

# Redis – Clase 1

Bases de Datos de Tipo KEY-VALUE (Clave-Valor)

# Key-Value (Clave-Valor)

- Una clave-valor es una **tabla hash simple**, principalmente usada cuando todos los accesos a las bases de datos se hacen por una **clave primaria**.
- En el mundo **NoSQL** son las bases de datos más simples para usar desde una aplicación.
- Las operaciones que se pueden realizar son:
  - Poner (PUT) un valor asociado a una clave
  - Obtener (GET) un valor dada una clave
  - Borrar (DEL) una clave

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

# Key-Value (Clave-Valor)

- El valor es un BLOB (Binary Large Object).
- La base de datos almacena el BLOB **sin saber lo que hay dentro** (opaco), y le da esa responsabilidad a la aplicación.
- Dado que estas bases de datos usan acceso por clave primaria, generalmente tienen **muy buenos tiempos de respuesta y escalan fácilmente**.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

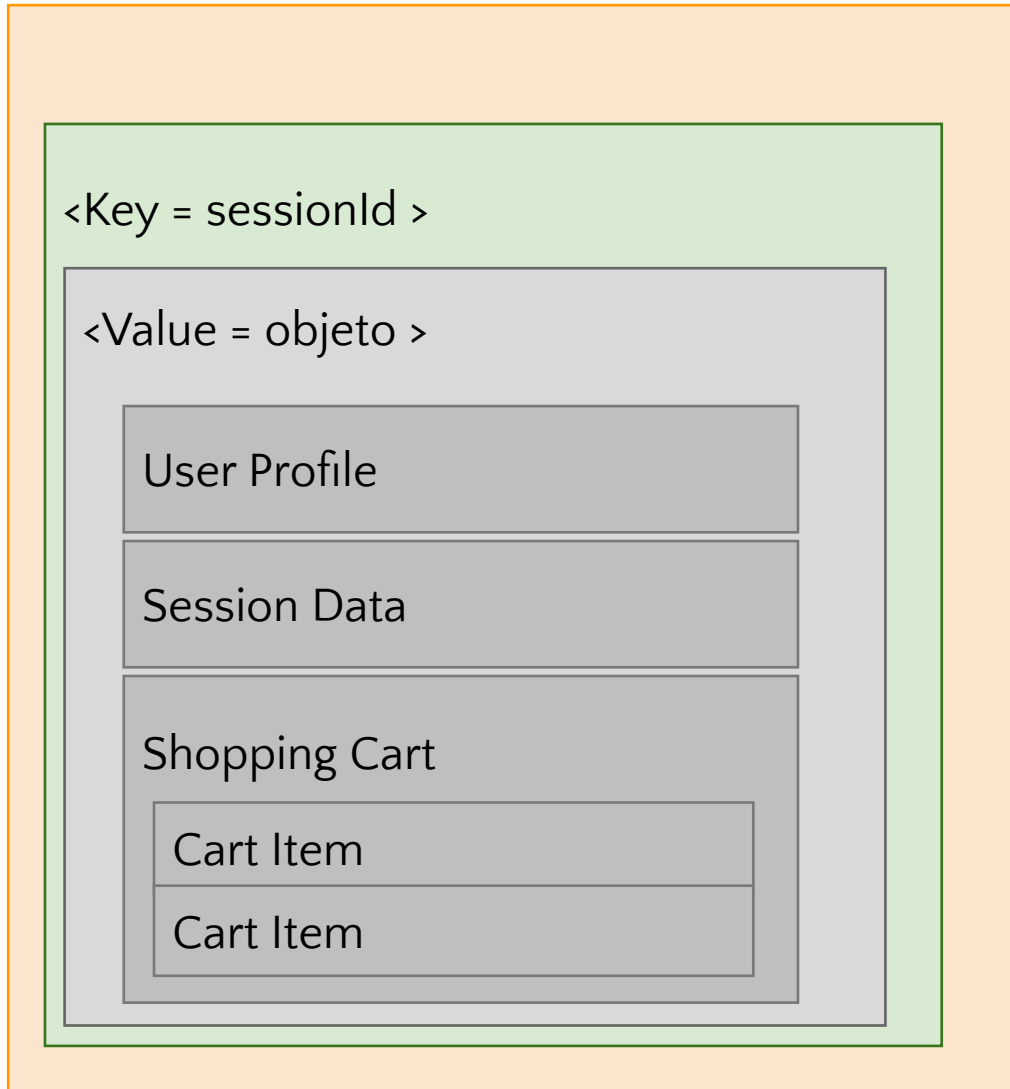
# Terminología RDBMS vs. KEY-VALUE

RDBMS	RIAK	REDIS
Database instance	Riak Cluster	Redis Instance
Schema		Redis Database
Table	Bucket	
Row	Key-Value	Key-Value
Rowid	Key	Key

# Ejemplos de BD de Tipo KEY-VALUE (Clave-Valor)

- Riak
- Redis
- Memcache DB
- Oracle NoSQL DataBase (ex Berkeley DB)
- Hamster DB
- Amazon Dynamo (not open-source)
- Project Voldemort (open source Amazon Dynamo)

# Ejemplo



- Datos del Perfil del Usuario
- Datos de la Sesión del usuario
- Preferencias del usuario.
- Información del carro de compras

# En qué casos usarlas...

## Almacenar información de sesión:

- Un buen elemento es el sessionId de la sesión web.
- Las aplicaciones que almacenan sessionId en disco o en un RDBMS pueden beneficiarse de usar una BD clave-valor.
- Todo en una sesión puede almacenarse con un solo PUT o recuperado con un solo GET.
- Toda la info de la sesión se almacena en un solo objeto.

# En qué casos usarlas...

## User Profiles – Preferences

- En general cada usuario tiene un único userId, userName, etc.
- Todos los atributos asociados al userId tales como lenguaje, paleta de colores, zona horaria, etc, pueden recuperarse en un solo acceso.
- Todos estos atributos pueden almacenarse en un único objeto.



