

# NoSQL Key-Value

Dr. Gerardo Rossel

FCEyN - UBA

1er Cuatrimestre 2024

- Dynamo-Amazon
  - Giuseppe DeCandia, et al., Dynamo: **Amazon's highly available key-value store**. In Proceedings of twentyfirst ACM SIGOPS symposium on Operating systems principles (SOSP '07). ACM, New York, NY, USA,

## Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

### ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

# Key-Value

- Diccionario o array asociativo

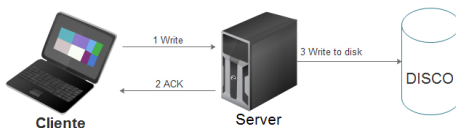
Clave	Valor
Juan	(123) 456-7890
Pedro	(234) 567-8901
Maria	(345) 678-9012
Francisca	(456) 789-0123

- Key-Value:

- Namespaces o Buckets
- In-Memory vs On-Disk

- Características

- Simples
- Escalables
- Veloces



# ¿Claves?

- Bases relacionales:
  - Garantizar la inmutabilidad de la la clave primaria.
  - Se usan claves sin significado.
- Key-Value
  - No hay columnas, no hay manera de saber el significado de un valor excepto dándole semántica a la clave.  
*Cart[12387] = 'SKUAK8912j4'*  
*CustName[12387] = 'Katherine Smith'*

# ¿Cómo construir una clave?



# ¿Cómo construir una clave?

## Entity Name + ':' + Entity Identifier + ':' + Entity Attribute

- *Cliente* : 12345678 : *Apellido*
- *Producto* : *SKU110* : *Nombre*
- Dependiendo de la BD hay soporte para varios tipos en los valores.
  - Redis soporta valores de: Strings, Lists, Sets, Sorted sets, Hashes, Bit Arrays
  - Keys en Redis son *binary safe*
- **Nota:** las claves sirven también para organizar valores en múltiples servers

# Key-Value: Keys

- Usar nombres significativos y no ambiguos
- Usar un delimitador común ":"
- Mantener las claves lo más cortas posibles sin sacrificar las otras características.

```
define getCustAttr(p_id , p_attrName)
    v_key = 'cust'+':'+p_id+':'+p_attrName;
    return (AppNameSpace[v_key]);

define setCustAttr(p_id , p_attrName , p_value)
    v_key = 'cust'+':'+p_id+':'+p_attrName;
    AppNameSpace[v_key]=p_value
```

# Key-Value: Values

- String: '1232 NE River Ave, St. Louis, MO'
- Lista: ('1232 NE River Ave', 'St. Louis', 'MO')
- HASH: **Street** '1232 NE River Ave' **City** 'St. Louis' **State** 'MO'
- JSON:

```
{ "Street" : "1232 NE River Ave", "City" : "St. Louis", "State" : "MO" }
```

---

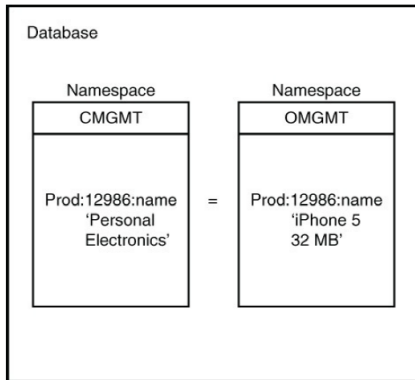


- Múltiples aplicaciones utilizando la misma db
- Definen la misma clave para guardar info de productos:

**"Prod:< idProd >:name"**








- Guardan distintos valores para un mismo producto
- Qué pasa?

Espacios de nombre permiten evitar conflictos



# Tiempo de Vida

Venta de tickets. Guardar asientos mientras se procesa la compra.

