Bases de datos

Tiempo estimado: 25min

Una vez aclarado qué es una instancia de bases de datos, cómo configurarla, arrancarla y detenerla, así como la principal herramienta con la que ejecutar consultas, redis-cli, ya tenemos los ingredientes principales para poder cocinar datos. Vamos a comenzar presentando el concepto de base de datos. Y poco a poco, iremos describiendo cada uno de los elementos o componentes que todo programador y DBA deben conocer.

Comenzamos la lección con una introducción del concepto de base de datos. A continuación, se describe varias operaciones de bases de datos: la creación, el listado, la supresión y el cambio de base de datos activa. Después enumeramos los tipos de valores y el comando que se puede utilizar para conocer el tipo del valor de una clave. Finalmente, presentamos algunos comandos independientes del valor de las claves y cómo definir pares clave-valor volátiles o temporales.

Al finalizar la lección, el estudiante sabrá:

- Qué es una base de datos.
- Cómo crear y suprimir bases de datos.
- Cómo conocer las bases de datos de una instancia.
- Qué es la base de datos activa y cómo cambiarla.
- Cuáles son los tipos nativos soportados por Redis.
- Cómo renombrar o suprimir pares clave-valor.
- Cómo conocer si existe una determinada clave.
- Qué diferencia hay entre un par clave-valor persistente y otro volátil.
- Cómo fijar el tiempo de vida de un par clave-valor.

Introducción

Una base de datos (*database*) es una colección estructurada de datos y/u objetos, organizada atendiendo a un determinado modelo de base de datos (*database model*), el cual especifica la manera en que la base de datos se estructura y manipula. Recordemos que Redis es un almacén clave-valor. Una base de datos clave-valor (*key-value database*) es un contenedor de pares clave-valor, donde cada par (*pair*) representa un objeto de datos. En Redis, estos pares tienen tres componentes:

- La clave (*key*), que actúa como identificador único del par. Nunca habrá dos o más pares con la misma clave.
- El valor (*value*), el objeto de datos almacenado. Este valor puede ser un texto, un número, una lista, un conjunto o un objeto con campos.
- Un tiempo de vida (*time to live*) que indica la fecha de caducidad del par.

Los motores clave-valor generalmente sólo permiten acceder a los datos a través de su clave. No se puede ejecutar consultas que seleccionen pares a partir de los valores como ocurre, por ejemplo, en otros sistemas SQL o NoSQL. Esto es un aspecto negativo, pero se puede resolver mediante la desnormalización y duplicidad de datos, tal como veremos más adelante en el curso.

No se recomienda utilizar claves largas, porque consumen espacio de memoria. Pero tampoco se recomienda hacerlas cortas y poco descriptivas para ahorrar unos bytes. Hay que llegar a un equilibrio entre longitud y descripción de clave. Las claves largas consumen más recursos de procesamiento

cuando se tienen que comparar en las operaciones de L/E. Otro aspecto para no hacerlas muy largas.

Propiedades de las bases de datos

Redis puede alojar varias bases de datos, al igual que otros muchos como MongoDB, PostgreSQL o SQL Server. En Redis, las bases de datos se identifican mediante un número, conocido formalmente como el identificador de base de datos (*database id*), en vez de mediante un nombre.

Debido a que no gusta mucho este aspecto, generalmente se utiliza únicamente la base de datos 0, la predeterminada.

Cambio de base de datos activa

Toda sesión a la instancia de Redis debe estar asociada a una de sus bases de datos, la cual se conoce como base de datos activa (*active database*) o base de datos actual (*current database*). En cualquier momento podemos cambiarla mediante el comando SELECT, cuya sintaxis es como sigue:

```
SELECT identificador
```

Si deseamos abrir una sesión a una base de datos que no sea la cero, con redis-cli, hay que indicar su identificador en la opción -n. Ejemplo:

```
$ redis-cli -n 2
127.0.0.1:6379[2]>
```

Creación de bases de datos

La creación de una base de datos (*database creation*) es la operación mediante la cual se crea una nueva base de datos en la instancia de Redis. A diferencia de otros motores de bases de datos, no hace falta crear una base de datos previamente a su uso. En el momento de usar el comando SELECT para saltar a ella, si la base de datos no existe, se creará. O bien en el momento de conectar a la instancia, por ejemplo, mediante redis-cli.

