10

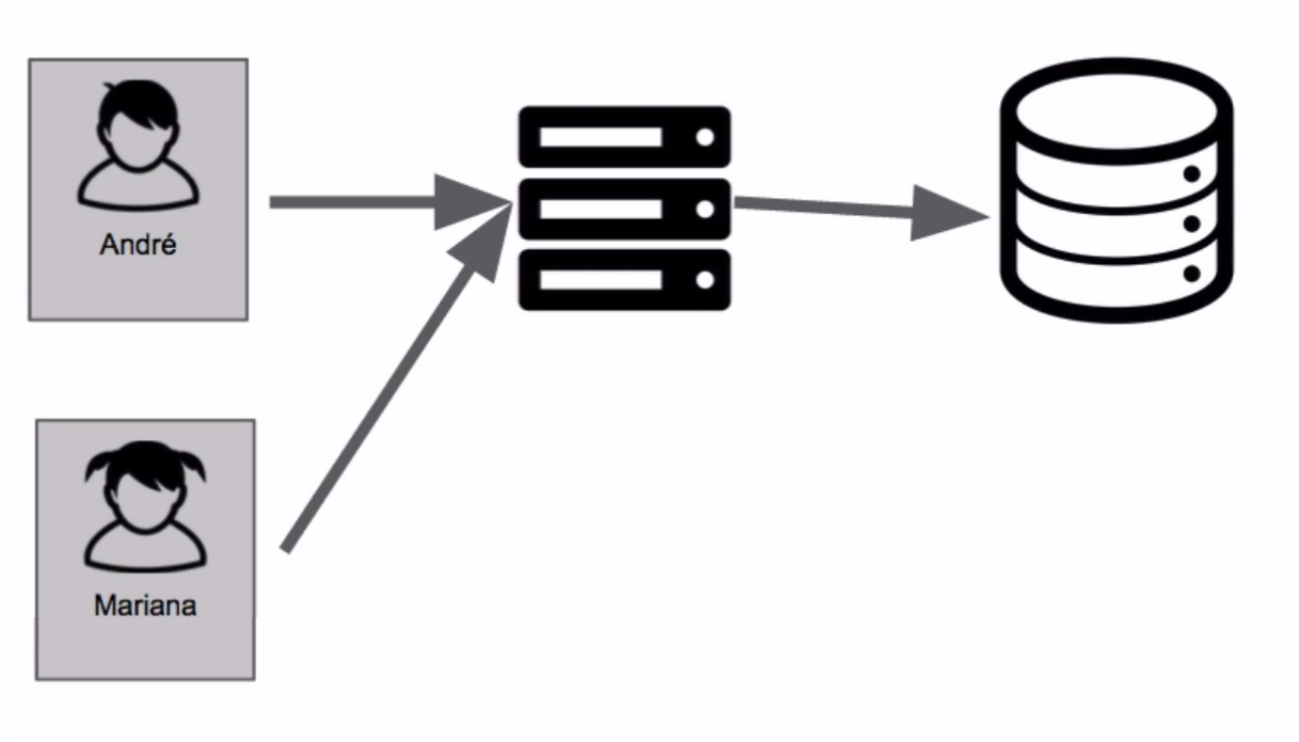
Quem acessou meu sistema

##Quem acessou meu sistema

Oi, tudo bem? Vou me apresentar: sou o usuário André e estou acessando do meu computador o [site do Alura] (http://www.alura.com.br (http://www.alura.com.br))

Vou acessar o site...

Porém, além do meu acesso, estará anotado no ***Redis*** que outras pessoas também visitaram a *Home*, o *Dashboard* e outras páginas. Por exemplo, o acesso da Mariana também estará registrado.



Da maneira como foi desenvolvido o sistema do Alura até o momento, somos capazes de dizer quantas vezes cada página foi acessada diariamente e fazer diversas *queries* ligadas a esses dados.

E se quisermos saber os dias em que o André e a Mariana acessaram o nosso sistema? Quais usuários acessaram o sistema ontem e hoje? Quero fazer perguntas ligadas ao usuário. Quero ir além da quantidade de acessos e fazer perguntas específicas, como por exemplo, quero saber se um determinado usuário acessou o sistema hoje, sim ou não? *True* ou *false*? Isto significa que quero marcar se é **verdadeiro** ou **falso**. Já vimos como trabalhar com *string*s, com números, com *float*s, ou com *hashes* (chaves e valores).

E se quisermos marcar *true* ou *false*? Se quero marcar apenas um verdadeiro ou falso, por exemplo, especificando se algo está ligado ou desligado: diremos "ligado verdadeiro" ou "ligado falso". Mas o nosso caso é mais complexo, mais divertido e maior. O que quero fazer é marcar diversos *true* e *false*, além de descobrir quais usuários acessaram o site no dia 25 de maio de 2015 (25/05/2015). Aqueles que tiverem acessado o site serão marcados com *true* ou *false*.

Qual é a menor unidade de memória que temos para marcar um verdadeiro ou falso, 0 ou 1? Será um ***bit***. Não preciso números inteiros para marcar apenas 0 ou 1. Então, utilizaremos uma estrutura otimizada no *Redis* que nos permitirá **setar** diversos *bits*, um conjunto deles - em inglês, o termo usado é *Setbit*. Quando quisermos setar um *bit* usaremos o comando *SETBIT*. E quando quisermos "pegar" um *bit* usaremos o comando *GETBIT*.

Então, imagine que temos um número grande em nosso computador, que utiliza um espaço grande de memória e podemos marcá-lo em cada *bit*: 0 ou 1, *true* ou *false*, veio ou não. Ou seja, temos um grande pedaço da memória em que podemos marcar **sim** ou **não*, independente da pergunta. No caso, fiz uma pergunta bem específica para o sistema: quais usuários acessaram o sistema no dia de hoje? Poderíamos também trabalhar com outras datas, como o dia 22 ou 15... Para responder, irei marcar com 1 cada usuário que tiver acessado o sistema - o *Redis* irá tentar otimizar esta estrutura. Se tivermos um número extremamente elevado de usuários, teremos que tomar alguns cuidados para a memória do computador não ser prejudicada. Realizaremos o processo, em seguida.