Key	TTL
'U7138'	
R3194	
S2241	
T1294	
K4111	
R1143	
S1914	

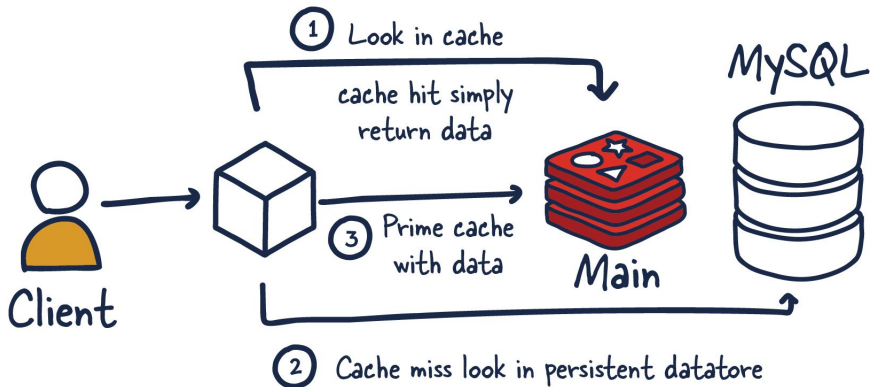


# Introducción a Redis





# How is redis traditionally used





Las claves de Redis son *binary safe*, esto significa que puede utilizar cualquier secuencia binaria como clave, desde una cadena como "foo" hasta el contenido de un archivo JPEG. La cadena vacía también es una clave válida.

## **Tipos de Datos:**

- String
- List
- Hash
- Set
- Sorted Set
- Bitmaps (basado en string pero como semántica propia)
- HyperLogLogs (basado en string pero como semántica propia)
- Streams



Un String puede comportarse como un entero, float, string de texto o mapa de bits en función de su valor y los comandos utilizados.

- 1 SET, GET, MSET y MGET
  - >**set** clave valor
  - >**get** clave
  - "valor"
  - >**mset** clave1 10 clave2 20 clave3 30
- 2 INCR, DECR, INCRBY, DECRBY y INCRBYFLOAT
  - >**set** contador 100
  - >**incr** contador
  - (integer) 101
  - >**incrbyfloat** clave 0.1
  - "0.1"

# Opciones del comando SET

- ❶ EX – Establece el tiempo de expiración especificado, en segundos.
- ❷ PX – Establece el tiempo de expiración especificado, en milisegundos.
- ❸ NX – Pone el valor únicamente si la clave aún no existe.
- ❹ XX – Pone el valor únicamente si la clave ya existe.
- ❺ KEEPTTL – Conserva el tiempo de vida (TTL )asociado con la clave

>**set** clavetemp "vence en 1 minuto" EX 60

>**ttl** clavetemp

(integer) 59





El acceso a elementos de una lista es  $O(n)$  salvo el primero y el último que es tiempo constante. Es una lista encadenada.

- LPUSH, RPUSH, RPOP, LPOP, LLEN, LINDEX, LRANGE, LRANGE, LTRIM

```
$ redis-cli
127.0.0.1:6379> LPUSH books "Clean Code"
(integer) 1
127.0.0.1:6379> RPUSH books "Code Complete"
(integer) 2
127.0.0.1:6379> LPUSH books "Peopleware"
(integer) 3
```



El acceso a elementos de una lista es  $O(n)$  salvo el primero y el último que es tiempo constante. Es una lista encadenada.

- LPUSH, RPUSH, RPOP, LPOP, LLEN, LINDEX, LRANGE, LRANGE, LTRIM

```
$ redis-cli
127.0.0.1:6379> LPUSH books "Clean Code"
(integer) 1
127.0.0.1:6379> RPUSH books "Code Complete"
(integer) 2
127.0.0.1:6379> LPUSH books "Peopleware"
(integer) 3
127.0.0.1:6379> LLEN books
(integer) 3
127.0.0.1:6379> LINDEX books 1
"Clean Code"
```



El acceso a elementos de una lista es  $O(n)$  salvo el primero y el último que es tiempo constante. Es una lista encadenada.