Listado de bases de datos

Teniendo en cuenta que no es necesario crear explícitamente una base de datos antes de su primer uso, lo máximo que podemos conseguir es conocer el número máximo de bases de datos que puede tener la instancia. El cual se indica mediante el parámetro de configuración databases. Recordemos que podemos consultar su valor mediante el comando CONFIG GET, tal como muestra el siguiente ejemplo:

```
127.0.0.1:6379> CONFIG GET databases
1) "databases"
2) "16"
127.0.0.1:6379>
```

Su valor predeterminado es 16. Muchas organizaciones utilizan una única base de datos por instancia, asignando el valor 1 a este parámetro.

Supresión de bases de datos

La supresión de una base de datos (*database drop*) es la operación mediante la cual se suprime una base de datos y, por ende, todos sus datos. En Redis, no existe ningún comando para hacer esto. Aunque sí se dispone del comando FLUSHDB para suprimir el contenido de la base de datos activa:

```
FLUSHDB
```

Tipos de datos

Los valores de los pares pueden ser de distinto tipo. Actualmente, se dispone de los siguientes:

Tipo de datos	Tipo	Descripción
Cadena (*string*)	Simple	Un texto, un número o una secuencia binaria.
Lista (*list*)	Compuesto	Un *array* de elementos de tipo cadena.

Conjunto (*set*)	Compuesto	Un conjunto desordenado de elementos de tipo cadena.
Conjunto ordenado (*sorted set*)	Compuesto	Un conjunto ordenado de elementos de tipo cadena.
Diccionario o mapa (*hash*)	Compuesto	Un objeto de campos de tipo cadena.

Los tipos se pueden clasificar, en primera instancia, en simples y compuestos. Un tipo de datos simple (*simple data type*) es aquel que contiene un único valor. En cambio, un tipo de datos compuesto (*composite data type*), varios valores.

Redis dispone de un único tipo de datos simple, la cadena, con el que puede almacenarse texto, números y binarios como, por ejemplo, imágenes, vídeos o audio. Mientras que dispone de varios tipos de datos compuestos como son la lista, el conjunto, el conjunto ordenado y el diccionario o mapa. Este último sería como el concepto de objeto de JavaScript. En las próximas lecciones, veremos cada tipo de datos detalladamente, junto con los comandos que podemos utilizar con ellos.

Adicionalmente, podemos extender los comandos y tipos soportados mediante módulos. Así, por ejemplo, el módulo ReJSON proporciona soporte para valores de tipo JSON.

Comando TYPE

Para conocer el tipo del valor de una clave, se puede utilizar el comando TYPE:

```
TYPE clave
```

Los valores devueltos por este comando son los siguientes:

Tipo de datos	Valor devuelto
Cadena	string
Lista	list
Conjunto	set
Conjunto ordenado	zset
Diccionario	hash

Pares clave-valor

Atendiendo al objeto de datos almacenado en el valor, hay que utilizar un comando u otro, tal como veremos a lo largo del curso. Esto tiene un pequeño inconveniente, el número de comandos disponibles es enorme. Pero a medida que nos vayamos sintiendo a gusto y Redis se encuentre en nuestra área de confort, no será ningún problema.

Por ejemplo, para crear un par con un valor de tipo cadena, hay que utilizar SET, mientras que para crearlo de tipo diccionario, HSET. Pero hay un conjunto de comandos comunes, independientemente del tipo de valor almacenado.

Existencia de pares clave-valor

Para conocer si existe un par con una determinada clave, se utiliza el comando EXISTS:

```
EXISTS clave
```

El comando devuelve 1 si existe y 0 si no.