##Dizendo que algo é verdadeiro ou falso

Queremos marcar um único *bit* e especificar se algo é verdadeiro ou falso, para um valor específico - que será o **Id** do usuário. Então, irei setar o *bit* do dia 25-05-2015, que os usuários de *id* 15, 32, 46 e 11 acessaram. Eles serão marcados com 1.

```
<ip> SETBIT acesso:25-05-2015 15 1
(integer) 0
<ip> SETBIT acesso:25-05-2015 32 1
(integer) 0
<ip> SETBIT acesso:25-05-2015 46 1
(integer) 0
<ip> SETBIT acesso:25-05-2015 11 1
(integer) 0
```

Todos os outros *ids* serão marcados com 0.

Se executarmos um `GETBIT acesso:25-05-2015` para o usuário 15, ele irá responder que **sim** com o número 1.

```
<ip> GETBIT acesso:25-05-2015 15
(integer) 1
```

E o usuário 30? Ele irá informar que o usuário não acessou com o número 0.

```
<ip> GETBIT acesso:25-05-2015 3 0
(integer) 0
```

Ou seja, para a chave `acesso:25-05-2015` , o computador irá alocar uma grande quantidade de memória. A quantidade suficiente para armazenar os *bits* que ele precisa.

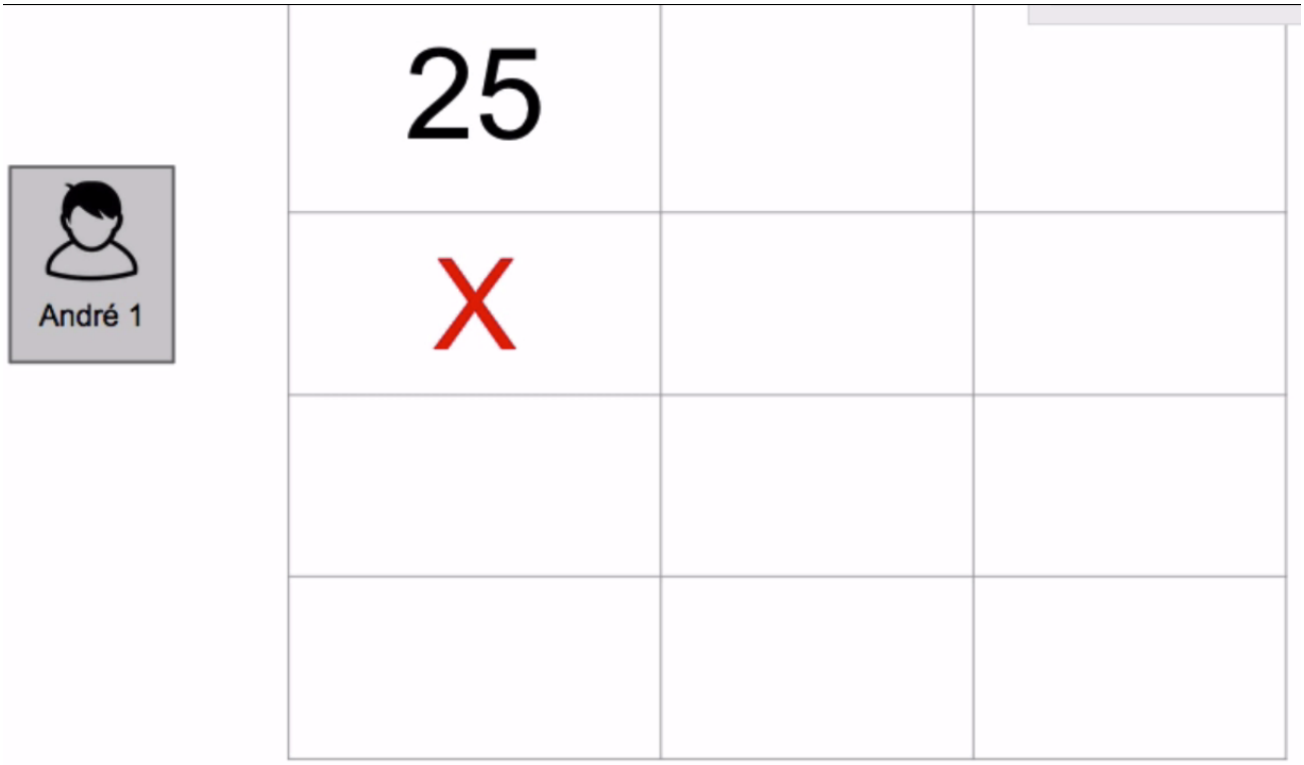
Cada um dos *bits* estará indicando para mim o *id* de um usuário. Alguns *ids* serão setados como 0 e outros como 1. Por exemplo, o *id* 15 está setado como 1, mas o 16, 17, 18, 19 estão como 0. Esta seria a maneira mais tradicional de armazenarmos estes dados. O *Redis* poderá fazer isto, ao armazenar apenas os *bits* e informando quais usuários serão 0 e quais serão 1. A sacada é que a nossa chave será tradicional (`acesso:25-05-2015)` , mas o nosso *hash* terá duas coisas especiais: o valores serão *booleanos*, o que significa que trabalharemos com 0 ou 1, verdadeiro ou falos, e assim conseguimos armazenar em um único *bit* e otimizamos a memória. E a segunda característica é que conseguimos um número pequeno e sequencial na chave. Se tivéssemos um número gigantesco, o *Redis* teria dificuldade para armazená-lo na memória. Já que trabalhamos com números pequenos e sequenciais, ele conseguirá armazenar os *bits* necessários. Então, quando além da chave, quando conseguimos transformar um *hash* em um número, por exemplo, armazenamos o *id* e não o nome de um usuário. E os valores serão 0 e 1. O que será armazenado? Ele guardeará uma sequência de *bits*, um *bitmap*. Trata-se de uma coleção de *bits* que mapeia um número determinado, como o *id* 15, para 1, ou o *id* 12 , para 0.

Revisando, com o *Setbit* iremos setar um *bit* específico para 1, enquanto o *Getbit* iremos getar um *bit* específico para 0. Até aqui, j´s podemos armazenar e extrair os dados, agora, é interessante realizarmos algumas operações com as informações levantadas. Eu quero criar um relatório a partir dos dados. Veremos isto em seguida.

