.MADD bf item1 item2 item2
BF.MEXISTS bf item1 item2 item3

BF.RESERVE myfilter 0.01 1000
BF.RESERVE myfilter 0.001 1000 MAXITEMS 5000
BF.RESERVE myfilter 0.0001 10000

`BF.SCANDUMP` Komutu

`BF.SCANDUMP` komutu, RedisBloom modülünde **Bloom Filter**'dan verileri dışa aktarmak için kullanılır. Bu komut, bir Bloom Filter'ın içeriğini **ikili (binary) formatta** dışa aktarır ve verilerin bir dosyaya kaydedilmesi veya başka bir Redis instance'ına aktarılması gibi işlemler için kullanılır.

Dışa aktarılan bu veriler daha sonra **`BF.LOADCHUNK`** komutuyla başka bir Redis veritabanına veya başka bir instance'a yüklenebilir. Bu, Bloom Filter'ların taşınmasını ve paylaşılmasını sağlar.

Söz Dizimi:

```
bash
```
Kodu kopyala
```
BF.SCANDUMP <filter_name> <iterator>
```

- **`<filter_name>`**: Dışa aktarmak istediğiniz Bloom Filter'ın adı.
- **`<iterator>`**: Bu parametre, verinin dışa aktarılacağı iteratör numarasını belirtir. Bloom Filter verisi büyük olduğunda, dışa aktarım işleminde verinin birkaç parça halinde aktarılması gerekebilir. Bu nedenle iteratör kullanılır.

Nasıl Çalışır?

- **`BF.SCANDUMP`** komutu, Bloom Filter'ı ikili formatta dışa aktarır.
- Verinin dışa aktarılması birkaç parça halinde yapılabilir. Bu, verinin büyüklüğüne ve kullanılan Bellek'le ilgili sınırlamalara bağlıdır.
- İteratör numarası, dışa aktarımın hangi parçasının verileceğini belirler.
    - İlk iteratör `0` ile başlar.
    - Her sonraki iteratör numarası, dışa aktarılacak sonraki veri parçasını temsil eder.

Dışa aktarılan ikili veri, daha sonra **`BF.LOADCHUNK`** komutuyla başka bir Redis veritabanına yüklenebilir.

Örnek Kullanım:

1. Bir Bloom Filter'ı Dışa Aktarma

Örnek olarak, `myfilter` adında bir Bloom Filter'ı dışa aktarmak için:

```
BF.SCANDUMP myfilter 0
```

Bu komut, **`myfilter`** adlı Bloom Filter'ı dışa aktarır. `0` iteratörü ile ilk veri parçası başlatılır.

2. Bir Bloom Filter'ın İkinci Parçasını Dışa Aktarma

Eğer Bloom Filter verisi büyükse, ikinci veri parçası için bir sonraki iteratör numarasını kullanabilirsiniz:

```
BF.SCANDUMP myfilter 1
```

Bu komut, **`myfilter`** Bloom Filter'ının ikinci veri parçasını dışa aktarır.

3. Dışa Aktarılan Veriyi Bir Dosyaya Kaydetme

Dışa aktarılan veriyi bir dosyaya kaydetmek için, Redis komut satırını bir dosya ile yönlendirebilirsiniz. Örneğin, Redis CLI kullanarak şu şekilde dışa aktarabilirsiniz:

```
redis-cli BF.SCANDUMP myfilter 0 > myfilter_chunk_0.bin
```

Bu komut, **`myfilter`** adlı Bloom Filter'ın ilk parçasını **`myfilter_chunk_0.bin`** dosyasına kaydeder. Daha sonra `BF.LOADCHUNK` komutuyla bu dosyayı başka bir Redis instance'ına yükleyebilirsiniz.

BF.SCANDUMP Komutunun Çıktısı:

Komut başarıyla çalıştığında, Redis ikili (binary) veri döndürür. Bu veri, Bloom Filter'ın içeriğini içerir ve daha sonra `BF.LOADCHUNK` komutuyla kullanılabilir.

Çıktı Örneği:

```
1) "16384"
2) "AgAApXQDgE8="
```

- **`16384`**: Bu, Bloom Filter'ın dışa aktarıldığı veri parçasının boyutudur.
- **`AgAApXQDgE8=`**: Bu, dışa aktarılan verinin ikili formatta base64 kodlamasıdır. Bu veriyi daha sonra başka bir Redis veritabanına yüklemek için kullanabilirsiniz.

Dışa Aktarılan Veriyi Yüklemek (BF.LOADCHUNK)

Eğer dışa aktarılan veriyi başka bir Redis instance'ına yüklemek istiyorsanız, **`BF.LOADCHUNK`** komutunu kullanabilirsiniz. Örnek:

`BF.LOADCHUNK myfilter <binary_chunk_data>`
Burada:

- `<binary_chunk_data>`: **`BF.SCANDUMP`** komutuyla dışa aktarılan ikili veri parçasıdır.

Dikkat Edilmesi Gerekenler:

1. **Büyük Bloom Filter'lar**: Eğer Bloom Filter çok büyükse, verinin dışa aktarımı birkaç parça halinde yapılacaktır. Bu nedenle iteratör numarasını kullanarak her bir parça sırasıyla dışa aktarılabilir.

2. **Veri Formatı**: Dışa aktarılan veriler **ikili (binary) formatta** olacaktır ve base64 kodlamasıyla temsil edilir. Bu veriyi düzgün bir şekilde saklamak için kodlama çözümleme işlemi yapılabilir.

3. **Yükleme ve Transfer**: Dışa aktarılan veriler başka bir Redis instance'ına veya başka bir ortamda kullanılmak üzere taşınabilir. Bunun için `BF.LOADCHUNK` komutu kullanılır.

4. **İteratörler**: Eğer çok büyük bir filtre varsa, veriyi parçalar halinde dışa aktarmak için iteratörler kullanılır. Örneğin, `BF.SCANDUMP myfilter 0` ile ilk parça, `BF.SCANDUMP myfilter 1` ile ikinci parça vb.