# En qué casos usarlas...

## Carros de compras

- Los sitios de e-commerce tienen carros de compra asociados al usuario.
- Toda la información del usuario asociado al carro de compra puede almacenarse como un único objeto.

# En qué casos no usarlas...

## Relaciones entre datos

- Transacciones multi-operación.
- Consultas por los valores dentro de los valores. (a menos que la BD provea comandos específicos para consultar por los values)
- Operaciones que abarcan diferentes conjuntos de datos.



redis

# Introducción a Redis



- Base de datos **key-value avanzada**.  
*“Obtener el email del usuario con mayor puntaje en el partido que comenzó el 23 de junio a las 11pm.”*
- Es **Open Source** [Licencia BSD].
- Escrita en **ANSI C**.
- Funciona en la mayoría de los sistemas POSIX: Linux, BSD, OS X. No hay soporte oficial para Windows.
- Para lograr su alta performance, usa **datasets in memory**.
- Su *value* no es tan opaco, puede almacenar distintas estructuras de datos.

# El origen



- 2009 El proyecto Redis es comenzado por el desarrollador italiano Salvatore Sanfilippo con el objetivo de optimizar la performance de [LLOOCC](#).
- 2010 [VMWare](#) contrata a Salvatore para trabajar full-time en Redis.
- 2011 Se funda [Redis Labs](#) (proveedor comercial y SaaS)
- 2013 Redis comienza a ser patrocinado por [Pivotal](#).
- 2015 En Julio Salvatore comenzó a trabajar sobre redis en [Redis Labs](#).
- 2016 Se lanza [Redis Modules](#) para extender la funcionalidad de Redis.
- 2017 Se lanza [Redis<sup>e</sup>](#). [e: enterprise]

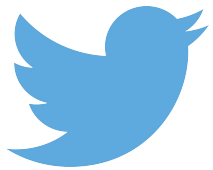




# ¿Quién lo usa?



- Utiliza Redis para cachear completamente [su capa de red](#). (usan 3 niveles de caché)



- Real-time delivery architecture
- Twitter utiliza Redis como Caché para almacenar sus Timelines
- Nuevas data-structures: [Hybrid List + BTree](#)



- Utiliza Redis para almacenar su [modelo de followers](#).



- Github utiliza Redis para [manejo de excepciones y administración de colas](#)
- También lo utilizan para administrar configuraciones y para [almacenar entre otros información de ruteo](#).



# Keys en Redis

- Las claves son **BINARY SAFE**: se puede usar cualquier secuencia binaria como key.
- Recomendaciones:
  - **No** usar keys demasiado grandes. Recordar que es una BD in memory y que una key mayor implica búsquedas más complejas.
  - **No** usar keys demasiado chicas, porque pueden resultar muy poco legibles
  - Buscar balance!  
Tratar de implementar un esquema. Por ej, usar “:” o “-” para separar multi claves (key -> “user:3100:purchases”)
- Tamaño máximo de clave: 512 MB

```
> set series:breakingbad '{"name": "Breaking Bad",  
"Seasons": 5, "quotes": ["I am the danger", "Say my  
name", "I am the one who knocks"]}'  
OK
```

```
> get series:breakingbad  
"{\"name\": \"Breaking Bad\", \"Seasons\": 5,  
\"quotes\": [\"I am the danger\", \"Say my name\", \"I am  
the one who knocks\"]}"
```