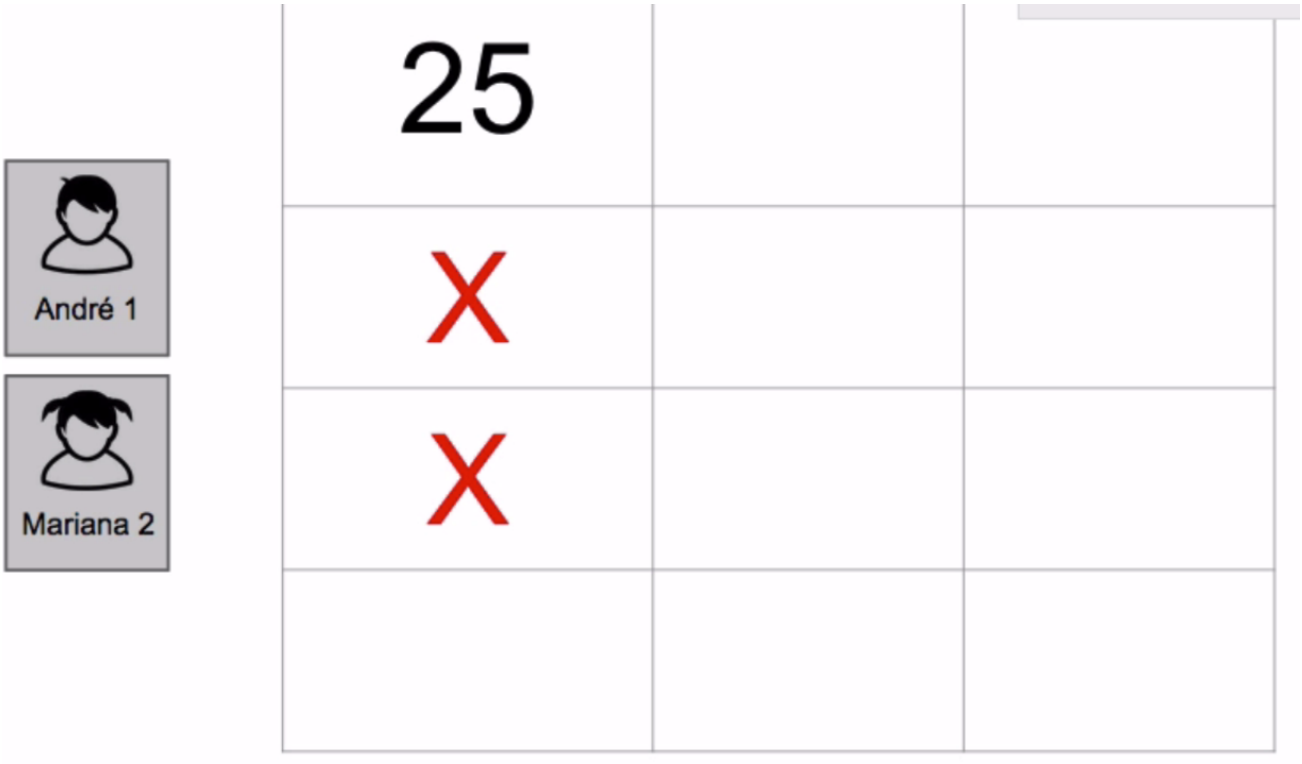
##Vamos fazer simulações

Nós vimos que somos capazes de armazenar quais usuários acessaram o sistemas e em quais dias. Faremos em seguida, um simulação de três dias em que o sistema foi acessado:

- No dia 25, o André (*id* 1) entrou no sistema. Vamos marcar o dia 25.



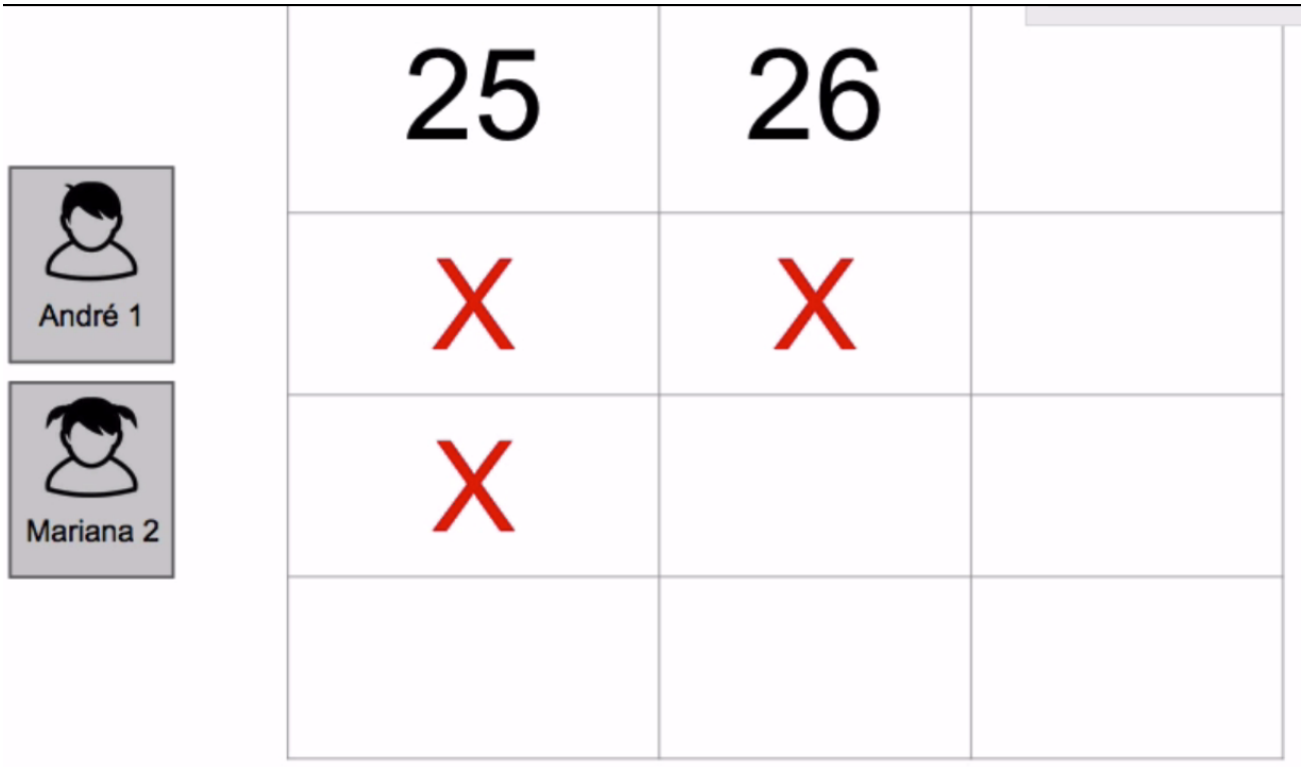
Fiz uma marcação no *Bitset* do dia 25. Na mesma data, outra pessoa acessou o sistema, a Mariana (*id* 2). Por isso, também a marcamos no dia 25.