- LPUSH, RPUSH, RPOP, LPOP, LLEN, LINDEX, LRANGE, LTRIM

```
$ redis-cli
127.0.0.1:6379> LPUSH books "Clean Code"
(integer) 1
127.0.0.1:6379> RPUSH books "Code Complete"
(integer) 2
127.0.0.1:6379> LPUSH books "Peopleware"
(integer) 3
127.0.0.1:6379> LLEN books
(integer) 3
127.0.0.1:6379> LINDEX books 1
"Clean Code"
127.0.0.1:6379> LRANGE books 0 1
1) "Peopleware"
2) "Clean Code"
```



Hashes son mapas compuestos por campos asociados a valores. Tanto el campo como el valor son strings

- HSET, HGET, HMSET, HMGET, HDEL, HLEN, HMGETALL, HKEYS, HVALS, HSETNX

```
> hmset user:1000 username antirez birthyear 1977 verified 1
```

```
OK
```



Hashes son mapas compuestos por campos asociados a valores. Tanto el campo como el valor son strings

- HSET, HGET, HMSET, HMGET, HDEL, HLEN, HMGETALL, HKEYS, HVALS, HSETNX

```
> hmset user:1000 username antirez birthyear 1977 verified 1
OK
```

```
> hget user:1000 username
"antirez"
```

```
> hget user:1000 birthyear
"1977"
```



Hashes son mapas compuestos por campos asociados a valores. Tanto el campo como el valor son strings

- HSET, HGET, HMSET, HMGET, HDEL, HLEN, HMGETALL, HKEYS, HVALS, HSETNX

```
> hmset user:1000 username antirez birthyear 1977 verified 1
```

```
OK
```

```
> hget user:1000 username
```

```
"antirez"
```

```
> hget user:1000 birthyear
```

```
"1977"
```

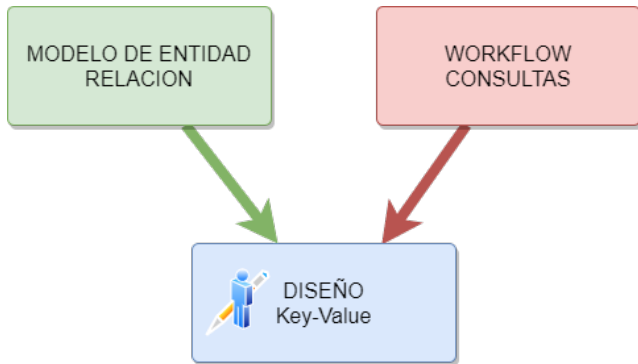
```
> hmget user:1000 username birthyear no-such-field
```

```
1) "antirez"
```

```
2) "1977"
```