# Values: REmote DIctionary Server



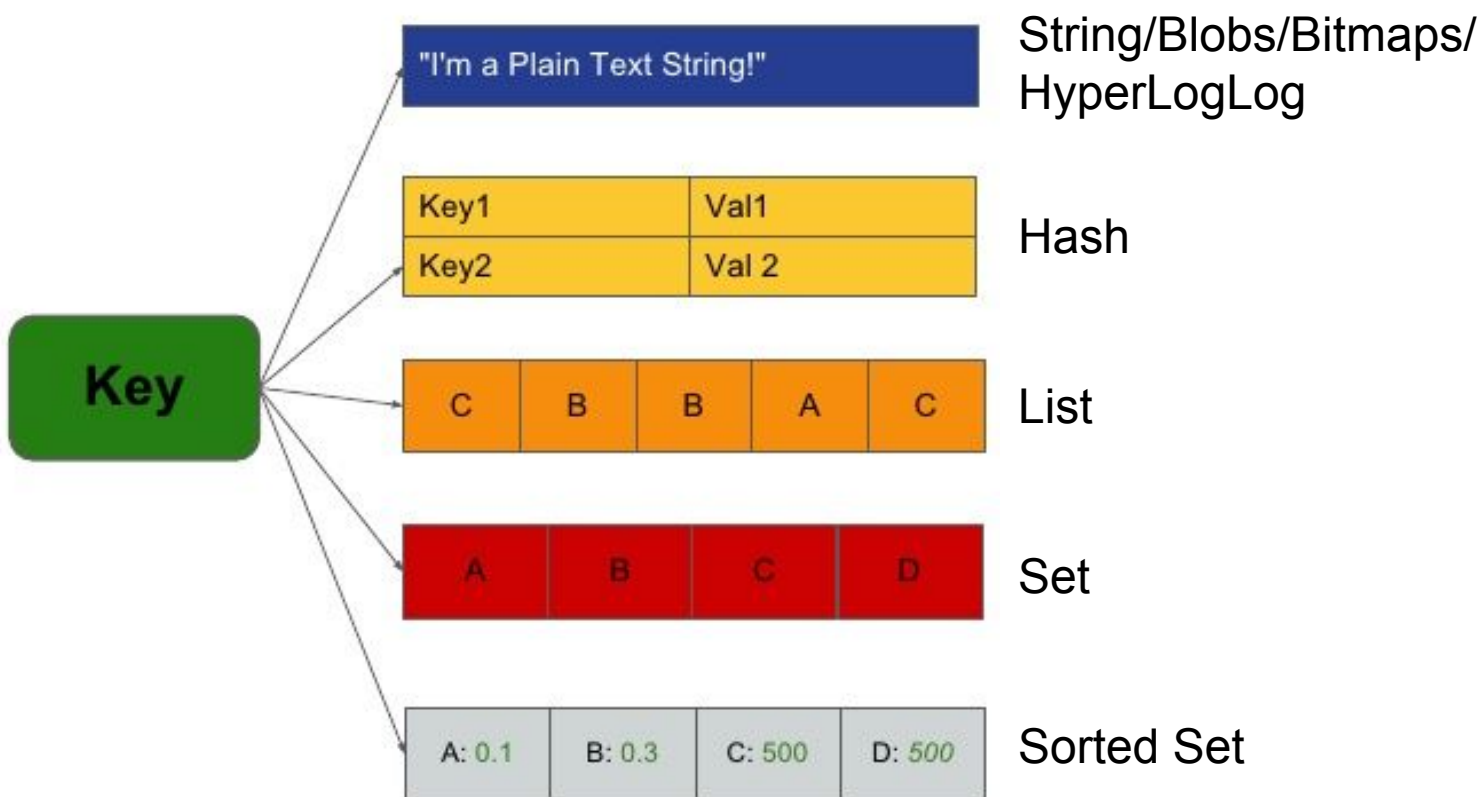
REmote Dictionary Server

- Redis es “in-memory data-structure store”.
- Usado como base de datos (clave-valor), caché, message broker.
- Se la suele llamar “**Data structure server**” porque en su *value* puede almacenar distintas estructuras de datos.
- *Values* almacenan comportamiento además de datos.





# Tipos de datos



Comentarios sobre las estructuras compuestas:

- Al agregar un elemento a una key **inexistente**, primero se **crea** un dataset vacío y luego se **agrega** el elemento.
- De igual forma, al eliminar el **último** elemento, se **elimina** la key.
- Las operaciones sobre keys **inexistentes** serán manejadas de igual forma a si los datasets estuvieran **vacíos**.

# Redis – VALUE como String

- Es el tipo de datos por defecto y más simple para un value.
- Para agregar un nuevo par key-value, basta con ejecutar el comando **SET key value**
- Para obtener un value por medio de la key, se puede utilizar el comando **GET key**
- Para eliminar un par key-value, se usa el comando **DEL key**

```
127.0.0.1:30000> set series:100 "Game of thrones"
OK
127.0.0.1:30000> set series:100 ble nx
(nil)
127.0.0.1:30000> del series:100
(integer) 1
127.0.0.1:30000> set series:100 ble nx
OK
127.0.0.1:30000> get series:100
"ble"
```

## Opciones SET

**EX seconds / PX milliseconds** -- al valor asignado se le asocia un tiempo de expiración en segundos/milisegundos.

**NX / XX** -- La operación es exitosa únicamente si la clave no existe/existe previamente.

# Redis – VALUE como COUNTER

- Existe la posibilidad de utilizar el string como un **contador**. A través de ciertos comandos redis interpretará el String como un número (si es que puede).
- Los comandos **INCR key** / **INCRBY key cantidad** permitirán incrementar por 1 o en una cantidad determinada al value asociado a una key en forma **ATÓMICA**.
  - Si dos usuarios simultáneamente modifican su valor, el mismo será consistente.
- También existe la opción de decrementar un contador con los comandos **DECR key** / **DECRBY key cantidad**.
- Si se intenta realizar alguna operación de contadores sobre un string no numérico se produce un error.

```
127.0.0.1:6379> incr likes
(integer) 1
127.0.0.1:6379> incrby likes 21
(integer) 22
127.0.0.1:30000> type likes
string
127.0.0.1:6379> set nuevo hola
OK
127.0.0.1:6379> incr nuevo
(error) ERR value is not an
integer or out of range
```

# Redis – VALUE como Hash

- Los hashes permiten dentro de un **mismo value**, mapear una nueva key con un nuevo valor. Tanto la key como el value dentro del value original deben ser de tipo String.
- Son útiles para representar “OBJETOS”.
- No hay un límite para la cantidad de elementos que se pueden guardar en un hash (salvo la memoria física!).
- Dentro de las operaciones que se pueden realizar se encuentran:
  - Agregar una nueva entrada para la key: **HSET *key subkey value***
  - Obtener todos los pares key-value guardados para una determinada key: **HGETALL *key***
  - Obtener el value de una subkey determinada: **HGET *key subkey***
  - Obtener el listado de subkeys para una key determinada: **HKEYS *key***
  - El value del par subkey-value puede ser tratado como un **Counter**, y por ende, aplicar las operaciones correspondientes.

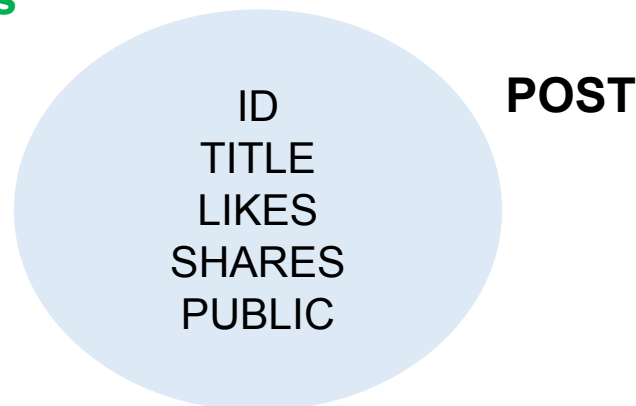
# Redis – VALUE como Hash

Modelar: Información asociada a publicaciones en una red social. Cada publicación tiene: un ID, un título, likes, shares y puede ser pública o privada.