Até aqui, especificamos `Setbit` do `acesso` realizado no dia `25-15-2015` do *id* `2` e setamos o *bit* com 1, porque a Mariana visitou o site.

```
<ip> SETBIT acesso:25-15-2015 2 1
```

No dia 26,o André também visitou o site, marcaremos com a 1 a sua posição no `Bitset` desta data.

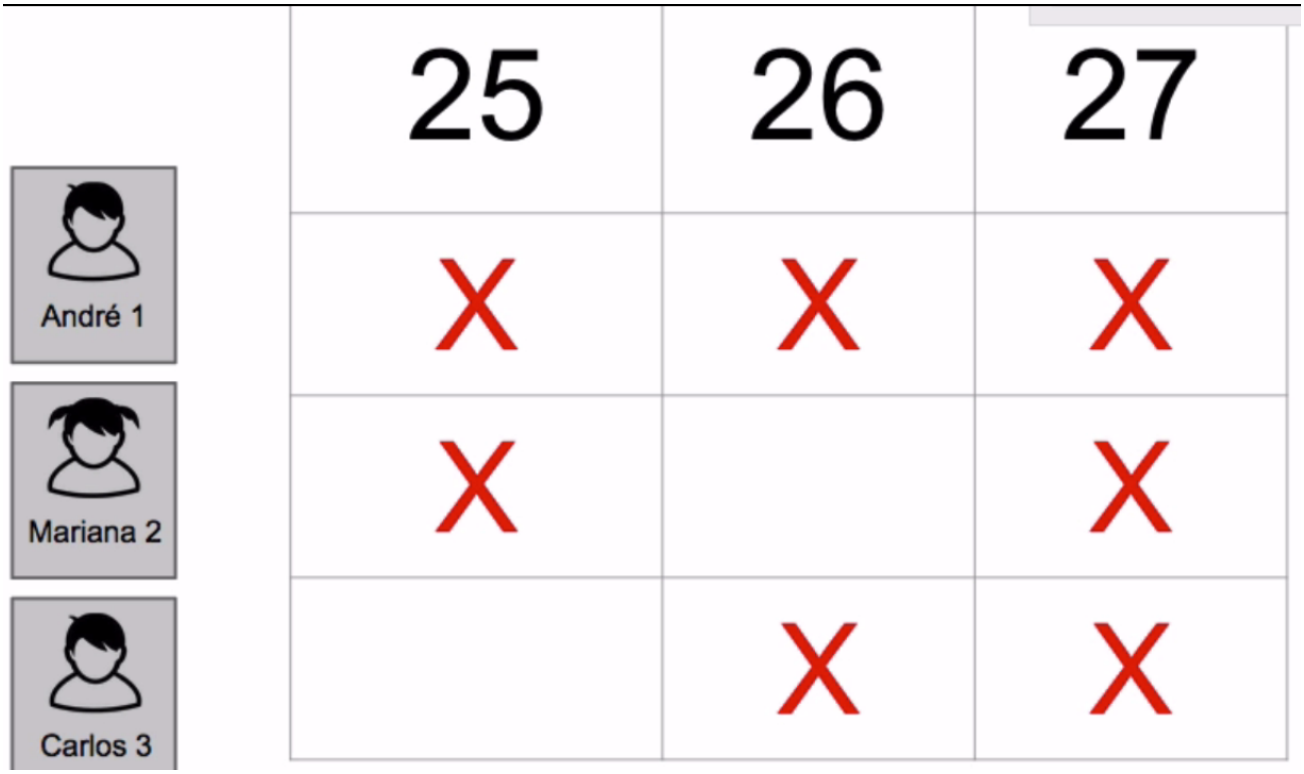


Já a Mariana não acessou o sistema neste dia. No entanto, um usuário novo entrou no sistema, o Carlos (*id* 3). Então, iremos escrever:

```
<ip> SETBIT acesso:26-15-2015 3 1
```

Nós marcaremos 1, quando for verdadeiro, e 0, quando for falso. No caso do Carlos, o setamos com 1.

Seguimos para o dia 27. Neste dia, todos acessaram o sistema.



Chamamos todos os *Setbits*: nos dias 25 e 26, foram duas vezes e no dia 27, foram três. Funcionou corretamente. Agora, queremos fazer perguntas para os meus dados... Sempre que armazenamos no *Redis*, nosso objetivo é levantar informações. Perguntas como, "o que aconteceu em determinado dia?" Uma questão importante é levantarmos tudo o que ocorreu em um *Bitmap*. Vamos usar como exemplo o dia 25, quem foi setado com 1 e com 0? Setamos com 1, todos os usuários que acessaram o sistema. Então, podemos responder a pergunta "quantas pessoas entraram no sistema, no dia 25?" Vamos contar as marcações na data. Se contarmos todos os *bits* marcados de um dia, nós saberemos quantas pessoas acessaram o sistema. O *Bitset* não só armazena as pessoas que acessaram no sistema em uma determinada data, uma série de *trues* e *falses* armazenados, também somos capazes de identificar rapidamente quantos *bits* foram marcados - ele me responderá com o *Bitcount*. Em seguida, veremos isto na prática.

