In-memory databases

Redis for large data projects

TE 03/04/25

Schedule of sessions

- 3 April: In-memory databases introduction
- 7 April: *Redis Tutorial*
- 10 April : *Redis lab*

- ***Redis Assignment final date: April 27th***

Simple initial starting point

- Data storage requirements in a single address-space
- Complexity of the solution is reduced
  - No need to page information out of memory
  - No need to design consistency methods
- Fast access to large memory space
- Data structures no need to be optimized for disk anymore

Main problems to solve

- Durability: what to do when system shutdowns?

- Capacity: what to do when data needs are larger than memory space?

Distributed in-memory solutions

- Shared-Nothing architecture
- Capacity problem:
  - Add more servers to the system
- Durability problem:
  - Create and distribute data copies
  - Increase redundancy
- Oracle Coherence, VoltDB, SAP HANA

Redundancy: now we have new problems!

- Complex queries will need to read all distributed data copies
- Need of a distributed global join
  - data update: different values of the same query
  - old values must be overwritten by new in each copy
- Avoid the need of cross-partition join queries
  - NoSQL databases use local data

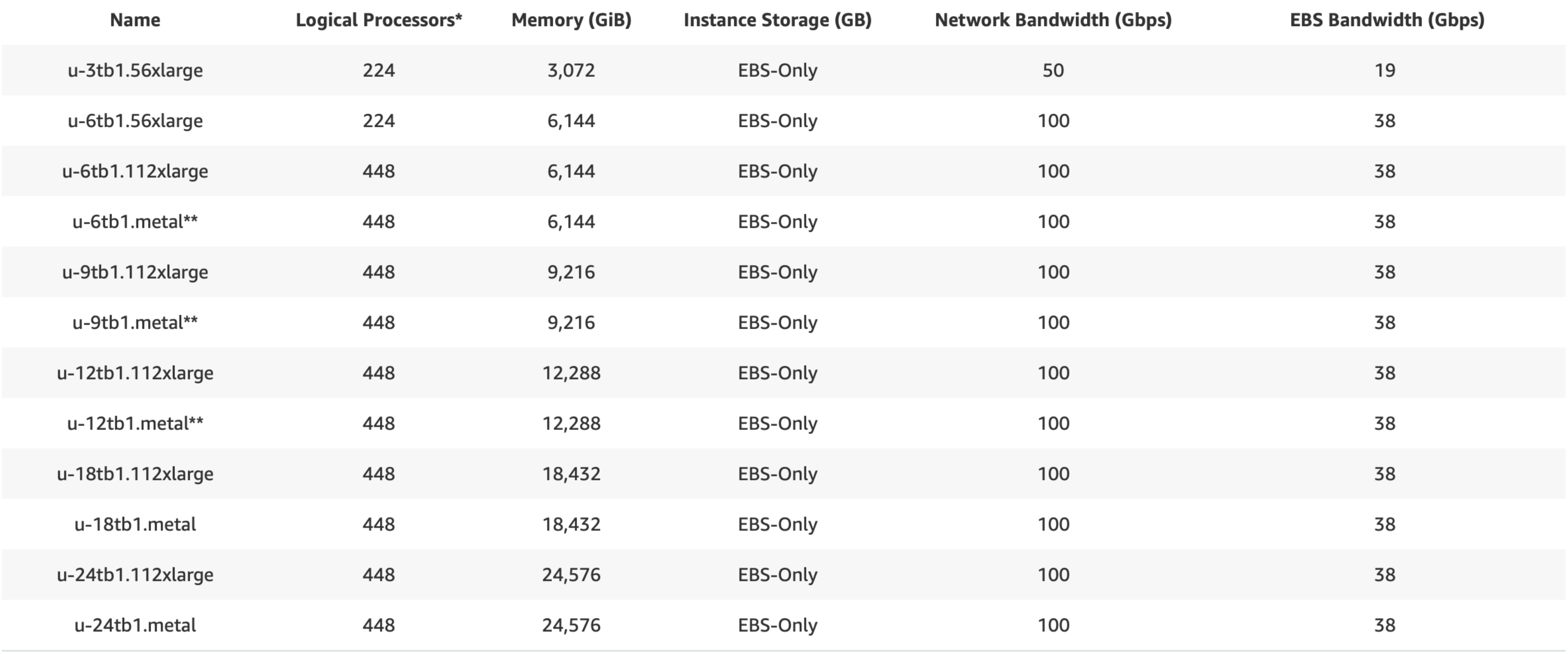
Technology advances help database systems

- many-core processors
- TB of DRAM
- TB of local SSD memory storage
  - 100s slower than memory, but 10x faster than disk

What do these changes mean?

- If your dataset fits in memory: better use an in-memory DB
  - Much faster than disk
- RAM Memory+SSD become a new standard
  - Already default in Cloud Database services (AWS EBS volumes)
- majority of DB are not so big

High-memory AWS EC2 instances



Name	Logical Processors*	Memory (GiB)	Instance Storage (GB)	Network Bandwidth (Gbps)	EBS Bandwidth (Gbps)
u-3tb1.56xlarge	224	3,072	EBS-Only	50	19
u-6tb1.56xlarge	224	6,144	EBS-Only	100	38
u-6tb1.112xlarge	448	6,144	EBS-Only	100	38
u-6tb1.metal**	448	6,144	EBS-Only	100	38
u-9tb1.112xlarge	448	9,216	EBS-Only	100	38
u-9tb1.metal**	448	9,216	EBS-Only	100	38
u-12tb1.112xlarge	448	12,288	EBS-Only	100	38
u-12tb1.metal**	448	12,288	EBS-Only	100	38
u-18tb1.112xlarge	448	18,432	EBS-Only	100	38
u-18tb1.metal	448	18,432	EBS-Only	100	38
u-24tb1.112xlarge	448	24,576	EBS-Only	100	38
u-24tb1.metal	448	24,576	EBS-Only	100	38

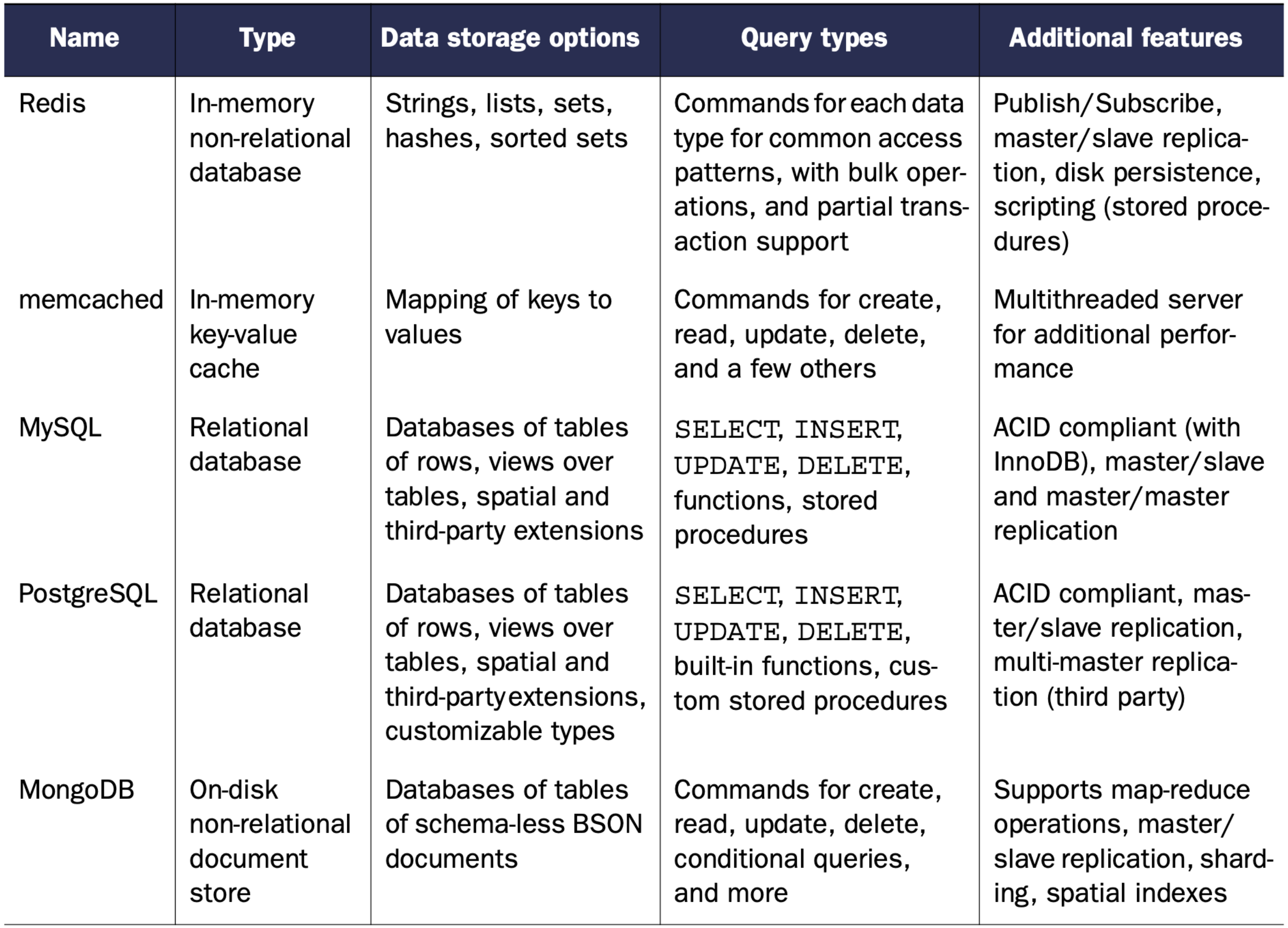
redis

- Memory as main storage support
  - If dataset is bigger than memory: cannot use Redis
  - Not a substitute for general DBMS, good for memory intensive apps
- Disk only for persistence
- Data structure oriented: more complex than simple {key,value} stores
- No need to switch to Redis: use it for partial solutions
- Open source + Redis Labs support
- Redis annual conference: redislabs.com/redisconf/sessions

Why Redis?