## Modelado BD Relacional

Posts				
ID	Title	Likes	Shares	Public
1000	Great Scott!!	22	1	false

## Modelado BD Orientada a Objetos



# Redis – VALUE como Hash

## Modelado Key - Value (Redis Hashes)

post:1000	Title	Great Scott!!
	Likes	22
	Shares	1
	Public	0

```
> hmset post:1000 title "Great Scott!!" likes 22  
shares 1 public 0  
OK  
> hget post:1000 title  
"Great Scott"  
> hgetall post:1000  
1) "title"  
2) "Great Scott"  
3) "likes"  
4) "22"  
5) "shares"  
6) "1"  
7) "public"  
8) "0"  
> hincrby post:1000 likes 1  
OK
```

# Redis – VALUE como List

- Una lista es una **serie de valores ordenados**, que se almacenan como value para una key determinada.
- Implementadas como **“Linked lists”**:
  - Las operaciones de agregar elementos al principio o al final se realizan a tiempo constante, independientemente del número de elementos de la lista.
  - El acceso a elementos intermedios no es tan performante como en las listas implementadas con arrays.
- Ejemplo de cuándo usarlas:
  - Conocer las últimas actualizaciones posteadas por un usuario en una red social.
  - Comunicación entre procesos usando el modelo del consumidor-productor.

```
> lpush user:1:tweets "Hoy empiezo el gym"
(integer) 1
> lpush user:1:tweets "Challenge Accepted!"
(integer) 2
> lpush user:1:tweets "It's gonna be LEGENDARY"
(integer) 3
> lpush user:1:tweets "Winter is coming"
(integer) 4
> lrange user:1:tweets 0 1
1) "Winter is coming"
2) "It's gonna be LEGENDARY"
> rpop tweets
"Hoy empiezo el gym"
```

# Redis – VALUE como List

Operaciones sobre listas	Descripción
<b>LPUSH / RPUSH</b> <i>key elemento.</i>	Agregar elementos a la lista al inicio (LPUSH) o al final (RPUSH) de la misma.
<b>LPOP / RPOP</b> <i>key</i>	Eliminar y devolver el último (RPOP) o primer (LPOP) elemento de la lista.
<b>LLEN</b> <i>key</i>	Devuelve la cantidad de elementos de la lista asociada a una key.
<b>LRANGE</b> <i>key primerÍndice ultimoÍndice</i>	Devuelve subconjuntos de una lista indicando el primer y último índice a considerar. Poner como último valor -1 indica que se deben retornar los elementos hasta el último índice.
<b>LTRIM</b> <i>key primerÍndice últimoÍndice</i>	Limita a una lista a un intervalo de elementos.
<b>BRPOP / BLPOP</b> <i>key [key ...] timeout</i>	Versión bloqueante de RPOP/LPOP.



# Redis – VALUE como List – Ops Bloqueantes

```
public class JobProducer{
    public void notifyJob(String job){
        jedis.rpush("jobs:priority:high", job);
    }
}
```

```
public class JobConsumtr
{
    public static void main( String[] args )
    {
        Jedis jedis = new Jedis("localhost");
        List<String> jobs= null;

        while(true){

            jobs = jedis.blpop(0,"jobs:priority:high", "jobs:priority:medium", "jobs:priority:low");
            process(job);
        }
    }
}
```

- BLPOP /BRPOP pueden ser utilizados para dar soporte a manejo de colas en un esquema **Productor - Consumidor**.
  - Evita la necesidad de polling.
- El timeout especifica el tiempo máximo que esperará el cliente para obtener un valor. Luego de pasado dicho tiempo obtendrá un nil.
- Si el valor del timeout indicado es 0 (cero) el cliente esperará indefinidamente.
- Si alguna de las listas indicadas posee elementos el cliente recibirá un elemento y no se bloqueará.
- Si varios clientes se bloquean ante una key, al agregar un elemento a su lista, se desbloqueará el primer cliente que se bloqueó ante la misma.

# Redis – VALUE como Set

- Es similar a una lista, pero se diferencia en que los elementos **NO** están ordenados y **NO** pueden repetirse.
- Algunas de las operaciones que se pueden hacer son:
  - Agregar al set un elemento: **SADD *key value***
  - Eliminar del set un elemento **SREM *key value***
  - Ver si un value está dentro del set. Devuelve 1 si pertenece, 0 si no : **SISMEMBER *key value***
  - Conocer los elementos del set para una key determinada. Este comando devuelve todos los elementos que componen el value y lo hace en cualquier orden: **SMEMBERS *key***
  - Obtener un miembro cualquiera en forma random: **SRANDMEMBER *key***
  - Obtener la intersección o unión entre distintos sets: **SINTER *key [key ...]*** – **SUNION *key1 key2***
  - Obtener la cantidad de elementos de un set: **SCARD *key***

**Ejemplo:** Un sistema posee distintos artículos que tratan de diferentes temas, especificados con tags. Se requiere:

**Caso 1:** Obtener los temas en común entre distintos artículos.

**Caso 2:** Obtener todos los artículos que poseen uno o más tags



# Redis – VALUE como Set

- Los sets son muy útiles pero, al ser desordenados, no funcionan bien para algunas situaciones.
- Por este motivo surgen los **SORTED SETS**: es similar a un set a secas, pero con la diferencia que cada value tiene un **score** (valor numérico) asociado.
- Los elementos son ordenados según las siguientes reglas:
  - Si A y B son dos elementos con un score diferente,  $A > B$  si  $A.Score > B.Score$
  - Si A y B tienen igual score, se ordenan lexicográficamente. Recordar que no pueden ser iguales porque los elementos NO se repiten!!
- Los valores se **almacenan** ordenados.
- Ejemplo de aplicación: juegos que guardan el ranking de usuarios y su puntaje.
- En Redis 5 se agregaron las operaciones **ZPOPMAX** y **ZPOPMIN**.

```
> zadd peliculas 7.8 "The theory of everything" 5  
"Rocky V" 8.1 "The Imitation Game" 4.2 "The Next  
Karate Kid"  
integer (4)  
> zrange peliculas 0 -1 -- orden creciente  
1) "The Next Karate Kid"  
2) "Rocky V"  
3) "The theory of everything"  
4) "The Imitation Game"  
> zrevrange peliculas 0 1 withscores  
1) "The Imitation Game"  
2) "8.1"  
3) "The theory of everything"  
4) "7.8"  
> zrevrank peliculas "Rocky V"  
2  
> zrangebyscore peliculas 8 10  
1) "The Imitation Game"
```

# Redis – VALUE como Bitmap

No son realmente una estructura de datos, sino una nueva forma de interpretar a los **Strings** gracias a un conjunto de operaciones.