##O *Bitcount*

O que faremos agora? Temos diversos mapas de *bits*, por exemplo, um que indica os acessos do dia 25 - em que ele informa se determinado usuário foi acessado ou não. Mostraremos outro exemplo, no terminal. Desta vez, usaremos o acesso do dia 25 no mês de junho. No nosso mapa, temos registrado que o André acessou o sistema. Vamos marcá-lo com 1, com isto iremos setar o *bit*.

```
<ip> SETBIT acesso:25-06-2015 1 1
```

A Mariana também acessou, vamos setar ela também.

```
<ip> SETBIT acesso:25-06-2015 1 1
(integer) 0
<ip> SETBIT acesso:25-06-2015 2 1
```

O *id* 3 não acessou, então, marcá-lo com 0 ou não incluí-lo, seria o mesmo.

```
<ip> SETBIT acesso:25-06-2015 1 1
(integer) 0
```

```
<ip> SETBIT acesso:25-06-2015 2 1
(integer) 0
<ip> SETBIT acesso:25-06-2015 3 0
(integer) 0
```

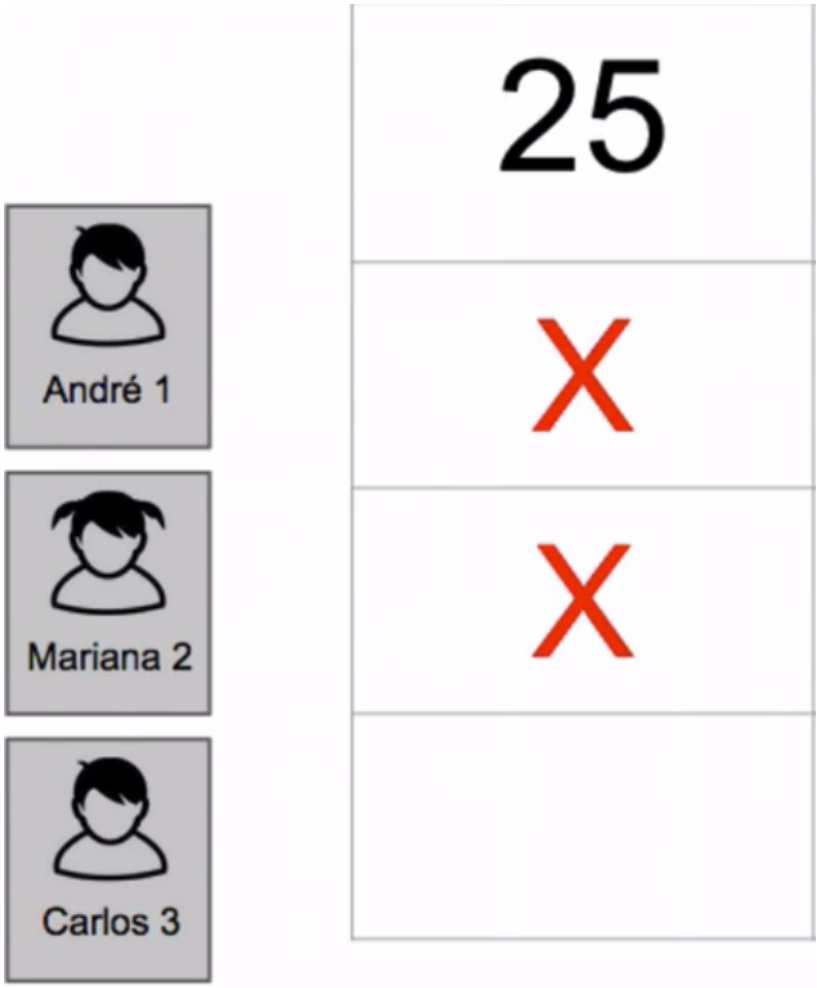
No dia 26, os acessos foram diferentes: os usuários 1 e 3 entraram no sistema. 27

```
<ip> SETBIT acesso:26-06-2015 1 1
(integer) 0
<ip> SETBIT acesso:25-06-2015 3 1
(integer) 0
```

No dia 27, os três usuários acessaram o sistema:

```
<ip> SETBIT acesso:27-06-2015 1 1
(integer) 0
<ip> SETBIT acesso:27-06-2015 2 1
(integer) 0
<ip> SETBIT acesso:27-06-2015 3 1
(integer) 0
```

Nós setamos cada *bit* que queríamos marcar. Observe que cada uma das chaves com as datas (por exemplo, `acesso:25-06-2015`) é um mapa de *bits*. Abaixo temos o mapa do dia 25:



Cada mapa de *bits* é formado pelos números 0 e 1, marcados para cada um dos *ids*. No dia 25, setamos o André com 1, a Mariana com 1 e o Carlos com 0. O mapa de *bits* da data permite calcular quantas pessoas acessaram naquele dia. Basta contar quantos *bits* diferentes foram setados. No dia 25 e 26, foram dois *bits*, e no dia 27, foram três. Se analisarmos um dia futuro, será igual a 0, porque ninguém acessou ainda.

O que iremos fazer? Nós pediremos para o *Redis* contar o número de *bits* (`Bitcount`) do acesso no dia `25-06-2015` .

```
<ip> BITCOUNT acesso:25-06-2015
```

Quantas pessoas acessaram? Foram duas.

```
<ip> BITCOUNT acesso:25-06-2015
(integer) 2
```

E no dia 26, quantos *bits* setamos para 1? Foram dois.

```
<ip> BITCOUNT acesso:26-06-2015
(integer) 2
```

Quanto *bits* setamos para 1? Foram três.

```
<ip> BITCOUNT acesso:27-06-2015
(integer) 3
```