- Available as a service in cloud providers (aws,google,azure)
  - https://aws.amazon.com/redis
- Who’s using Redis? Twitter, Snapchat, Github, Coinbase, Shopify, ...
  - https://techstacks.io/tech/redis
- Project presentations from Redisconf
  - Data warehousing – IBM
  - Machine learning – RedisAI – Atlassian
  - Redis on the 5G Edge – Verizon
  - Microservices and Redis – AWS
  - Redis and Apache Kafka – Microsoft

Features and functionality of databases



Name	Type	Data storage options	Query types	Additional features
Redis	In-memory non-relational database	Strings, lists, sets, hashes, sorted sets	Commands for each data type for common access patterns, with bulk operations, and partial transaction support	Publish/Subscribe, master/slave replication, disk persistence, scripting (stored procedures)
memcached	In-memory key-value cache	Mapping of keys to values	Commands for create, read, update, delete, and a few others	Multithreaded server for additional performance
MySQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions	`SELECT`, `INSERT`, `UPDATE`, `DELETE`, functions, stored procedures	ACID compliant (with InnoDB), master/slave and master/master replication
PostgreSQL	Relational database	Databases of tables of rows, views over tables, spatial and third-party extensions, customizable types	`SELECT`, `INSERT`, `UPDATE`, `DELETE`, built-in functions, custom stored procedures	ACID compliant, master/slave replication, multi-master replication (third party)
MongoDB	On-disk non-relational document store	Databases of tables of schema-less BSON documents	Commands for create, read, update, delete, conditional queries, and more	Supports map-reduce operations, master/slave replication, sharding, spatial indexes

No tables! individual objects in memory

RELATIONAL TABLE

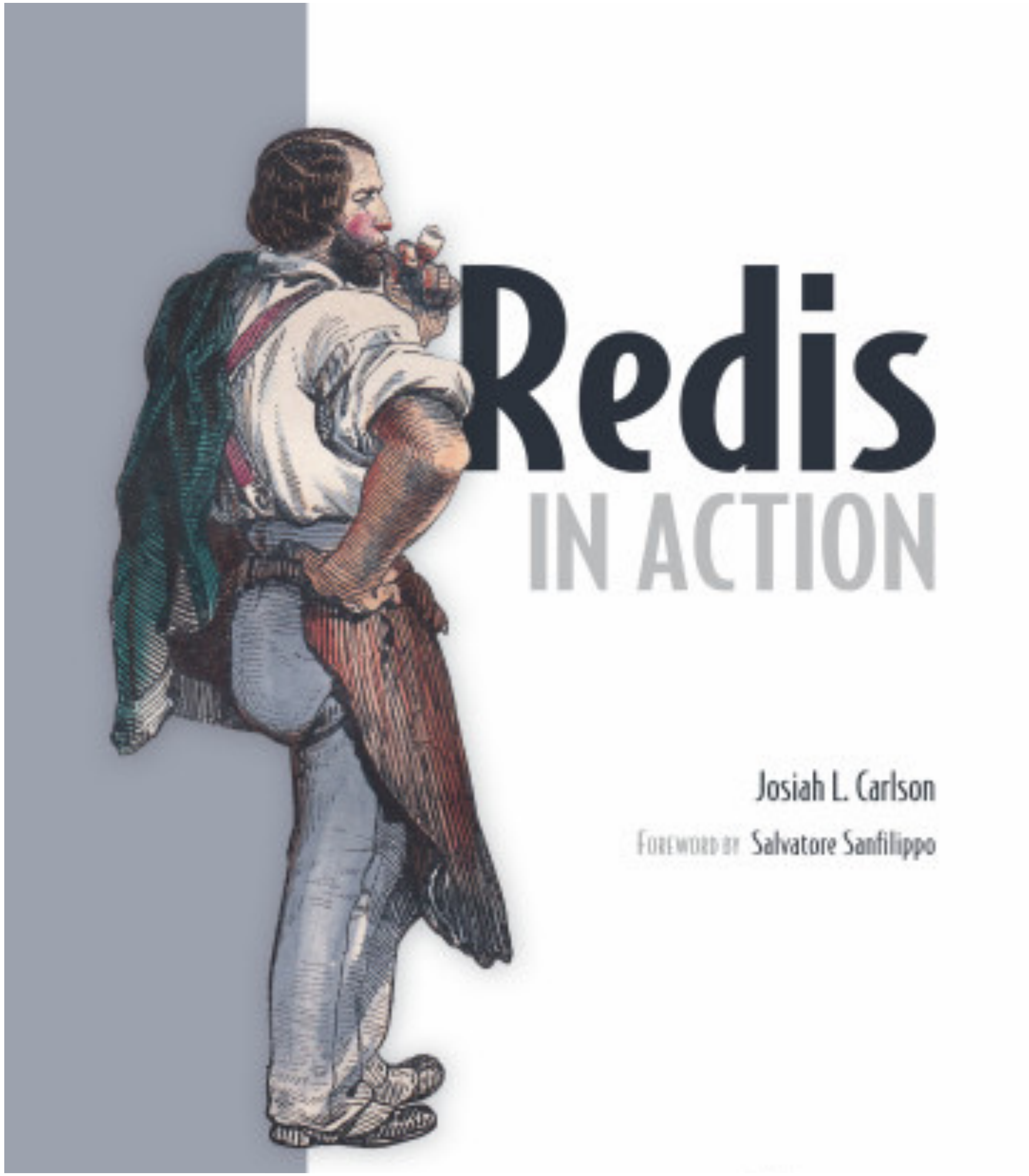
short	website
uab	www.uab.cat/enginyeria
google	www.google.com
yahoo	www.yahoo.com

REDIS IN-MEMORY OBJECTS

uab	www.uab.cat/enginyeria
google	www.google.com
yahoo	www.yahoo.com

Ready for changes

- React to user events with new structures
- Views are dynamic: no need to redeploy new database schema
- No migration process needed