Como los Strings son “Binary Safe” y su longitud puede ser de hasta 512 MB, los bitmaps se pueden aplicar en hasta  $2^{32}$  bits diferentes.

Las operaciones se dividen en dos grupos:

***Operaciones de un único bit.*** son de tiempo constante. Por ejemplo, setear un bit u obtenerlo.

***Operaciones de grupos de bits.*** Por ejemplo, contar la cantidad de bits de un rango.

Ventajas:

Permiten ahorro de espacio en el almacenamiento de información.

Pueden ser divididos en múltiples claves.

Ejemplos de uso:

Real time analytics.

Almacenamiento de información booleana relacionada con un object id.



# Redis – VALUE como Bitmap (Cont)

## Operaciones de grupos de bits

(operaciones lógicas AND / OR / XOR / NOT)

**BITOP *operacion keyDestino key1 key2***

```
> BITOP XOR discorecuperado discol  
discoparidad
```

```
> GET discorecuperado
```

```
"\x00\x01\x01\x01\x00\x00\x00\x01\x00\  
\x00\x00\x00\x01\x01\x01\x00\x00\x00\x0  
0\x00\x01\x001010"
```

**BITCOUNT *key -- cuenta los 1s***

```
> BITCOUNT bitmap:inodos
```

**BITPOS *key bit -- primer posición con  
cierto valor (0/1)***

```
> BITPOS bitmap:inodos 0  
(integer) 2
```

## Operaciones de un sólo bit

**SETBIT *key offset value***

```
> SETBIT discol 13 0
```

```
1
```

```
> SETBIT discol 13 0
```

```
0
```

(devuelve el valor anterior)

**GETBIT *key offset***

```
> GETBIT bitmap:inodos 3
```

```
1
```

# Redis – índices Geoespaciales

**GEOADD** key longitude latitude member [longitude latitude member ...]

Los datos se almacenan como un **sorted set** de forma que luego se pueden realizar consultas por radio:

- GEORADIUS -> devuelve los elementos que se encuentran en un área (indicando el centro y el radio)
- GEORADIUSBYMEMBER -> se usa como centro, un elemento del set
- GEODIST -> devuelve la distancia entre dos coordenadas asociadas a una key

```
> GEOADD ciudades 13.361389 38.115556  
"Palermo" 15.087269 37.502669 "Catania"  
(integer) 2  
  
> GEODIST ciudades Palermo Catania  
"166274.15156960039"  
  
> GEORADIUS ciudades 15 37 100 km  
1) "Catania"  
  
> GEORADIUS ciudades 15 37 200 km  
1) "Palermo"  
2) "Catania"
```

# Redis – VALUE como HyperLogLog (HLL)

- Es muy común el problema de querer **contar la cantidad de ocurrencias de “cosas” ÚNICAS**.
  - Suele consumir mucha memoria, ya que hay que recordar elementos pasados (cantidad proporcional al número de elementos a contar).
- HyperLogLog (HLL):
  - Estructura de datos **probabilística**.
  - Útil para determinar la cardinalidad de un set.
  - Tamaño fijo y pequeño de memoria (12KB como máximo).
- Uso similar a usar un Set, pero en HLL realmente no se agrega almacena elementos.
- HLL, permiten bajar los requerimientos de memoria a cambio de menor precisión en los resultados. En el caso de HLL el error estándar es  $< 1\%$ .
- HLL es una estructura de datos que se codifica como un **String** → se pueden aplicar operaciones SET y GET.

## Ejemplo de uso

- Contar la cantidad de búsquedas distintas realizada por un usuario por día.
- Contar la cantidad de usuarios únicos que visualizaron una página.

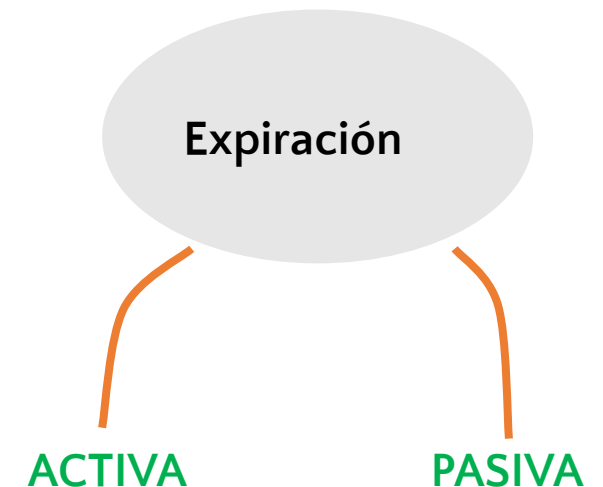
```
> pfadd views:home 122 100 100 123
(integer) 1
> pfcount views:home
(integer) 3
> pfadd views:home 122
(integer) 0
> pfcount views:home
(integer) 3
```

# Redis – Operaciones sobre Keyspace

```
> exists set
(integer) 0
> ttl set
(integer) -2
> sadd set algo
(integer) 1
> ttl set
(integer) -1
> sadd set algo
(integer) 1
> expire set 10
(integer) 1
> exists set
(integer) 1
> ttl set
(integer) 2
> exists set
(integer) 0
```

Todos los comandos mencionados en esta sección son **INDEPENDIENTES DEL TIPO DEL VALUE!!!**

- Verificar si existe una clave con un valor: **EXISTS key**
  - Devuelve 1 si existe, 0 si no existe.
- Obtener el tipo de un value: **TYPE key**
- Determinar un TTL a una clave. Esto significa que cuando expira, es borrada la clave automáticamente “como si se hubiera realizado un DEL”.
  - Modificando el value con **SET key value PX/EX ttl**
  - Sin modificar el value **EXPIRE key segundos**
  - Se puede conocer el TTL de una clave con **TTL key**
    - Devuelve el TTL , -1 si no lo tiene seteado o -2 si la key no existe.