He aquí un ejemplo ilustrativo:

```
127.0.0.1:6379> EXISTS user:1:username
(integer) 1
127.0.0.1:6379> EXISTS user:1
(integer) 0
127.0.0.1:6379>
```

Supresión de pares clave-valor

Para eliminar un par clave-valor, se utiliza el comando DEL:

```
DEL clave
DEL clave clave clave...
```
El comando devuelve el número de claves suprimidas. Por ejemplo, se puede indicar tres claves, pero si una de ellas no existe, el comando devolverá dos.

Ejemplo:

```
127.0.0.1:6379> DEL user:1
(integer) 0
127.0.0.1:6379> DEL user:1:username
(integer) 1
127.0.0.1:6379>
```

Renombramiento de claves

Para renombrar una clave, se usa los comandos RENAME y RENAMENX:

```
RENAME actual nuevo
RENAMENX actual nuevo
```
Hay una diferencia entre ambos comandos. Su comportamiento varía atendiendo a si existe ya una clave con el nuevo nombre. RENAMENX no renombra si ya existe una clave con ese nombre. En cambio, RENAME suprime la clave existente y después renombra. Si no existe ninguna clave con el nuevo nombre, ambos comandos hacen lo mismo, renombrar.

Veamos sus funcionamientos mediante unos ejemplos ilustrativos. Primero, vamos a analizar RENAMENX:

```
127.0.0.1:6379> EXISTS uno
(integer) 1
127.0.0.1:6379> EXISTS dos
(integer) 1
127.0.0.1:6379> EXISTS tres
(integer) 0
127.0.0.1:6379> RENAMENX uno dos
(integer) 0
127.0.0.1:6379> RENAMENX uno tres
(integer) 1
127.0.0.1:6379>
```
El primer intento consiste en renombrar uno a dos. No lo lleva a cabo porque ya existe la clave dos. El segundo intento sí se realiza, porque no existe ninguna clave con el nombre tres. Observemos los valores devueltos. Cuando no puede renombrar, devuelve 0; si puede, 1.

Ahora, veamos RENAME:

```
127.0.0.1:6379> GET uno
"111"
127.0.0.1:6379> GET dos
"222"
127.0.0.1:6379> RENAME uno dos
OK
127.0.0.1:6379> EXISTS uno
(integer) 0
127.0.0.1:6379> GET dos
"111"
127.0.0.1:6379>
```
Como se puede observar, cuando ya existe una clave con el nuevo nombre, lo que hace RENAME es suprimirla primero y, a continuación, renombrar como se le indica. Si no existe ninguna clave con el nuevo nombre, simplemente renombrará.

¿Qué ocurre si no existe ningún par con la clave a renombrar? Redis propagará un error:

```
127.0.0.1:6379> RENAME noexiste abc
(error) ERR no such key
127.0.0.1:6379> RENAMENX noexiste abc
(error) ERR no such key
127.0.0.1:6379>
```

Listado de claves

En muchas ocasiones, es muy útil obtener un listado de claves atendiendo a un determinado patrón de nombramiento. Para este fin, existe el comando KEYS:


```
KEYS patrón
```

El comando devuelve una lista con las claves cuyo nombre cumple el patrón indicado. El patrón (*pattern*) indica el modelo o formato que debe tener la clave para aparecer en el resultado. Un patrón puede contener cualquier carácter, así como comodines (*wildcard*), caracteres con un significado especial en la operación de comparación:

Comodín	Descripción
?	Representa un carácter cualquiera.
*	Representa cero, uno o más caracteres cualquiera.
[ab]	Representa un carácter de los indicados.
[^ab]	Representa cualquier carácter que no sea uno de los indicados.
[a-z]	Representa cualquier carácter entre los indicados.
\	Escapa un carácter especial.