Redis by example

Redis in Action, Josiah L. Carlson, Manning, 2013

Build a simple reddit.com

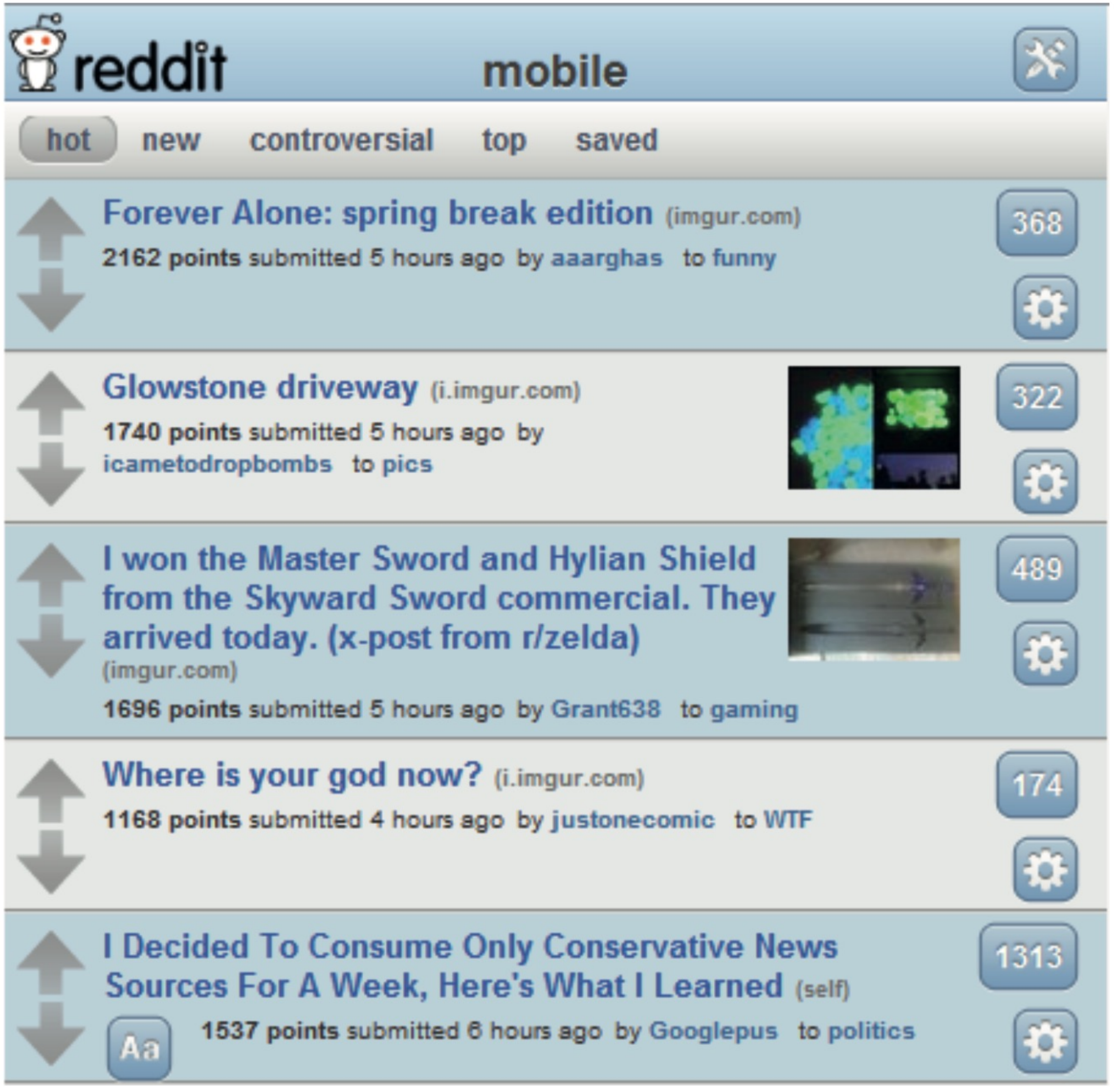
HASH, ZSET, SET

Redis Ubuntu installation

```
sudo apt install -y redis

redis-server -v

>Redis server v=5.0.7 sha=000000:0 malloc=jemalloc-5.2.1
bits=64 build=636cde3b5c7a3923
```

Interact with Redis CLI

```
$redis-cli

127.0.0.1>ping
PONG
127.0.0.1>quit
```



Forever Alone: spring break edition (imgur.com)
2162 points submitted 5 hours ago by aaarghas to funny

Glowstone driveway (i.imgur.com)
1740 points submitted 5 hours ago by icametodropbombs to pics

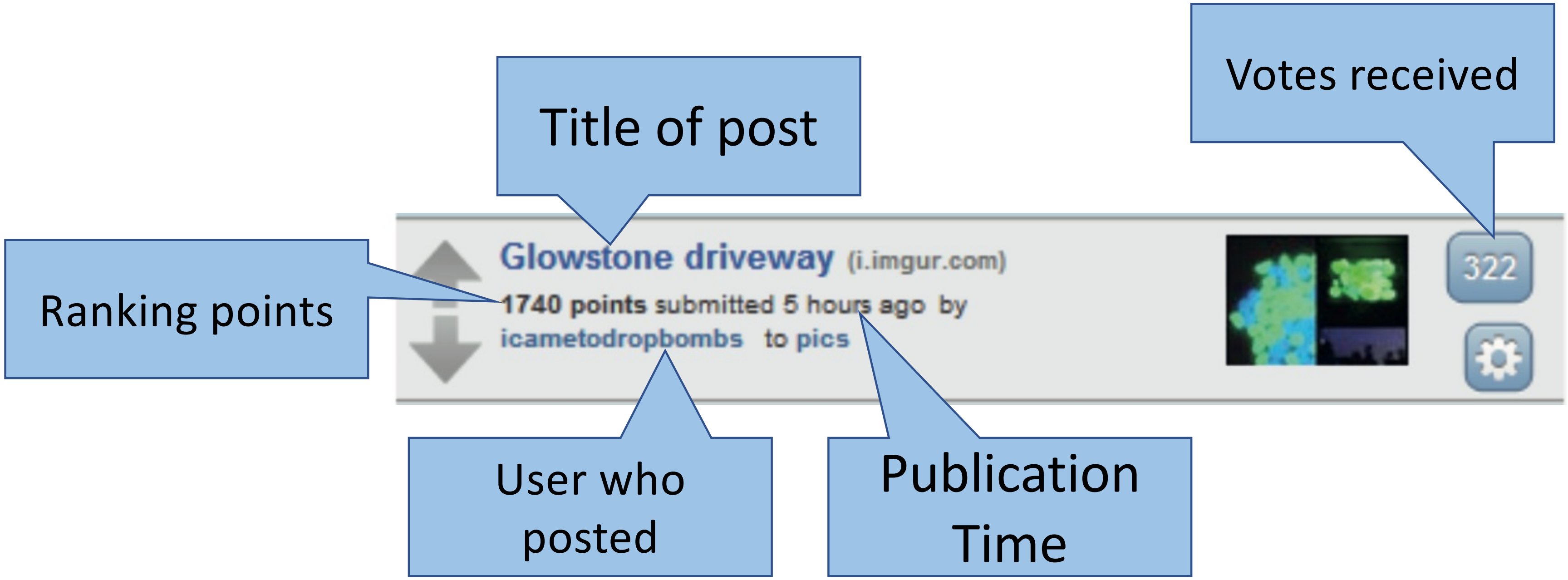
I won the Master Sword and Hylian Shield from the Skyward Sword commercial. They arrived today. (x-post from r/zelda) (imgur.com)
1696 points submitted 5 hours ago by Grant638 to gaming

Where is your god now? (i.imgur.com)
1168 points submitted 4 hours ago by justonecomic to WTF

I Decided To Consume Only Conservative News Sources For A Week, Here's What I Learned (self)
1537 points submitted 6 hours ago by Googlepus to politics

Articles can be voted on by clicking on up and down arrows.

Anatomy of a Reddit article



Glowstone driveway (i.imgur.com)

1740 points submitted 5 hours ago by icametodropbombs to pics

Title of post

Votes received

Ranking points

User who posted

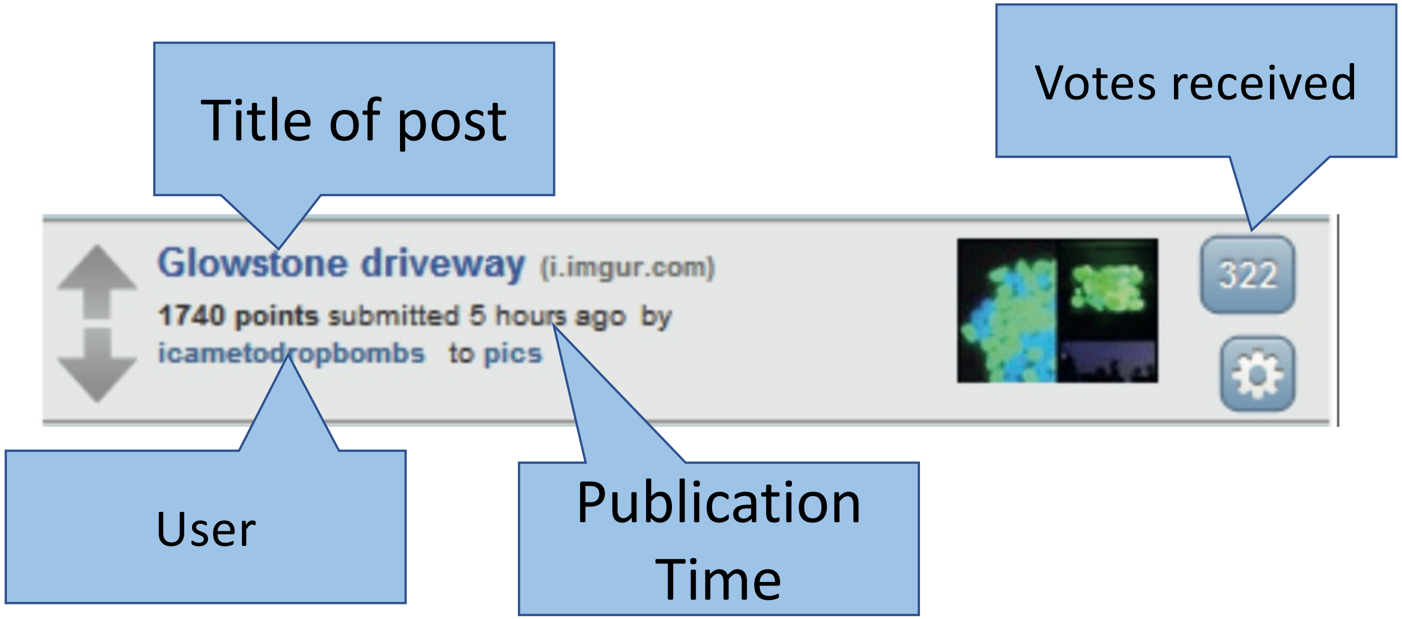
Publication Time

Article list ranking by users vote

- Web page to vote on published articles
- Posts have a ranking score (more votes -> more ranking)
- Posts are published following ranking (highest ranking -> position 1)
- Ranking score will decrease over time automatically (aging)
  - timestamp of post
  - score value = timestamp + number of votes * 500
    - what happens if no one votes for an article?