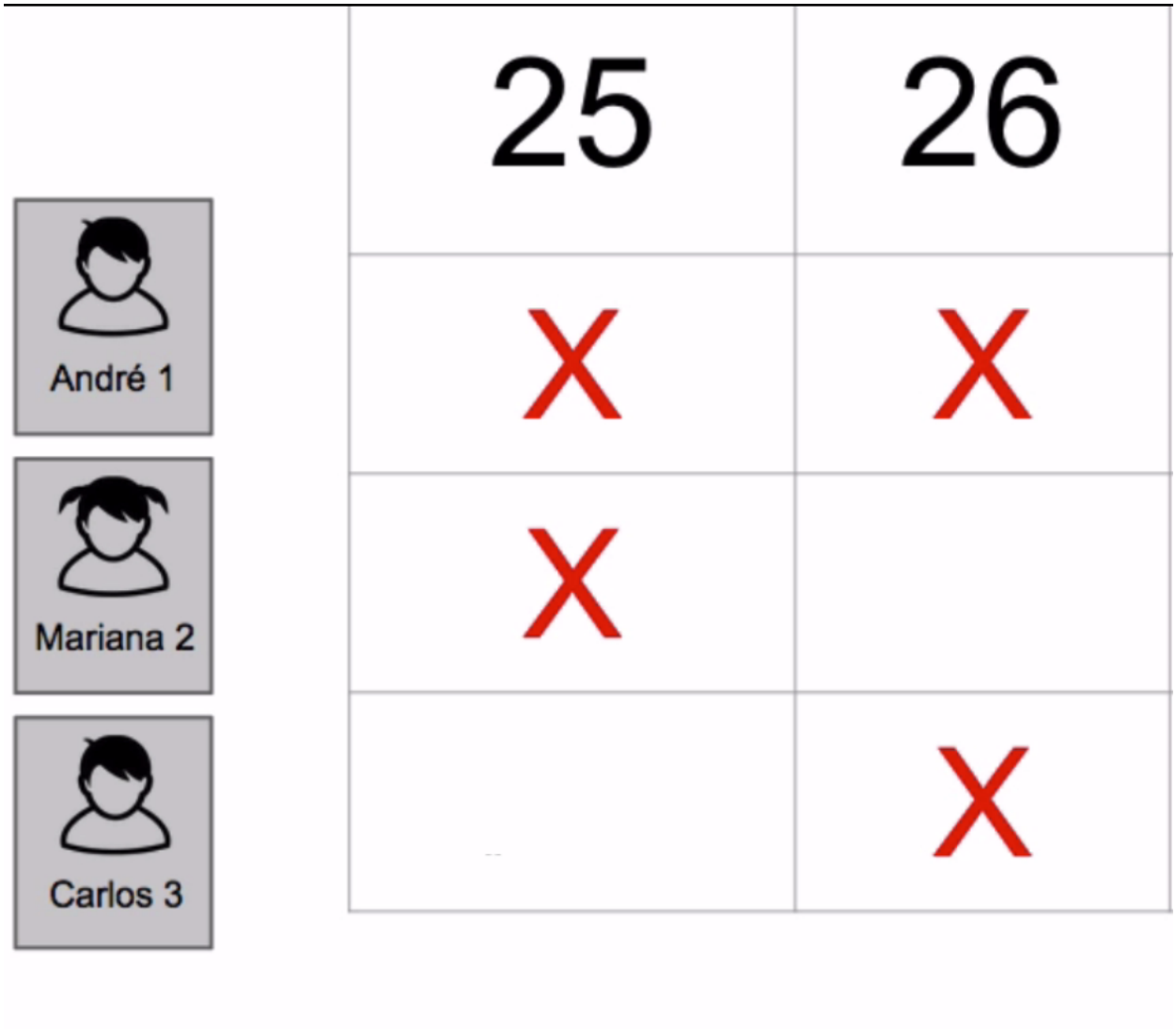
O *Bitcount* dos três dias irá aparecer assim, no terminal:

```
<ip> BITCOUNT acesso:25-06-2015
(integer) 2
<ip> BITCOUNT acesso:26-06-2015
(integer) 2
<ip> BITCOUNT acesso:27-06-2015
(integer) 3
```

O *Bitcount* utiliza os dados do papo do dia 25 que setamos e conta quantos *bits* diferentes foram marcados. Depois, ele nos informará o total. No meu cenário, o *Bitcount* serve para informar quantos usuários distintos acessaram o site em um dia específico.

Trabalhando com o *Bitcount*

Temos armazenado na memória diversas informações sobre o acesso ao meu sistema, está tudo no *Redis*. Nós conseguimos acessar os dados rapidamente, ler e criar resumos. Por exemplo, podemos somar quantos *bits* estão acesos em uma chave. Porém, às vezes, queremos fazer perguntas mais complexas, como exemplo: quantas pessoas acessaram o sistema no dia 25 e 26? Ou seja, a pessoa precisa estar marcada no dia 25 e 26:



O André acessou os dois dias, a Mariana apenas um, assim como o Carlos. Apenas o André entrou no dia 25 e** 26. Temos algo importante aqui, estamos usando o operador **e (`&`), teremos um dia e** o outro, será verdadeiro **e verdadeiro. O *bit* estará ligado nas duas datas, nas duas chaves. Então, queremos usar a primeira chave, que é uma sequência de *bits* (1-1-0), e a segunda chave (1-0-1), para identificar qual *bit* está ativo em ambas sequências.

Então, temos a primeira sequência `1-1-0` , do dia 25, e a segunda sequência `1-0-1` , do dia 26.

(25)	1	1	0
(26)	1	0	1

Agora, quais *bits* estão ativos nas duas sequências? O primeiro está ativo nas duas sequências ? Sim. O segundo está ativo nos dois? Não. O terceiro está nas duas sequências? Não. O resultado da operação de `E` entre as duas sequências será: 1-0-0.

(25)	1	1	0
(26)	1	0	1
(25 & 26)	1	0	0

Apenas o *id* 1 acessou o sistema nos dois dias. Podemos aplicar uma operação de *bits* em duas sequências e assim, gerar um sequência nova. Desta forma, podemos responder a pergunta: quantas pessoas acessaram o sistema?

O que queremos agora, é um comando no *Redis* que seja capaz de executar uma operação de `AND` com duas sequências de *bits*, gerando como resultado uma nova sequência.

##Aplicando o operador `AND`

Considerando as sequência de *bits* de quem acessou o sistema no dia 25 e 26, que aplicar o operador de **&*. Para cada *bit* (`Bitop`) queremos aplicar um operador. Queremos verificar se um *bit* foi setado em uma chave e em outra. Usaremos o operador **& - que em inglês será ***AND***. Vamos escrever isto no terminal:

```
<ip> BITOP AND
```

Podemos aplicar o operador em várias sequências, porém, quando o aplicarmos no nosso exemplo, ele terá que gerar uma sequência nova de *bits*, que receberá o nome de `acesso:25-e-26-06-2015` , em referência as duas data que trabalhamos.

```
<ip> BITOP AND acesso:25-e-26-06-2015
```

Nós esperamos que a sequência resulte m 1-0-0.