# Redis – SCAN

En Redis existen diversos comandos que permiten iterar incrementalmente sobre conjuntos de elementos.

Devuelven una versión actualizada de un cursor.

**Inicio iteración:**  
Cursor seteado en 0

**Fin iteración:**  
Cursor devuelto en 0

<b>SCAN</b>	Dentro de todas las keys de una DB Redis
<b>SSCAN</b>	Dentro de las keys de un set
<b>HSCAN</b>	Dentro de las subkeys de un hash
<b>ZSCAN</b>	Dentro de las keys de un Sorted Set

Ofrecen garantía limitada de los elementos obtenidos

# Redis – SCAN (Cont.)

## Ejemplo Cursor

<pre>redis &gt; scan 0 1) "17" 2) 1) "key:12"    2) "key:8"    3) "key:4"    4) "key:14"    5) "key:16"    6) "key:17"    7) "key:15"    8) "key:10"    9) "key:3"   10) "key:7"   11) "key:1"</pre>	<pre>redis &gt; scan 17 1) "0" 2) 1) "key:5"    2) "key:18"    3) "key:0"    4) "key:2"    5) "key:19"    6) "key:13"    7) "key:6"    8) "key:9"    9) "key:11"</pre>
--	--

## Ejemplo Match

```
>sadd materias:sistemas AM1 AM2

Diseño Análisis SO Discreta

(integer) 6

> sscan materias:sistemas 0 match A*

1) "0"

2) 1) "AM2"
   2) "Análisis"
   3) "AM1"
```

# Redis – Publisher / Subscriber

- Paradigma Publisher/Subscriber ->  
Un remitente envía el mensaje a un canal, no a un receptor específico.

SUBSCRIBE / UNSUBSCRIBE ->

Suscribirse/desuscribirse a un canal o canales

PSUBSCRIBE / PUNSUBSCRIBE ->

Suscribirse/desuscribirse a un patrón o patrones

PUBLISH -> Publicar un mensaje a un canal particular

PUBSUB:

- CHANNELS
- NUMSUB
- NUMPAT

```
> SUBSCRIBE canal1
Reading messages...
(press Ctrl-C to quit)
1) "subscribe"
2) "canal1"
3) (integer) 1
1) "message"
2) "canal1"
3) "Hola, mundo"
```

```
> PUBLISH canal1 "Hola, mundo"
(integer) 1
```

```
> PSUBSCRIBE
novedades:*
Reading messages...
(press Ctrl-C to quit)
1) "psubscribe"
2) "novedades:*"
3) (integer) 1
1) "pmessage"
2) "novedades:*"
3) "novedades:devops"
4) "Hoy hay deploy"
```

```
> PUBLISH novedades:devops
"Hoy hay deploy"
(integer) 1
```



# Redis – Streams

Stream: estructura de datos “append only” implementada en memoria.

```
XADD key ID field value [field value ...]
```

```
XRANGE key start end [COUNT count]  
XREVRANGE key end start [COUNT count]
```

```
XREAD [COUNT count] [BLOCK milliseconds]  
STREAMS key [key ...] id [id ...]
```

- El campo ID puede ser:
  - Generado por Redis: `XADD miStream * clave1 valor1`
  - Generado por el cliente: `XADD miStream 1526919030474-55 clave1 valor1`
- Puedo crear Capped Streams con la opción `MAXLEN`.
- Los ID tienen la forma `<millisecondsTime>-<sequenceNumber>`
- Para los campos *start* y *end* puedo usar los ID especiales “+” y “-” para indicar, respectivamente, el máximo y el mínimo ID creado en un stream.
- Ejemplos:
  - `XRANGE miStream - +`
  - `XRANGE miStream - 1518951480106-0`
  - `XREVRANGE miStream + -`
- Lee datos de uno o más streams, únicamente aquellas entradas con ID más alto que el último recibido por el cliente que lo invocó.
- Se puede utilizar el ID especial “\$” para indicar que se leerán solo datos nuevos.
- Tiene una opción bloqueante.

# Redis – Streams / Consumers Groups

- Concepto inspirado en los consumers groups de Kafka.
- Requiere mucho manejo por parte del cliente.

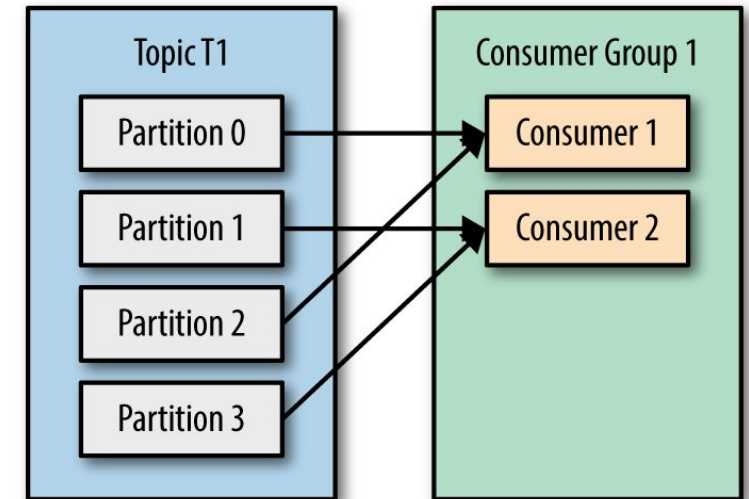
```
XREADGROUP GROUP group consumer [COUNT count] [BLOCK milliseconds]
[NOACK] STREAMS key [key ...] ID [ID ...]
```

```
XACK key group ID [ID ...]
```