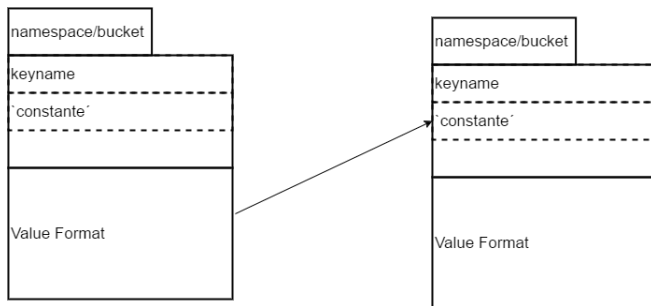
```
3) (nil)
```

# Diseño de Key-Value Data Stores



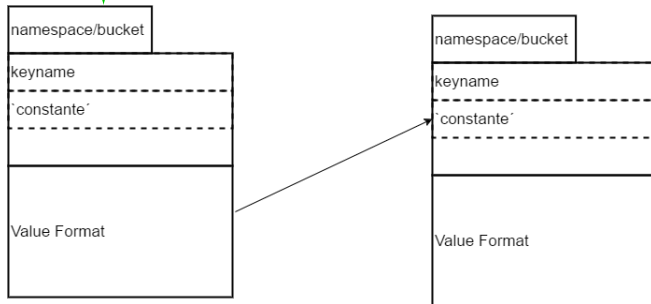


# Notación Key-Value



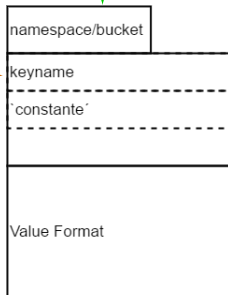
# Notación Key-Value

Espacio de nombres  
optativo

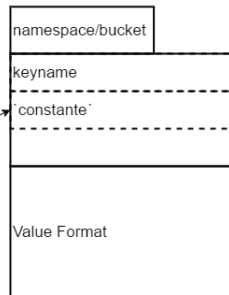


# Notación Key-Value

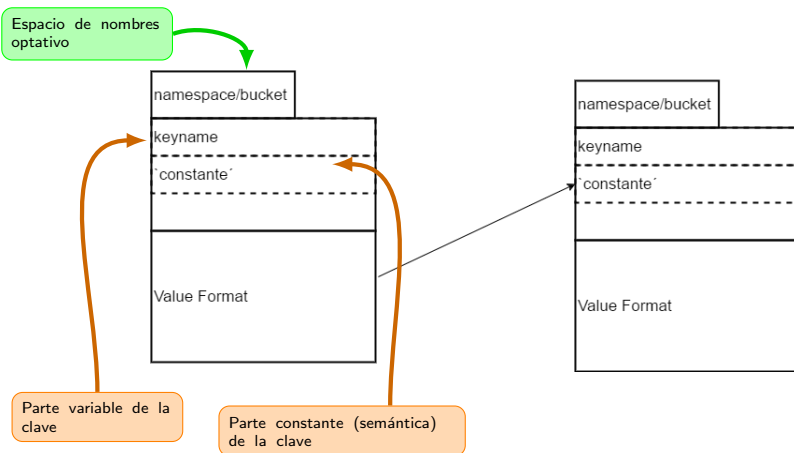
Espacio de nombres  
optativo



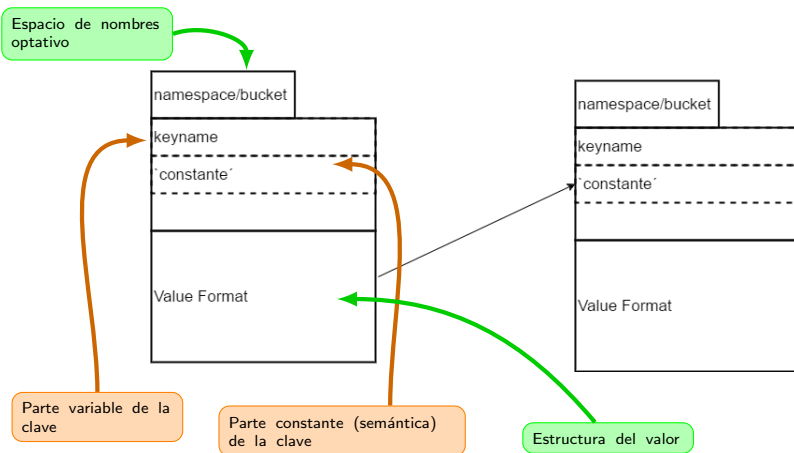
Parte variable de la  
clave



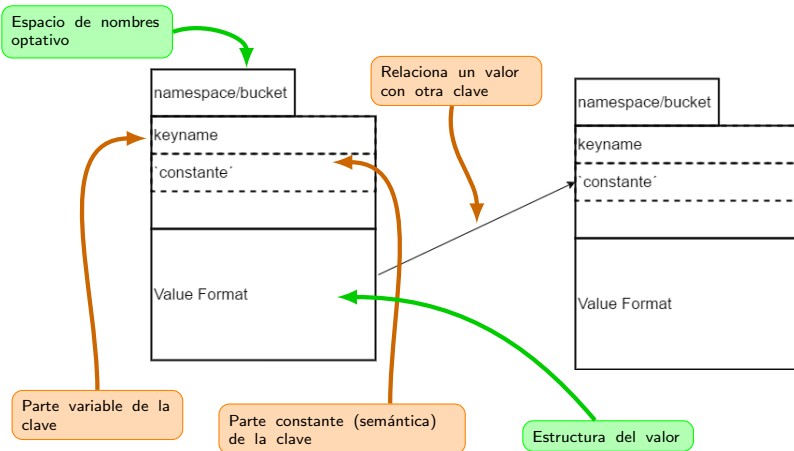
# Notación Key-Value



# Notación Key-Value



# Notación Key-Value

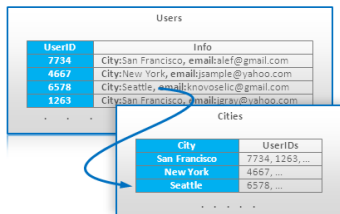


# Notación Key-Value: Valores

```
ConcertApp[ticketLog:9888] = {"conDate":15-Mar-2015, "locDescr": "Springfield Civic Center", "assgnSeat": "J38"}
```

ConsertApp	ConsertApp	ConsertApp