Quais são as sequências em que iremos aplicar o operador? A primeira será do dia 25 (`acesso:25-06-2015`), e a segunda será do dia 26 (`acesso:26-06-2015`). Quando aplicarmos o `AND` resultado será:

```
<ip> BITOP AND acesso:25-e-26-06-2015 acesso:25-06-2015 acesso:26-06-2015
(integer) 1
```

Sempre que o *bit* estiver definido nas duas sequências, ela deverá ser definida também na sequência de saída. Vamos testá-la? Será que o usuário 1 acessou? Sim.

```
<ip> BITOP AND acesso:25-e-26-06-2015 acesso:25-06-2015 acesso:26-06-2015
(integer) 1
<ip> GETBIT acesso:25-e-26-0602015 1
(integer) 1
```

E o usuário 2? Não. O usuário 3 também não acessou. Apenas o usuário 1 acessou os dois dias.

```
<ip> BITOP AND acesso:25-e-26-06-2015 acesso:25-06-2015 acesso:26-06-2015
(integer) 1
<ip> GETBIT acesso:25-e-26-06-2015 1
(integer) 1
<ip> GETBIT acesso:25-e-26-06-2015 2
(integer) 0
<ip> GETBIT acesso:25-e-26-06-2015 3
(integer) 0
```

Não necessariamente iremos começar com o valor 1. Podemos ter um usuário que tenha o *bit* com o valor 0. É possível existir um *id* 0. O sistema do *Redis* não está focado nos *ids* de usuários, ele está interessando nas chaves que iremos determinar.

Então, um delimitador de *bit* começa do 0 e segue adiante - não faria sentido começar uma sequência de *bits* pelo 0.

Temos a sequência de *bits* 1-0-0 e já sabemos quais são os usuários que acessaram o sistema nos dois dias. Bastou somarmos quantos *bits* estavam ativos nas duas sequências.

(25)	1	1	0
(26)	1	0	1
(25 & 26)	1	0	0

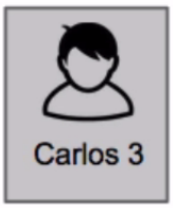
Pela sequência, vemos que apenas um usuário esteve ativo nos dois dias.

```
<ip> BITCOUNT acesso: 25-e-26-06-2015
(integer) 1
```

Primeiro fizemos um resumo de **quem** entrou no sistema nos dois dias, depois somamos para descobrir a quantidade de acessos. Inicialmente fizemos um operador `AND` para saber quem acessou os dois dias e depois, somamos para saber quantas pessoas haviam acessado. Desta forma, soubemos que apenas uma pessoa havia acessado.

##Aplicando a operação de `OU`

Conseguimos identificar quantos *bits* temos ativados, quando trabalhamos com duas sequências. Vamos responder um pergunta diferente agora: Como saber quantos usuários acessaram no fim de semana? Ou seja, quero saber quem acessou no sábado **ou** (|) no domingo, ou em ambos dias. Que pessoas acessaram pelo menos um dos dias, isto é, em um conjunto de dias. Então, não iremos mais aplicar o `AND` que exigia que os usuários tivessem acessado todos os dias.

	25	26	27
 André 1	X	X	X
 Mariana 2	X		X
 Carlos 3		X	X

Para isto, todos os *bits* do mapa precisavam estar setados, tanto em uma sequência como nas duas. Agora precisamos que o *bit* esteja ativo em apenas uma das colunas, indicando que o usuário acessou o sistema uma vez. Por exemplo, se perguntarmos quais usuários acessaram no dia 25 e 26, veremos que o André, a Mariana e o Carlos acessaram. Vamos ver como ficaria no nosso mapa?

(25)	1	1	0
(26)	1	0	1

A sequência do dia 25 é 1-1-0, o que significa que os *ids* 1 e 2 acessaram, mas o 3 não. No dia 26, os *ids* 1 e 3 acessaram, mas o 2 não.

(25)	1	1	0
(26)	1	0	1
(25 OU 26)			

Se queremos aplicar a operação de `|`, significa que a primeira sequência **ou** a segunda precisam estar ativas. Se setarmos 1 no dia 25 ou 26, também iremos marcar 1 na terceira sequência. No nosso caso, os três usuários acessaram pelo menos um dia.

(25)	1	1	0
(26)	1	0	1
(25 OU 26)	1	1	1

Para sabermos quantos usuários acessaram, basta contarmos a quantidade de *bits* ativos na terceira sequência. Assim, saberemos que **três** usuários acessaram durante o fim de semana, seja no sábado ou no domingo. O número de usuários ativos (únicos) será 3. Agora, queremos fazer esta operação no *Redis*.

##Trabalhando com o operador `OR` no *Redis*

Queremos saber quem acessou o sistema nos dias 25 ou 26.

```
<ip> BITOP AND acesso:25-e-26-06-2015 acesso:25-06-2015 acesso:26-06-2015
(integer) 1
<ip> GETBIT acesso:25-e-26-06-2015 1
(integer) 1
<ip> GETBIT acesso:25-e-26-06-2015 2
(integer) 0
<ip> GETBIT acesso:25-e-26-06-2015 3
(integer) 0
```

Se formos no terminal, aplicaremos um operador de *bits*, porém, desta vez queremos usar `|` (que em inglês, será `OR`).

```
<ip> BITOP OR
```

Iremos gerar a chave `acesso:25-ou-26-06-2015` e nos basearemos em dois conjuntos de *bits*: `acesso:25-06-2015` e `acesso:26-06-2015`.

```
<ip> BITOP OR acesso:25-ou-26-06-2015 acesso:25-06-2015 acesso:26-06-2015
```

Isto é, selecionaremos essas duas sequências:

(25)	1	1	0
(26)	1	0	1

E aplicaremos o operador `OR`. O resultado será a sequência: 1-1-1.

(25 OU 26)	1	1	1

Armazenaremos os dados, porque queremos utilizá-los. Então, usando as sequências no dia 25 e 26, iremos armazenar o resultado na chave `acesso:25-ou-26-06-2015`.