- XINFO y XGROUP se utilizan para el manejo de los consumers groups

```
XPENDING key group [start end count] [consumer]
```

```
XCLAIM key group consumer min-idle-time ID [ID ...] [IDLE ms]
[TIME ms-unix-time] [RETRYCOUNT count] [FORCE] [JUSTID]
```



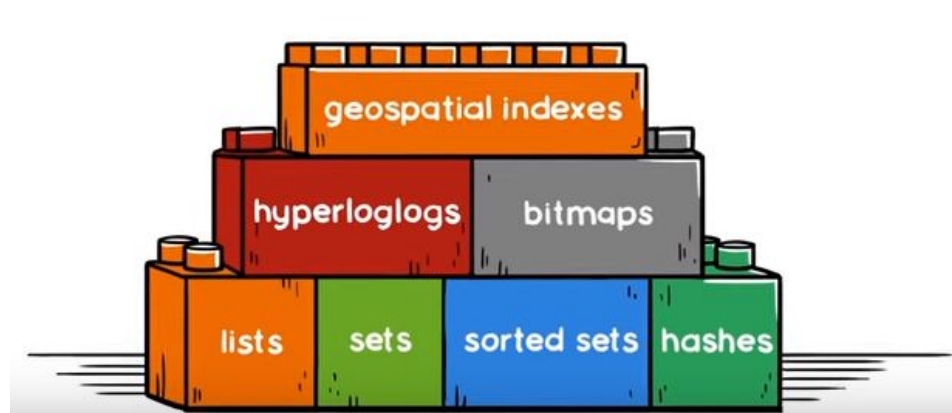
Los comandos XPENDING y XCLAIM son utilizados para resolver los casos de “dead letter”.

# Redis Building Blocks

# Redis Building Blocks

- Estructuras de datos se **almacenadas en memoria** y se **ejecutan en la base de datos**, haciéndolas extremadamente **rápidas**.
- Pensar el **modelado en base a los patrones de acceso** o consultas a la DB.
- Las estructuras de datos pueden ser **usadas como Legos**, o *building blocks*, **ahorrando tiempo y esfuerzo a los desarrolladores**.
- **Combinar estructuras de datos** para conseguir nuevas funcionalidades de forma simple:

*“Obtener el email del usuario con mayor puntaje en el partido que comenzó el 23 de junio a las 11pm.”*

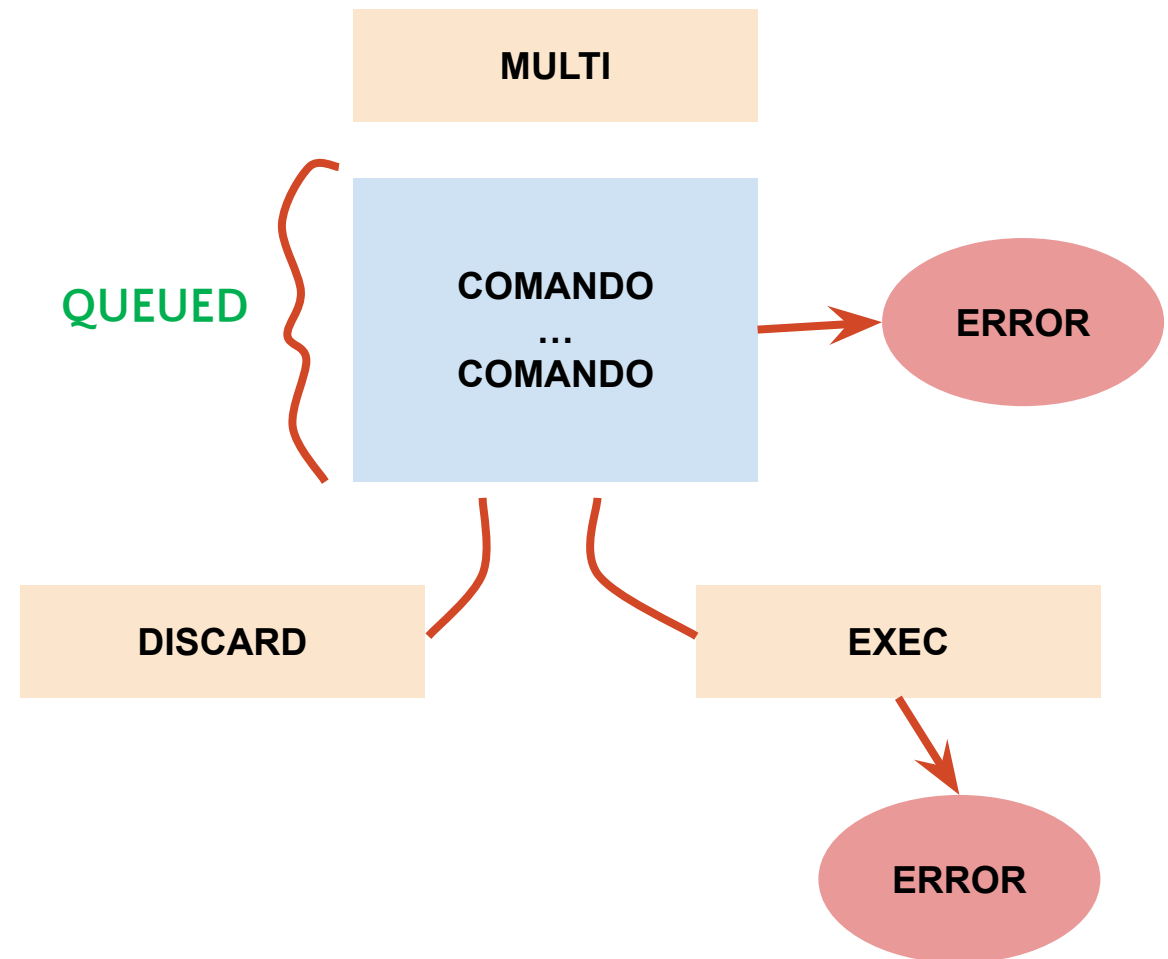


# Redis – Transacciones

Redis permite ejecutar un conjunto de comandos como un bloque garantizando:

- **Aislamiento**  
No se procesan otras operaciones durante una transacción.
- **Atomicidad**  
Se ejecutan todos los comandos o ninguno, si el desarrollador es responsable.

```
> multi
OK
> sadd post:100:likes:users "Jon Snow"
QUEUED
> incr post:100:likes:counter
QUEUED
> exec
1) (integer) 1
2) (integer) 1
```





# Redis – Transacciones & Manejo de errores

## Error ANTES de EXEC

```
> multi
OK
> incr counter
QUEUED
> inc counter
(error) ERR unknown command 'inc'
> exec
(error) EXECABORT Transaction discarded
because of previous errors.
```



## Error DESPUÉS de EXEC

```
> multi
OK
> set notalist ble
QUEUED
> lpop notalist
QUEUED
> exec
1) OK
2) (error) WRONGTYPE Operation against a key
holding the wrong kind of value
```



## REDIS NO REALIZA ROLLBACKS...

- Errores Programáticos, un tipo de error que se detecta durante el desarrollo.
- Evitar rollbacks, ganar velocidad

# Redis – Transacciones & Optimistic Locking

Si varios clientes ejecutan:

```
val = GET mykey
```

```
val = val + 1
```

```
SET mykey $val
```

Se podría llegar a diferentes resultados **erróneos** dependiendo del orden.

Check And Set (CAS) → Comando WATCH

- Se usa para "**monitorear**" keys y detectar cambios sobre las mismas.
- Si alguna de las keys con **WATCH** son modificadas por otro cliente antes de ejecutar **EXEC** toda la transacción aborta.
- Requiere el trabajo extra de reintentar la transacción si es que es abortada por esta causa.

# Scripts LUA

Ejecutar en el server redis un conjunto de acciones custom en forma **atómica**.

- Redis **scripts es una transacción por definición**: Todo lo que se hace con una transacción, se puede hacer con un script.
- Redis no necesita recompilar el script cada vez que es ejecutado.
- Para **minimizar el ancho de banda consumido** -> se puede usar EVALSHA
- Hay que tener cuidado con que los scripts no sean muy lentos -> **bloquea al server**

## INCRNX.lua

```
if redis.call("EXISTS",KEYS[1]) == 1 then
    return redis.call("INCR",KEYS[1])
else
    return nil
end
```



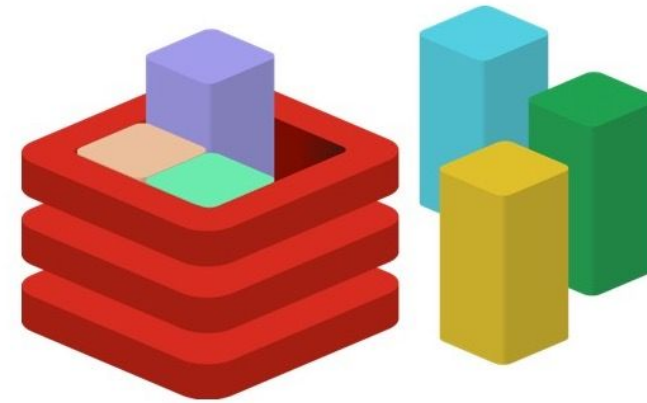
```
nosql@utn:~$ redis-cli EVAL "$(cat /path/INCRNX.lua)" 1
likes
(integer) 3
```

# Redis Modules



Redis Data Structures

**Simplicity**



Redis Modules

**Extensibility**











# Redis Modules

- Extienden la funcionalidad de Redis para atacar casos de usos comunes.
- En formato de Add-ons: Librerías dinámicas que se cargan al iniciar el motor.
- Pueden ser creados por cualquier persona.
- Utilizan el Redis Core mediante una API Alto y Bajo nivel, en C.
- Cada módulo aporta nuevas comandos al set de comandos default de redis.
- Compatibilidad binaria: Librerías no están acopladas a una version especifica del motor.



# Redis Modules

This is a list of Redis modules, for Redis v4.0 or greater, ordered by Github stars. Only modules under an OSI approved license are listed here. Also to have the source code hosted at Github is currently mandatory. To add your module here please send a pull request for [the modules.json file](#) in the [Redis-doc](#) repository.

neural-redis		Online trainable neural networks as Redis data types.	<a href="#">antirez</a>	BSD	1912 ★
RediSearch		Full-Text search over Redis	<a href="#">dvirsky</a> <a href="#">RedisLabs</a>	AGPL	583 ★
rediSQL		A redis module that provide full SQL capabilities embedding SQLite	<a href="#">siscia</a> <a href="#">RedBeardLab</a>	AGPL-3.0	409 ★
ReJSON		A JSON data type for Redis	<a href="#">itamarhaber</a> <a href="#">RedisLabs</a>	AGPL	366 ★
redis-cell		A Redis module that provides rate limiting in Redis as a single command.	<a href="#">brandur</a>	MIT	196 ★
Redis Graph		A graph database with a Cypher-based querying language	<a href="#">swilly22</a>	AGPL	170 ★
Redis-ML		Machine Learning Model Server	<a href="#">shaynativ</a> <a href="#">RedisLabs</a>	AGPL	99 ★
cthulhu		Extend Redis with JavaScript modules	<a href="#">sklivvz</a>	BSD	67 ★
redis-timerseries		Time-series data structure for redis	<a href="#">danni-m</a>	AGPL	58 ★
redis-cuckoofilter		Hashing-function agnostic Cuckoo filters.	<a href="#">kristoff-it</a>	MIT	37 ★

<https://redis.io/modules>