Data model entities: articles

Articles: title, user, votes received, publication time



Glowstone driveway (i.imgur.com)

1740 points submitted 5 hours ago by icametodropbombs to pics

322

Title of post

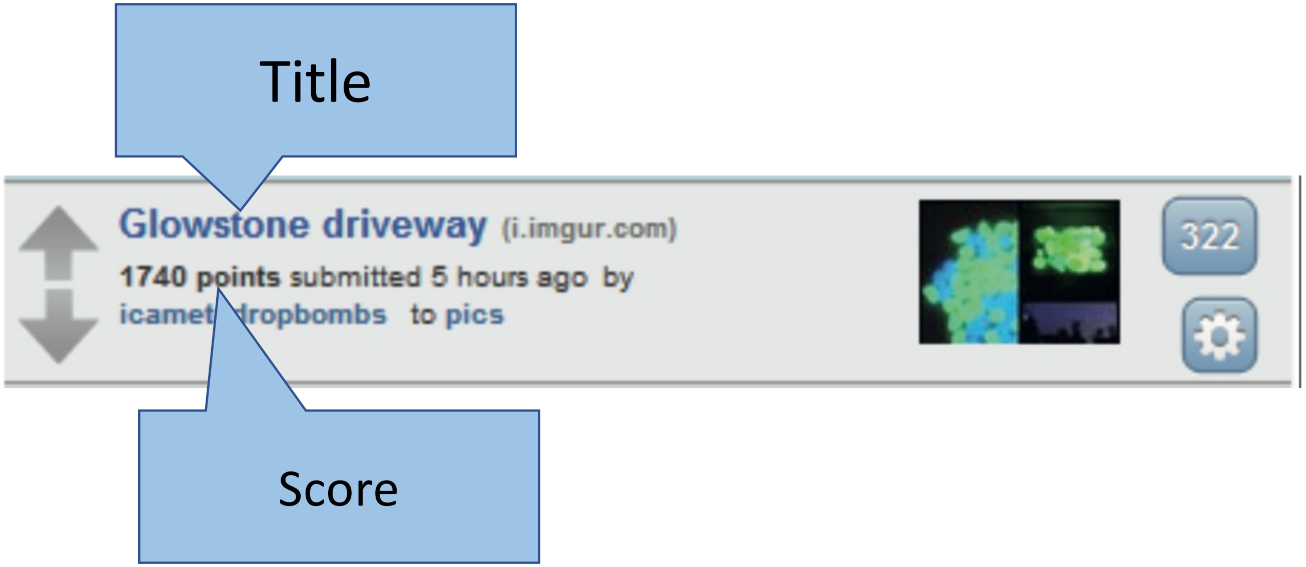
Votes received

User

Publication Time

Data model entities: ranking

- Article ranking: article id, article score



Title

Glowstone driveway (i.imgur.com)

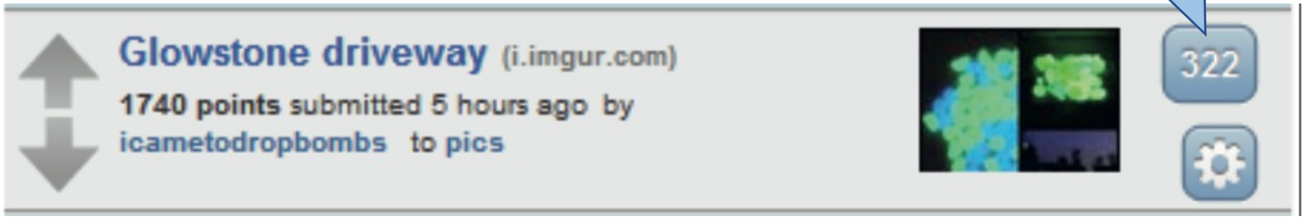
1740 points submitted 5 hours ago by icametodropbombs to pics

Score

Data model entities: voters

- Voters: users who voted for an article
  - users can only vote once per article

Votes received



Glowstone driveway (i.imgur.com)

1740 points submitted 5 hours ago by icametodropbombs to pics

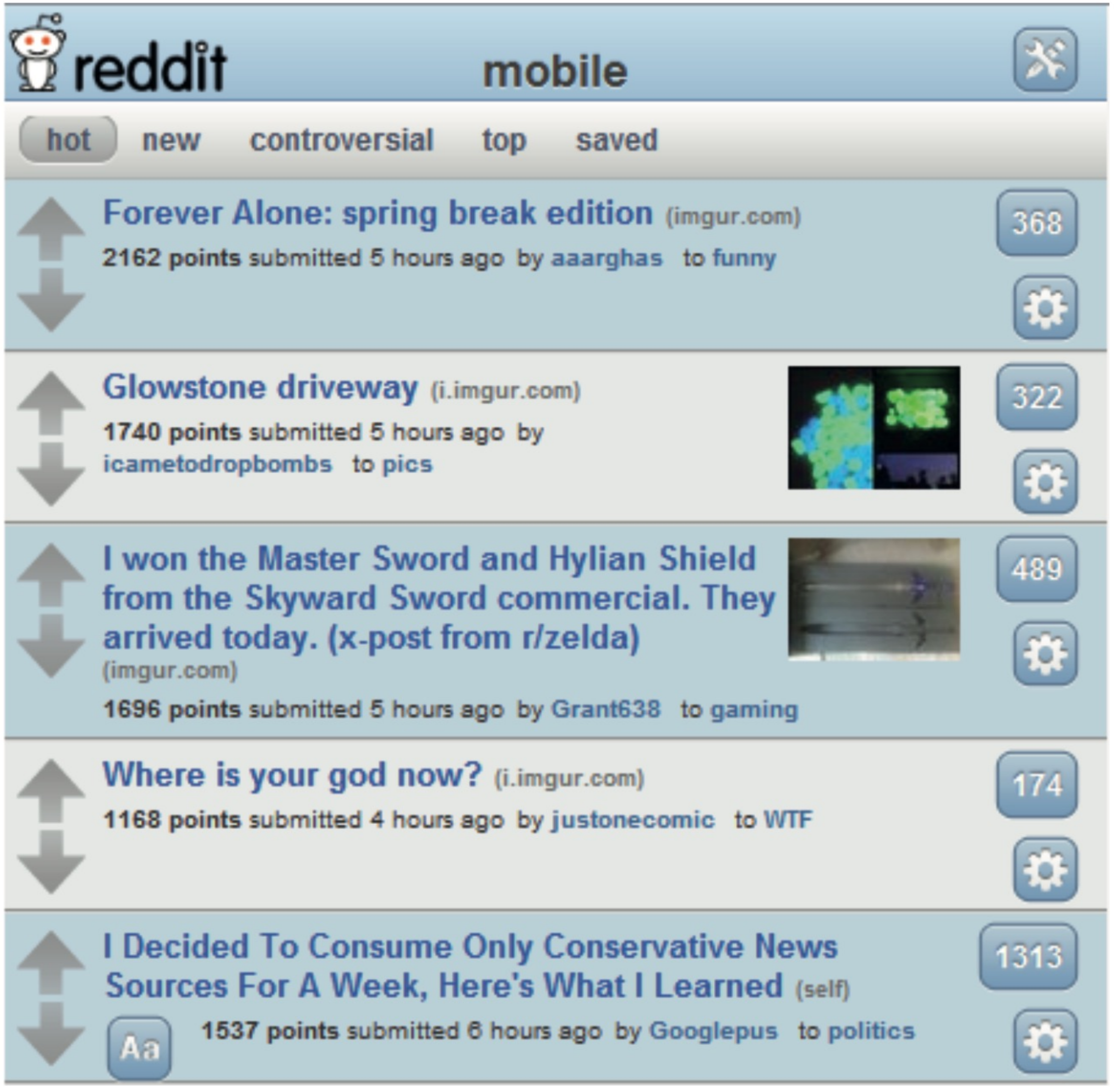
Data model entities

- Articles
- Article rankings
- Voters

- Redis data structures
  - How to store article, ranking and voters information?
  - How to read article information to implement applications?
- Implementation of Python functions with Redis CLI/API

Reddit ranking example view

Articles can be voted on by clicking on up and down arrows.



reddit mobile

hot new controversial top saved

Forever Alone: spring break edition (imgur.com)
2162 points submitted 5 hours ago by aaarghas to funny — 368

Glowstone driveway (i.imgur.com)
1740 points submitted 5 hours ago by icametodropbombs to pics — 322

I won the Master Sword and Hylian Shield from the Skyward Sword commercial. They arrived today. (x-post from r/zelda) (imgur.com)
1696 points submitted 5 hours ago by Grant638 to gaming — 489

Where is your god now? (i.imgur.com)
1168 points submitted 4 hours ago by justonecomic to WTF — 174

I Decided To Consume Only Conservative News Sources For A Week, Here's What I Learned (self)
1537 points submitted 6 hours ago by Googlepus to politics — 1313

How to store articles?

How to sort articles by score?