'ticketlog'	'ticketlog'	'ticketlog'
nroticket: int	nroticket: int	nroticket: int
JSON: "assgnSeat": {"type": "string"}, "conDate": {"type": "string", "format": "date - time"}, "locDescr": {"type": "string"}	HASH assignSeat: string conDate: date locDescr: string	JSON: Schema ConsertApp

# Notación Key-Value: Referencias

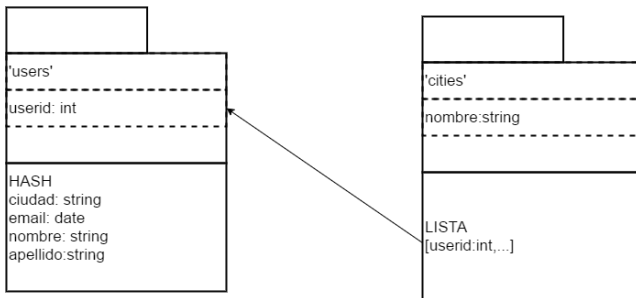




# Notación Key-Value: Referencias

UserID	Info
7734	City:San Francisco, email:alef@gmail.com
4667	City:New York, email:sample@yahoo.com
6578	City:Seattle, email:knovoselic@gmail.com
1263	City:San Francisco, email:jiray@yahoo.com

City	UserIDs
San Francisco	7734, 1263, ...
New York	4667, ...
Seattle	6578, ...

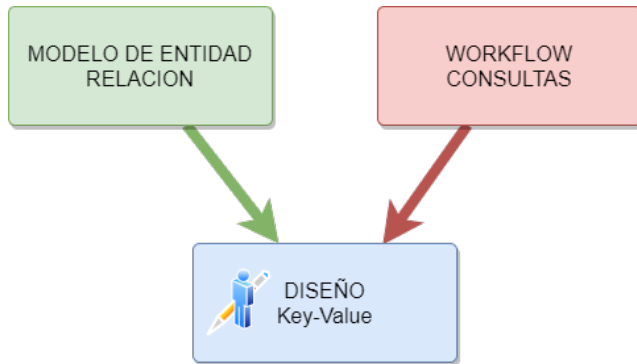


# Ejemplo Videojuego

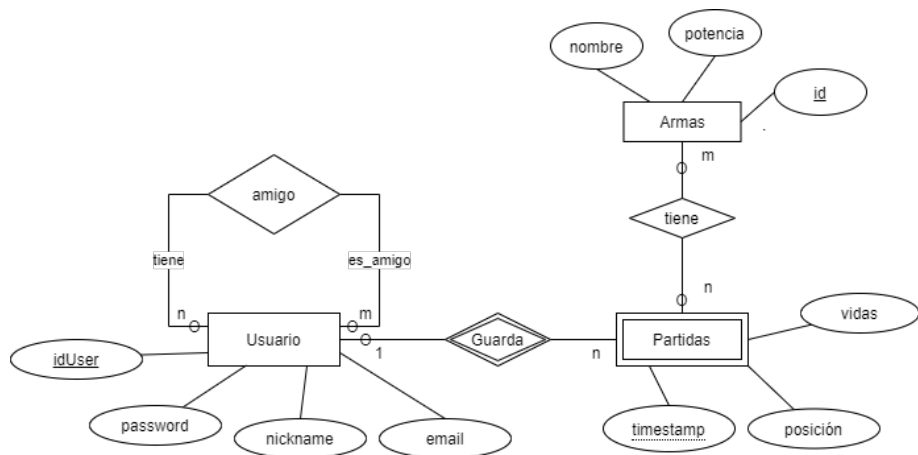
## Requerimientos:

- Los usuarios tienen nick, password, email y a otros usuarios como amigos
- Los usuarios pueden ver sus datos personales así como los datos de sus amigos.
- Un usuario puede guardar su partida en cualquier momento, se guarda la vida, la posición y las armas que posee en esa partida.
- Puede elegir continuar desde cualquier partida guardada.
- Al recuperar una partida se debe recuperar la posición, vida y armas que tenía.
- Las armas tienen un nombre y una potencia.

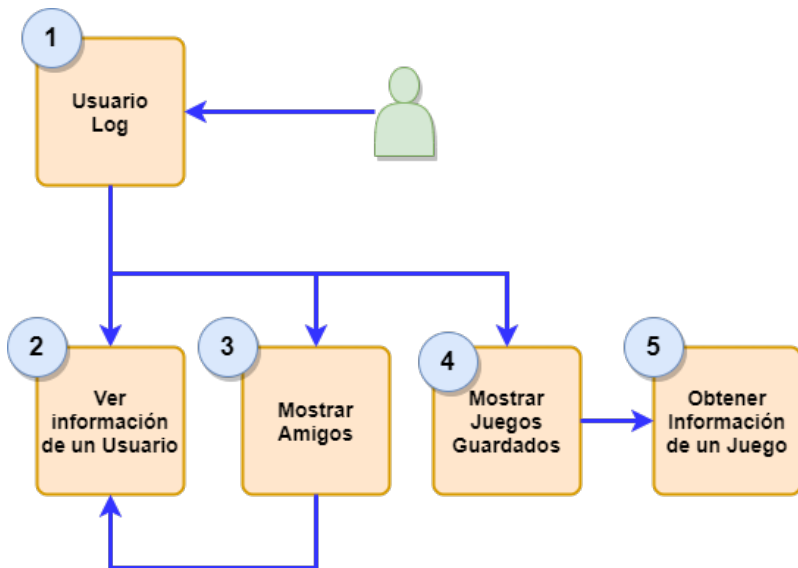
# Ejemplo Videojuego



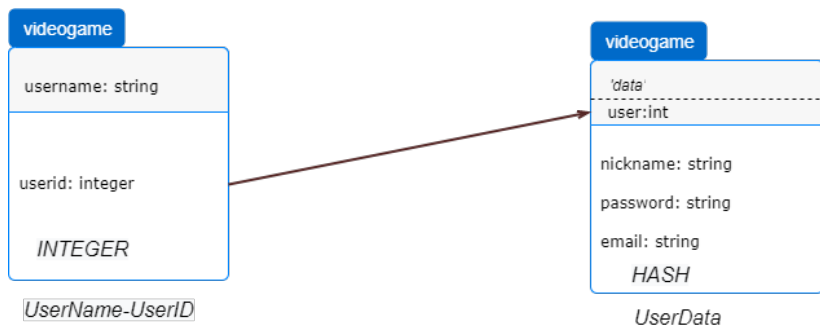
# Ejemplo Videojuego DER



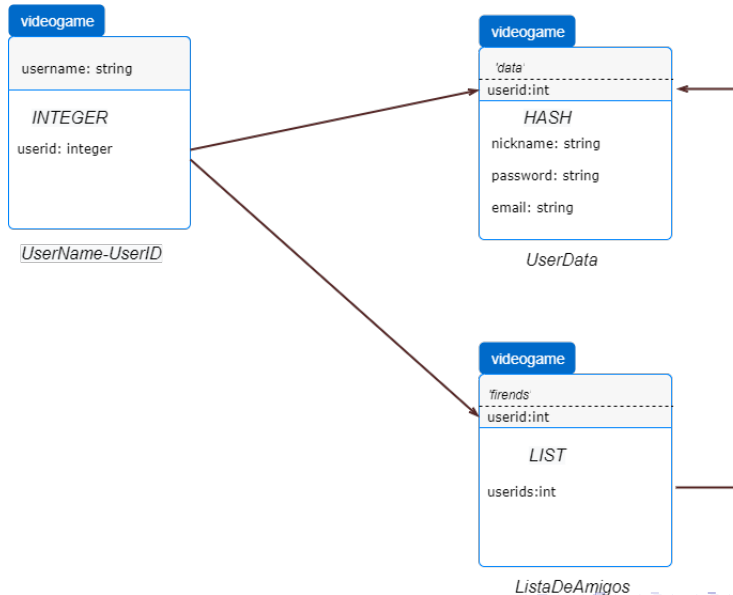
# Ejemplo Videojuego Workflow



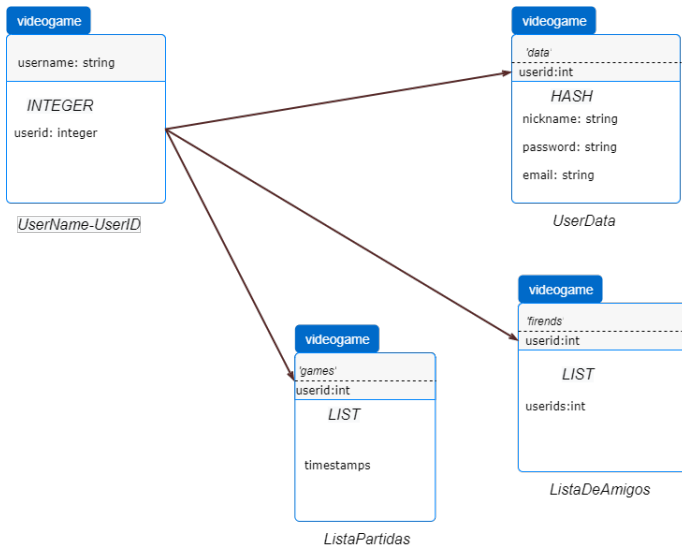