A continuación, unos ejemplos ilustrativos:

```
127.0.0.1:6379> KEYS *
1) "dos"
2) "uno"
3) "tres"
127.0.0.1:6379> KEYS *s
1) "dos"
2) "tres"
127.0.0.1:6379> KEYS *o*
1) "dos"
2) "uno"
127.0.0.1:6379>
```

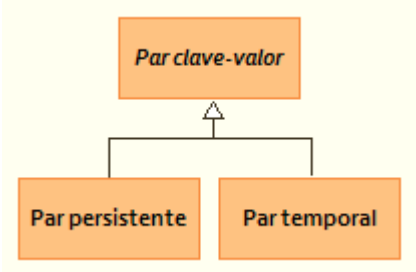
Acceso a los valores

Si para crear un par clave-valor se utiliza un comando u otro atendiendo al tipo de datos del valor, para obtener su valor ocurre igual. Así, por ejemplo, para obtener el valor de un par cuyo valor es de tipo cadena, se utiliza GET. Si en cualquier momento deseamos obtener un nombre de clave aleatoria de los pares almacenados en la base de datos, podemos utilizar RANDOMKEY:

```
RANDOMKEY
```

Pares clave-valor temporales

Un par clave-valor temporal (*volatile key-value pair*) es aquel que tiene fecha de caducidad. Este tipo de pares puede ser suprimido por el usuario de manera explícita o bien por el motor de bases de datos automáticamente cuando su tiempo de vida se ha alcanzado. El tiempo de vida (*time-to-live*) indica la fecha de caducidad del par, con objeto de que Redis lo suprima cuando se haya alcanzado. Los pares que no tienen tiempo de vida, se conocen formalmente como pares clave-valor persistentes (*persistent key-value pairs*) y deben suprimirse manualmente.



Se utiliza principalmente en pares que actúan a modo de caché. Una caché (*cache*) es un componente que almacena datos para futuros accesos. Se usa cuando el coste de generar un dato es mayor que el de acceder a una copia generada previamente. En vez de generar el dato cada vez que es accedido, lo que hacemos es generarlo una vez y entonces almacenarlo en una caché. Así cuando el dato se accede de nuevo, se coge de la caché, reduciéndose los recursos que se consumen en su generación, además de ser más rápido recurrir a la copia cacheada.

Para este tipo de datos es muy útil el uso de pares clave-valor temporales. Tengamos en cuenta que en muchas ocasiones este cacheo no puede ser *ad eternum*. Si sabemos que el dato no se modificará en las próximas 24 horas, lo que hacemos es cachearlo en Redis con un tiempo de vida de 24 horas. Así, nos evitamos tener que ir a las 24 horas y suprimir la entrada nosotros mismos explícitamente. Lo hará el motor de bases de datos sin necesidad de hacerlo nosotros. Es más, si el dato no existe, es que nunca se cacheó o el valor cacheado ha vencido. Y si el dato existe, sabemos que todavía está en vigor y podemos usarlo sin miedo.

Configuración del tiempo de vida

Para especificar el tiempo de vida de un par clave-valor, se utiliza los comandos EXPIRE y EXPIREAT:

```
EXPIRE clave segundos
PEXPIRE clave milisegundos
EXPIREAT clave fecha
```

EXPIRE y PEXPIRE indican el tiempo de vida en segundos y milisegundos, respectivamente, a comenzar desde el momento de ejecutarse. Por su parte, EXPIREAT especifica el momento exacto del tiempo en el que debe expirar. La fecha se indica en formato de tiempo Unix, esto es, un valor numérico que representa la fecha y hora exactas desde el día 1 de enero de 1970. Por ejemplo, el valor 1450073294 representa el día 14 de diciembre de 2015 a las 06:08 horas y 14 segundos.

Consulta del tiempo de vida

Para conocer el tiempo de vida que le queda a un par temporal, se utiliza los comandos TTL y PTTL:

```
TTL clave
PTTL clave
```

El comando TTL devuelve el tiempo de vida que le queda al par clave-valor en segundos, mientras que PTTL lo hace en milisegundos. Ambos comandos devuelven -1 si el par es persistente; y -2 si no existe.

Persistencia

Un par clave-valor se puede convertir de temporal a persistente mediante el comando PERSIST:

```
PERSIST clave
```

El comando devuelve 1 si existe el par, es temporal y se ha convertido a persistente; y 0 cuando no existe la clave o ya es persistente.