Learning how to use Redis Hash and Zset

1 – Redis Hash

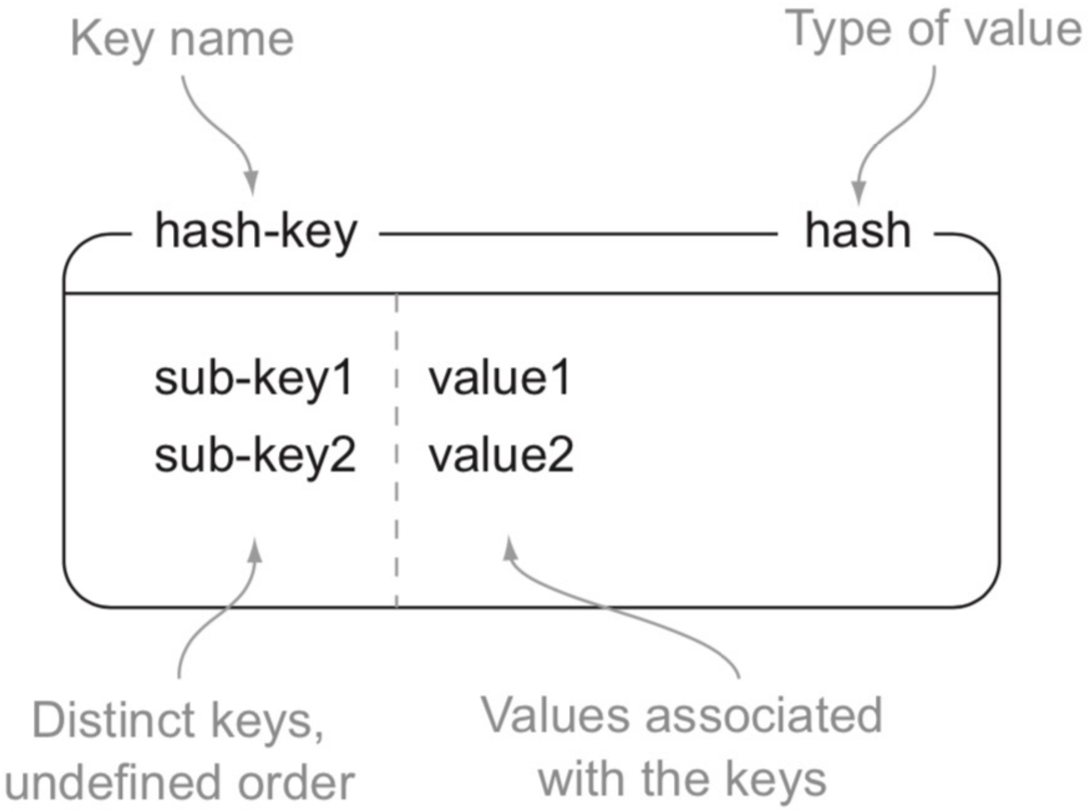
To keep elements of our dataset: one element, one object

Redis hash example: Article:00001

Object Key Name <article:number>

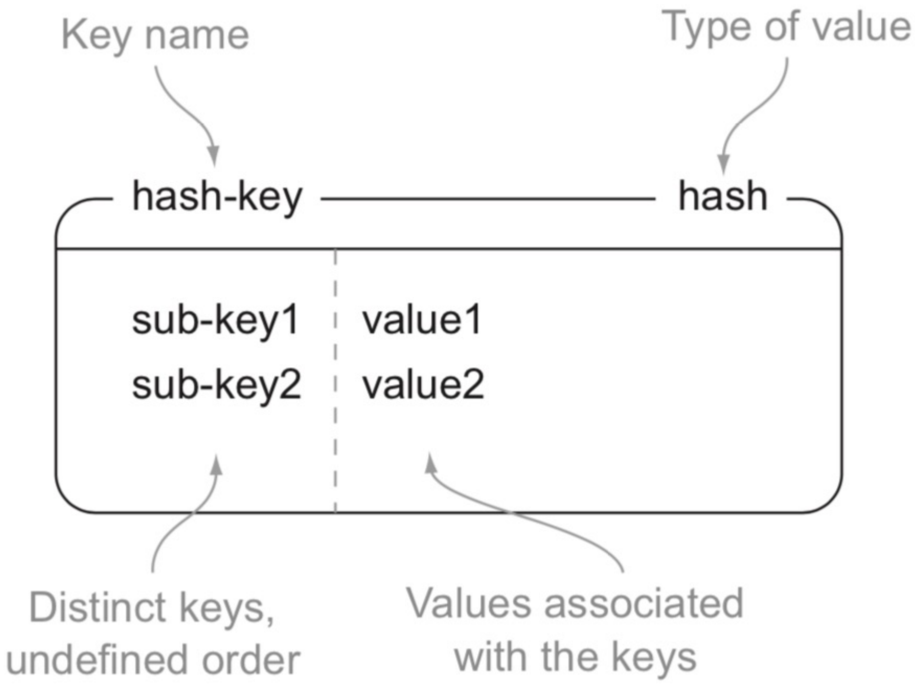
Subkeys for my article objects	article:00001
• **Title**	• *title: effects of palm oil in cancer*
• **Link** to article	• *link: http://pubmed.org/12345*
• **user** who posted it	• *user: toni*
• **Timestamp**	• *timestamp: 21/10/2023 10:00*
• **Votes** received	• *Votes: 0*

Redis HASH



Key name
Type of value
hash-key
hash
sub-key1
value1
sub-key2
value2
Distinct keys, undefined order
Values associated with the keys

Redis HASH


Key name
Type of value
hash-key
hash
sub-key1
value1
sub-key2
value2
Distinct keys, undefined order
Values associated with the keys

- New hash key: <article:1>
  - title: *effects of palm oil in cancer*
  - votes: *0*

- Redis operations
  - HSET: create new object
  - HGET: get hash-key from object
  - HMGET: get all values for an object
  - HDEL: eliminate subkey

Article object creation

Redis HASH object: object id

Object attributes:

- Title
- user who posted it
- Timestamp
- Votes received

- Object id: ?
- Object information
  - Title
  - User
  - Time
  - Votes

key design <article:00001>

- Very long keys are not a good idea: key lookup will take a long time
    - 12345678912345678
- Very short keys are not a good idea: too many lookup matches
    - 1234 vs user:1234
- Use an informal key schema
- "object-type:id" is a good idea, as in “user:1000” or “article:92617”

Article object creation

Redis HASH object: object id

- Title
- user who posted it
- Timestamp
- Votes received

- Object id:

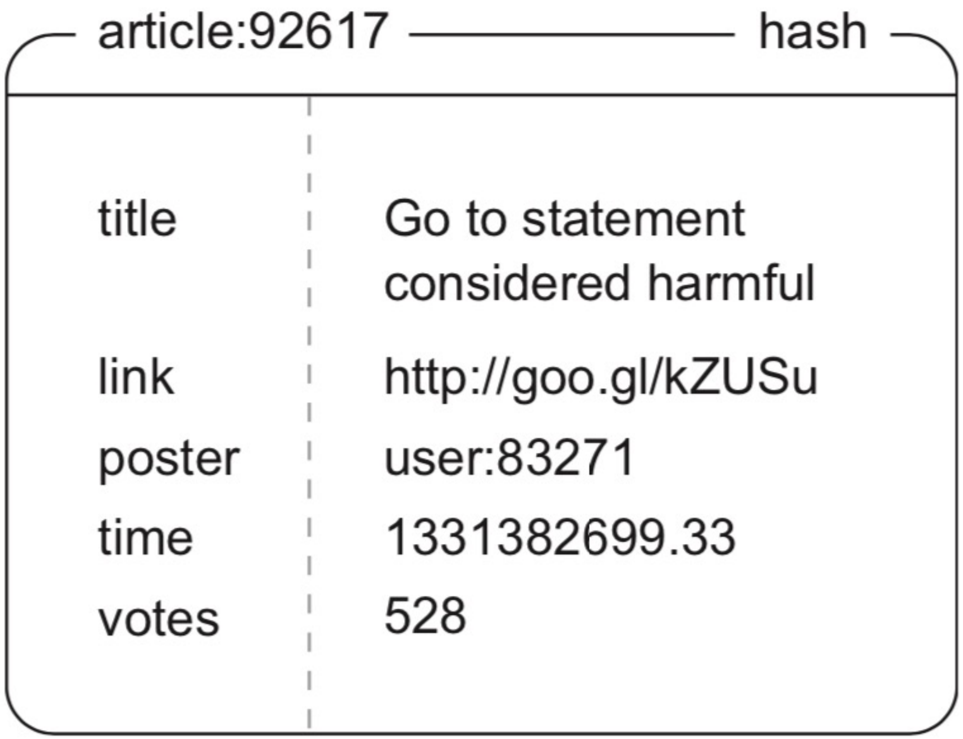
  <article:92617>

- Object information
  - Title
    - <title, *genomic analysis*>
  - User
    - <user, user-83721>
  - Time
    - <time, 1331382699>
  - Votes
    - <votes, 1>