# Ejemplo Videojuego - Solución Usuarios



# Ejemplo Videojuego - Solución Usuarios-Amigos

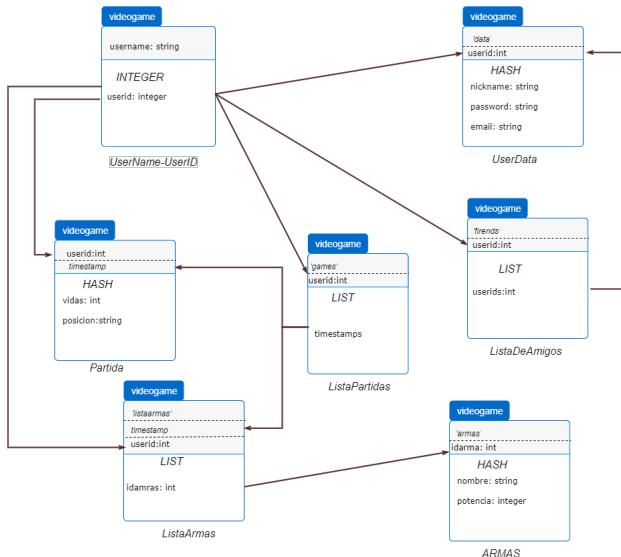


# Ejemplo Videojuego - Solución Lista de Partidas





# Ejemplo Videojuego Solución



- Redis Essentials. Maxwell Dayvson Da Silva and Hugo Lopes Tavares. 2015. Packt Publishing.
- Redis in Action Josiah L. Carlson. 2013. Manning Publications Co., USA.
- NoSQL for Mere Mortals - Dan Sullivan. 2015. Addison-Wesley Professional.
- NoSQL Distilled. *A Brief Guide to the Emerging World of Polyglot Persistence* - Pramod J. Sadalage y Martin Fowler 2012. Addison-Wesley Professional.
- A modeling methodology for NoSQL Key-Value databases. Database Systems Journal, Vol VIII, Issue: 2/2017. ISSN 2069 - 3230- Gerardo Rossel, Andrea Manna