Sempre armazenaremos os resultados na primeira chave e depois, teremos um série de sequências que queremos analisar. O resultado da análise será 1:

```
<ip> BITOP OR acesso:25-ou-26-06-2015 acesso:25-06-2015 acesso:26-06-2015
(integer) 1
```

Agora, vamos ver se os *bits estão ativos... Começaremos com um `GETBIT` e a chave `acesso:25-ou-26-2015`. O primeiro usuário acessou o sistema?

```
<ip> GETBIT acesso:25-ou-26-2015 1
(integer) 0
```

Não. Tem algo errado no resultado, porque o *id* 1 acessou o sistema. A chave correta é `25-ou-26-06-2015`, ao invés de `25-ou-26-2015`. O nosso erro de digitação modificou o resultado. No entanto, se testarmos novamente com a chave correta, o resultado será o que corresponde.

```
<ip> GETBIT acesso:25-ou-26-2015 1
(integer) 0
<ip> GETBIT acesso:25-ou-26-06-2015 1
(integer) 1
```

O usuário 2 acessou? Sim. E o usuário 3? Sim, ele também acessou.

```
<ip> GETBIT acesso:25-ou-26-2015 1
(integer) 0
<ip> GETBIT acesso:25-ou-26-06-2015 1
(integer) 1
<ip> GETBIT acesso:25-ou-26-06-2015 2
(integer) 1
```

```
<ip> GETBIT acesso:25-ou-26-06-2015 3
(integer) 1
```

Para saber quantos usuários acessaram no fim de semana, faremos um `BITCOUNT`:

```
<ip> GETBIT acesso:25-ou-26-06-2015 1
(integer) 1
<ip> GETBIT acesso:25-ou-26-06-2015 2
(integer) 1
<ip> GETBIT acesso:25-ou-26-06-2015 3
(integer) 1
<ip> BITCOUNT acesso:25-ou-26-06-2015
```

O *Redis* irá devolver o resultado 3.

```
<ip> BITCOUNT acesso:25-ou-26-06-2015
(integer) 3
```

Nós contamos quantos usuários acessaram o nosso sistema, usando o `Bitop` e o `Bitcount` com o `OR`.

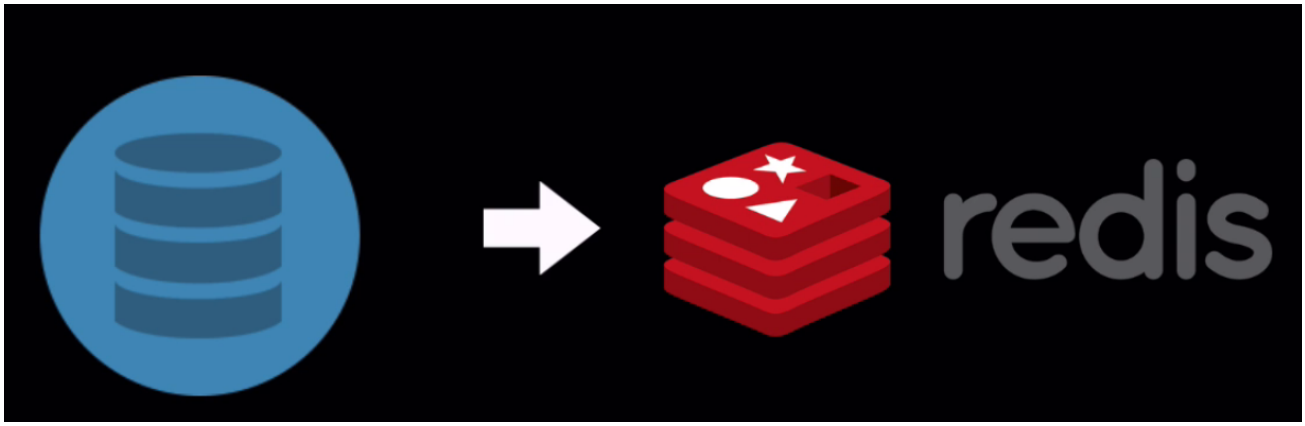
##Resumo

Nós vimos que o *Redis* é capaz de armazenar uma sequência de *bits*, um exemplo foi quando usamos uma delas para identificar quais usuários acessaram o sistema no dia 25 (1-1-0). Depois, quais usuários acessaram no dia 26 (1-0-1) e no dia 27 (1-1-1).

Se podemos marcar com 0 e 1, podemos usar os `Bitmaps`. Usamos também o `Setbit` para setar e o `Getbit` para "pegar" valores. Com o `Bitop` aplicamos um operador em duas sequências de *bits* e o `Bitcount` para contar quantos *bits* estão ativos em uma sequência. Observe que os nossos usuários usaram os *ids* 1, 2 e 3. Mas em exemplos anteriores, usamos

ids como 15, 37, 42... O importante é que eles sejam positivos. O *Redis* suporta que o número que identifica o *bit* analisados sejam do 0 para cima.

Concluímos aqui a primeira parte de introdução ao *Redis*, onde vimos vários comandos e como trabalhar com chaves e fazer perguntas para elas com os ***keys***: por exemplo, como setar valores *String*s, colocar, trocar e buscar os valores das chaves. Se forem valores numéricos, vimos como adicionar ou tirar números (*Increase/Decrease*). Vimos que se um número não for inteiro, ou seja, for um *float* e tiver pontos decimais, podemos trabalhar da mesma forma. Se quisermos incrementar uma sequência, com um número específico, por exemplo, com 37,3. E vimos que podemos representar marcações para usuários que acessaram ou não o sistema, com *bits* (os números 0 e 1). Podemos representar os *bits* de maneira otimizada no *Redis*. Lembre-se que para tudo que estamos usando o *Redis*, por padrão, está na memória do computador. Se desligarmos o *Redis*, perderemos os dados na memória. Por padrão, tudo que armazenamos no *Redis* são valores cacheados, que podemos recalcular a qualquer instante. É por isso, que encontramos constantemente recomendações para usarmos outro banco de dados para armazenar a origem das informações, processá-las e então, armazená-las processadas no *Redis*. Assim, as pesquisas serão realizadas mais rapidamente.



Porém, os dados originais estarão armazenados em outros bancos como o *MySQL*. Se por alguma razão perdermos os dados do *Redis*, poderemos gerá-los novamente. Existem opções avançadas no *Redis* para armazenar estes dados num banco externo. Por padrão, uma das razões do seu grande uso no mercado é justamente sua habilidade de executar todas estas operações na memória com *bits*, chaves do tipo *String*s e de forma ágil. Esta é a sacada do *Redis* e por isso, ele é tão utilizado - juntamente com uma fonte original de dados externa. Nós continuaremos a falar mais sobre ele, em seguida.