Article information structure

Redis HASH – <article:number>

- Title
- Link to article
- user who posted it
- Timestamp
- Votes received



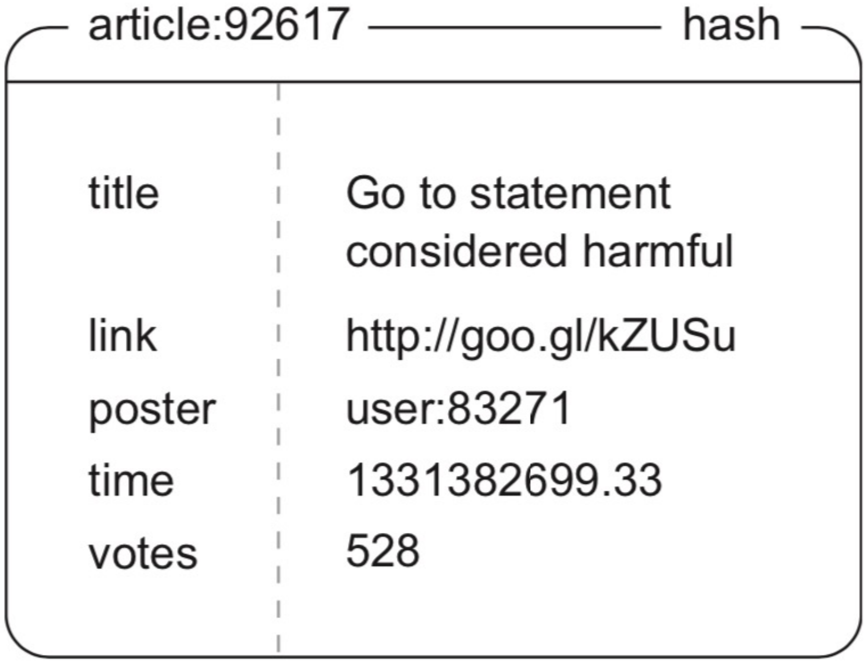
article:92617 hash

title	Go to statement considered harmful
link	http://goo.gl/kZUSu
poster	user:83271
time	1331382699.33
votes	528

Add a new article to database – Redis CLI

```
HSET <key> <attrib name> <attrib value>
HSET article:1 title “data analysis” user user:1 votes 1

HGET <key> <attrib name>
HGET article:1 title
“data analysis”

HMGET <key>
HMGET article:1
```

Hash operations using Python API



article:92617 hash

title	Go to statement considered harmful
link	http://goo.gl/kZUSu
poster	user:83271
time	1331382699.33
votes	528

```
myarticle = ‘article:92617’
```

- **create new object**:

```
hmset(myarticle,
        {‘title’: “data analysis”,
        ‘time’: “01/11/2021 10:10”,
        ‘votes’:1,} )
```

- **read all values for an object**:

```
hgetall(myarticle)
```

Add a new article to database - Python

```
now=round(time.time()*1000)
article_id="1234"
article="article:"+article_id
client.hset(article, mapping={
     "title": "data analysis",
     "time": now,
     "votes": 1,
})
```

Hash objects summary

- Good for keeping collections of individual elements of our data model
  like rows of our articles table
- Must create a key for each object: <article:123456>
- Store attributes with label keys: “title”: “data analysis”
  Multi attribute with json format
- Get values at anytime searching for object key

ZSETS – sorted lists of elements

global list of relevant objects

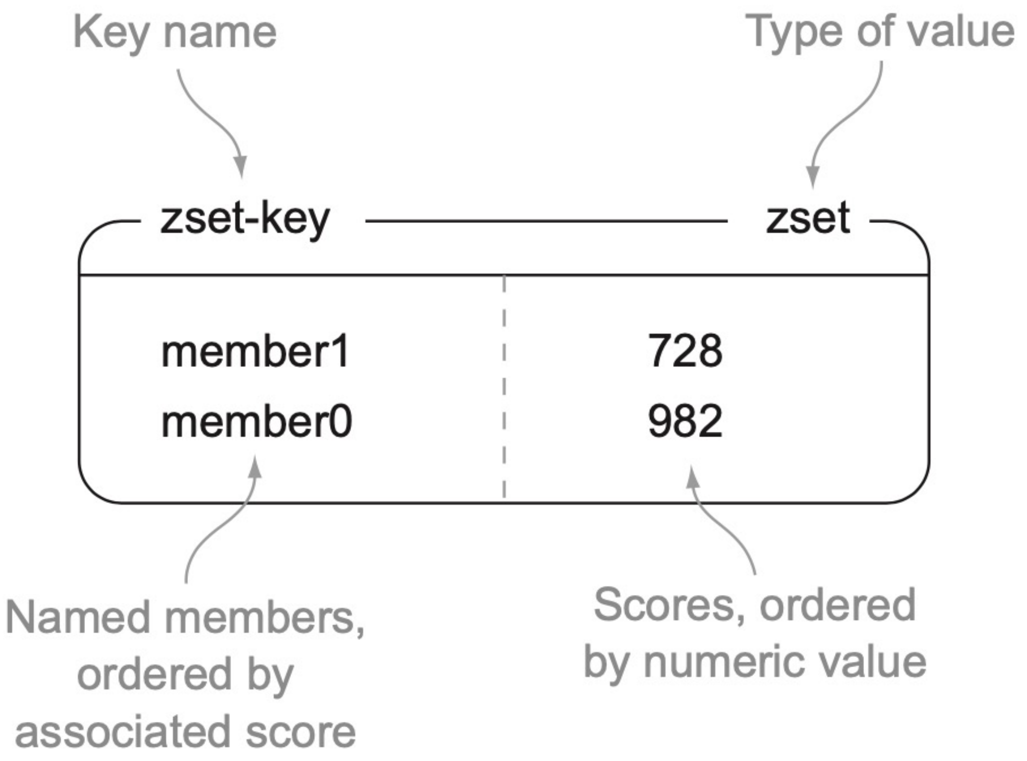
Article ranking

- Must keep a top list view of articles with its score
  - Ranking is a sorted collection of our articles
  - Article, score of article

- Get most ranked articles
  - by position: give me the top 10 highest score items
  - by ranking: all articles with score between (900 and 1000)

- use ZSET: sorted **set** of elements (by score)

Redis sorted sets ZSET



Key name

Type of value

zset-key — zset

member1	728
member0	982

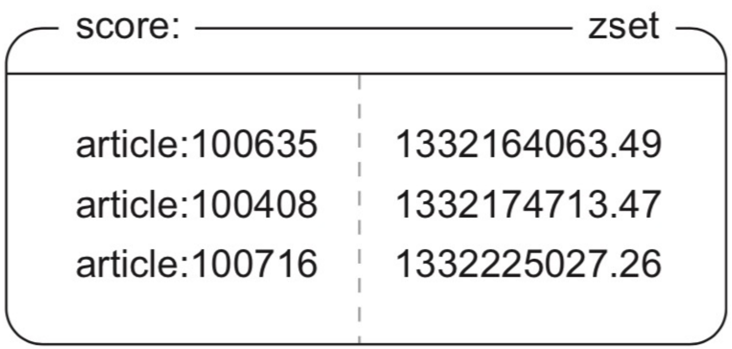
Named members, ordered by associated score

Scores, ordered by numeric value

Sorted set of articles with ZSET

Keep all articles sorted by score

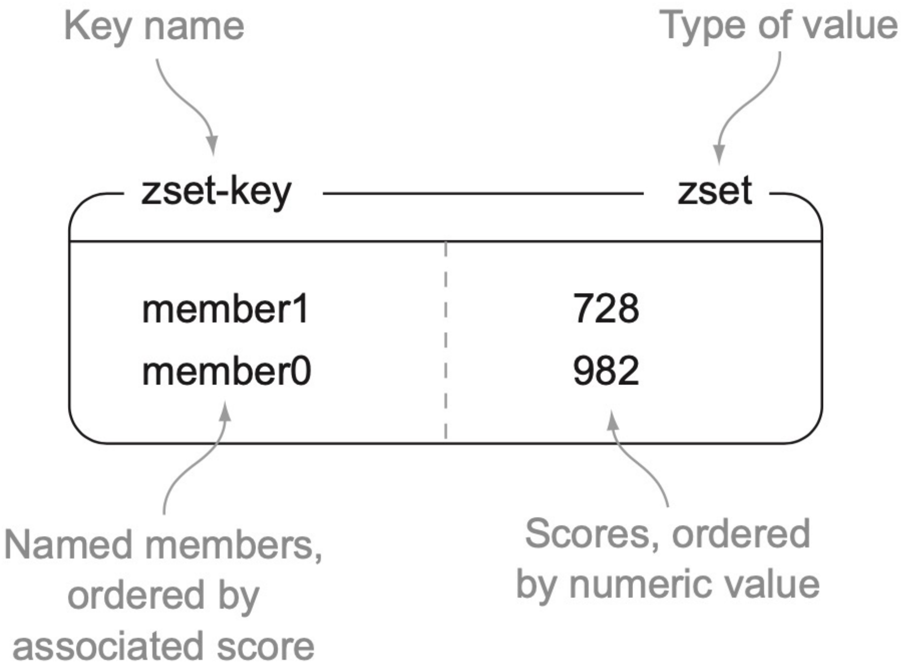
- ZSET name score:
- article id (key), article score (value)
  - article:100408, 1332065417.47



score:	zset
article:100635	1332164063.49
article:100408	1332174713.47
article:100716	1332225027.26

A score-ordered ZSET of articles

Redis sorted sets ZSET


Key name
Type of value
zset-key
zset
member1
728
member0
982
Named members, ordered by associated score
Scores, ordered by numeric value

Operations with ZSET

- ZADD: add a new item
- ZSCORE: get score from item
- ZRANGE: get TOP items
- ZRANGEBYSCORE: get items with some score

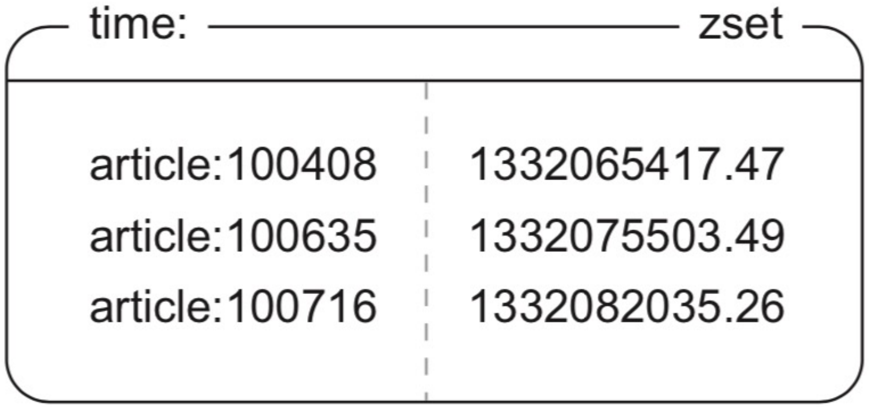
Article time of posting with ZSET

Keep the time of article posting

- ZSET name time:
- Article id, Linux timestamp
  - article:100635, 1332164063

Timestamp:

  - number of seconds since 1/1/1970



article:100408	1332065417.47
article:100635	1332075503.49
article:100716	1332082035.26

A time-ordered ZSET of articles

ZSET operations

```
new article score
ZADD <key of zset> <member-score member-id>
ZADD score: 500 article:1

get article score
ZSCORE <key of zset> <member-id>
ZSCORE score: article:1

get 10 first elements
ZRANGE <key> <first position> <last position>
ZRANGE score: 0 9
```

Add a new article to global score - Python

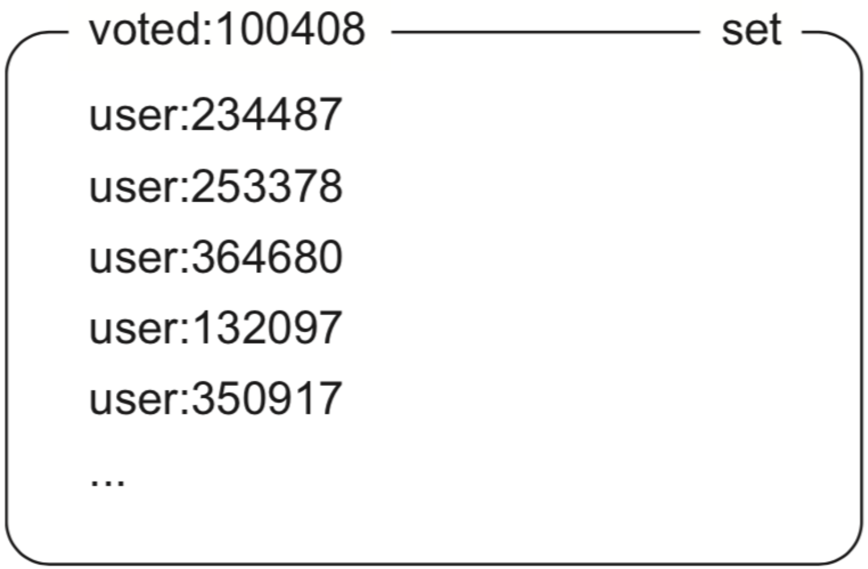
```
myarticle="article:1234"
votes=500

score_zset="score:"
client.zadd(score_zset, {myarticle : votes})
```

SETS: keep groups of unique elements

One element per SET, unique values only

Article voters structure with SET

- List of users who voted for an article
- Prevents users voting more than once for the same article
- One SET for each article
  - *voted:<article id>*



voted:100408

set

user:234487

user:253378

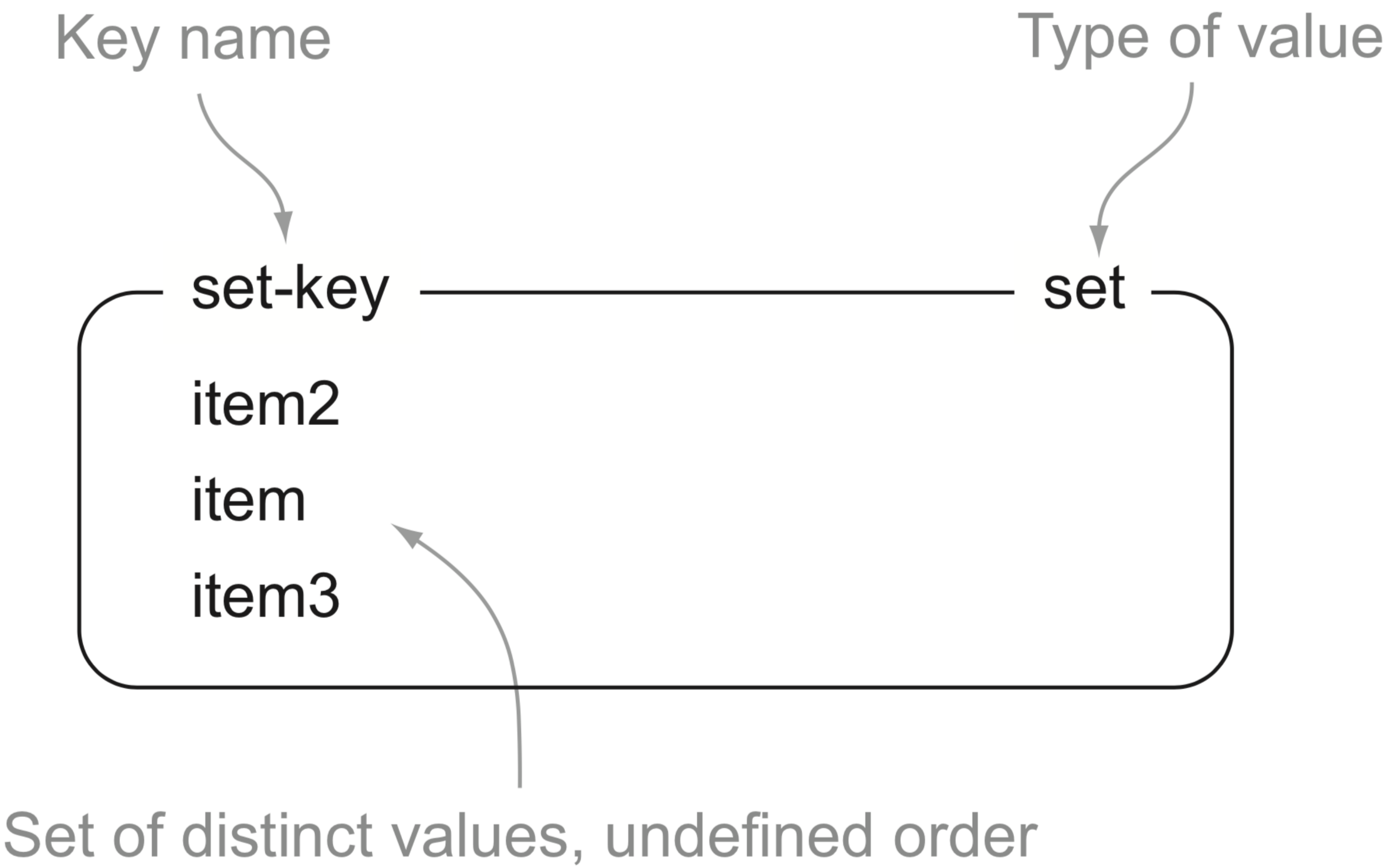
user:364680

user:132097

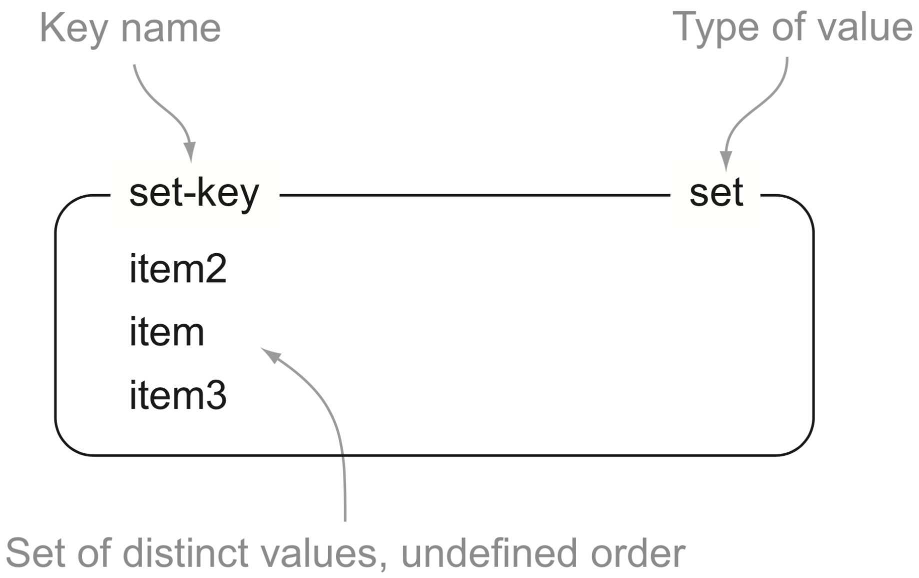
user:350917

...

Redis SET structure



Redis SET structure



SET common operations:

- SADD: Add a new item to SET
- SREM: Remove item
- SISMEMBER: check if item is in SET
- SMEMBERS: get all items from SET
- SINTER: intersection
- SCARD: cardinality

Add a new user to list of voters for an article

Add user:1234 to the list of article:1 voters

```
SADD <set key> <member-id>
SADD voted:1 user:1234
```

get all users who voted for article 1

SMEMBERS <set key>

```
SMEMBERS voted:1
```

check if user:1234 voted for article 1

SISMEMBER <key> <member-id>

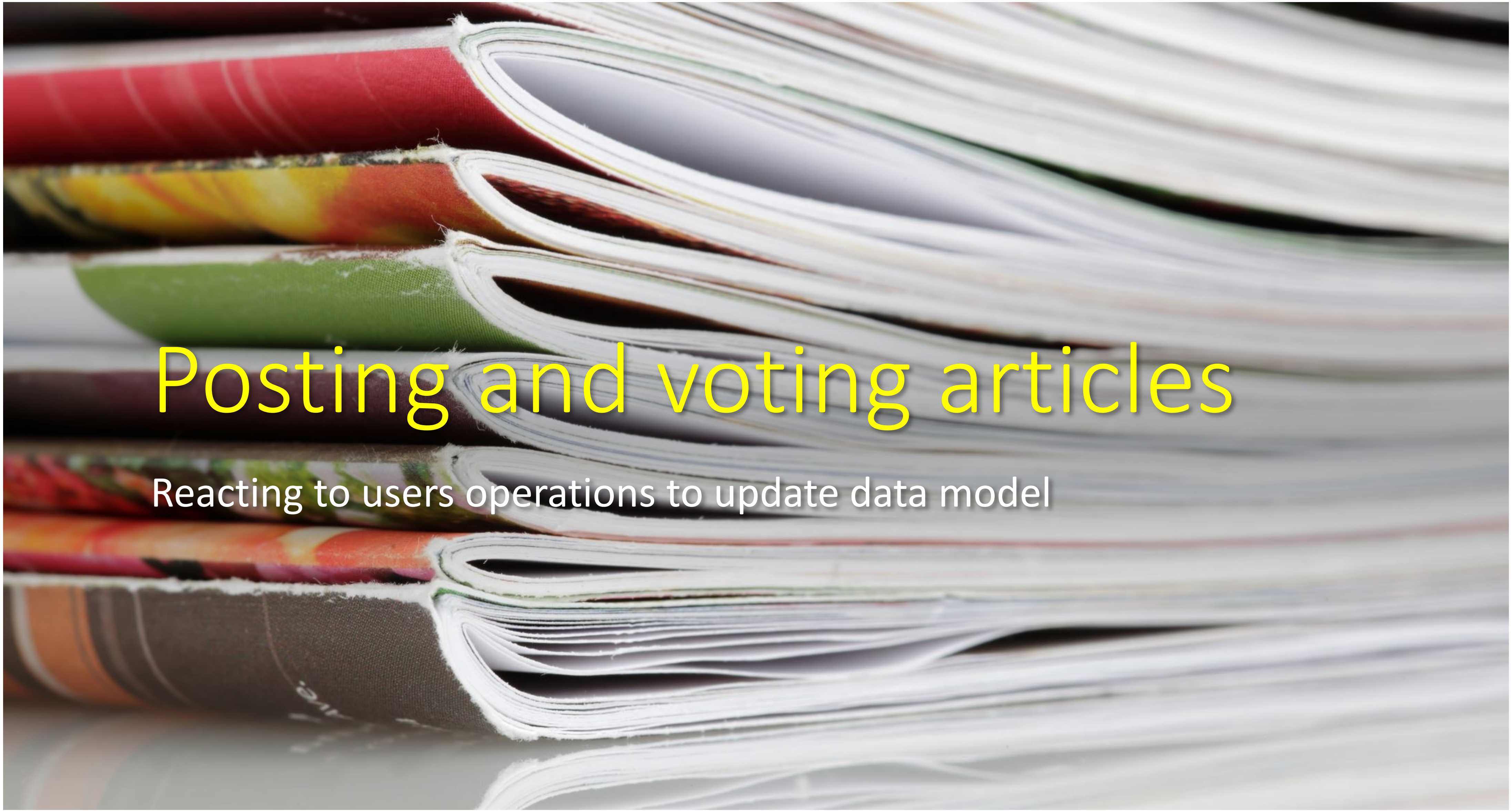
```
SISMEMBER voted:1 user:1234
```

Add a new user to list of voters for an article

```
myarticle= ‘article:’+article_id
voted = ‘voted:’+article_id
user = ‘user:1234’

sadd(voted, user)
```

Data structures for our Reddit example

- Articles: individual HASH per article (article:1234)
  - keep article details
- Article ranking: global ZSET for ranking and to store publish time
  - score:
  - keep global article ranking
- Voters: individual SET per article
  - voted:1
  - keep list of voters to limit one vote per user



Posting and voting articles

Reacting to users operations to update data model

Application events

- External
  - user creates a new article
  - user votes for an article
- Internal
  - produce current top 10 ranking articles
- How must our Redis design deal with these events?
  - Application reacts to external events then executes some code
  - Keep always a coherent view of the data model

User creates a new article

1. Build a new article-id
2. Store article attributes in new <article:id> object
3. Create voted SET to allow other users vote for this article
4. Add initial score and posting time for new article

New article – build an article ID

- User posts a new article
- build a new article id

```
counter = counter + 1
myarticle = ‘article:’+counter

article:1234
```

New article – build a new article hash object

```
HMSET  article:1234
        title:’this is the title’
        time:1111
        votes:1
```

New article – build a new article hash object

```
counter = counter + 1
myarticle = ‘article:’+counter
now=round(time.time())

client.hset(myarticle, mapping={
      ‘title’: ’this is the article title’,
      ‘time’: now,
      ‘votes’:1,
     })
```

New article: new voted SET

Add post user-id to a new voted: SET for the new article

```
SADD voted:1234 user:1
```

add article with initial score

- Add initial score to score: global ZSET

```
ZADD score:  500  article:1234
```

add article with initial score

- Add initial score to score: global ZSET

```
myarticle=‘article:’+counter
VOTE_SCORE= 500
zadd(‘score:’, {myarticle : now + VOTE_SCORE})
```

add article initial posting time

- Add initial posting time to time: global ZSET

```
ZADD time: 12345679  article:1234
```

add article initial posting time

- Add initial posting time to time: global ZSET

```
now=round(time.time())
zadd('time:', {myarticle: now})
```

new article operations summary

1. Build an article-id
   - Increment global article counter
2. Add article details into <article:id> HASH
   - HMSET article-id title: ‘this is my title’ votes:1})
3. Create voted SET
   - Add post user ID to the SET with SADD voted:article-id user id
   - give expiration date of one week with EXPIRE
4. Add initial score and posting time
   - ZADD score: article-id SCORE
   - ZADD time: article-id publication time

```
counter=counter+1
myarticle = ’article:”+counter
now = round(time.time())
client.hset(myarticle, mapping={
     ‘title’: title,
     ‘time’: now,
     ‘votes’:1,
})

voted = ‘voted:’+counter
client.sadd(voted, user)
client.expire(voted, ONE_WEEK_IN_SECONDS)

client.zadd(‘score:’, {myarticle : now+VOTE_SCORE})
client.zadd(‘time:’, {myarticle : now})
```

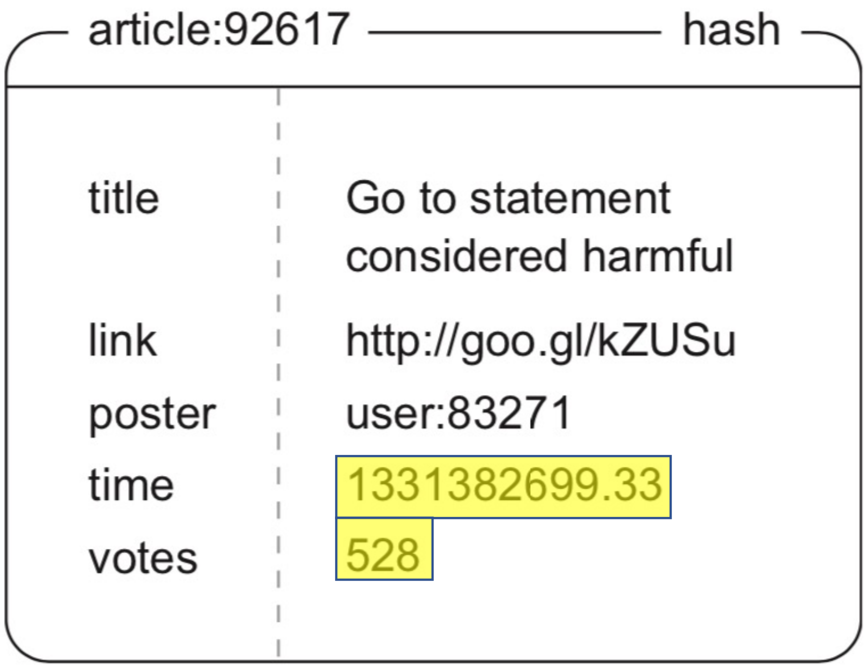
User wants to vote for an article

Steps to manage voting action:

1. Verify that article was posted within last week in article HASH
2. Add the user to article voted: SET

   Only one vote per user is allowed
3. Increment the score of the article in score: SET
4. Update vote count in article HASH

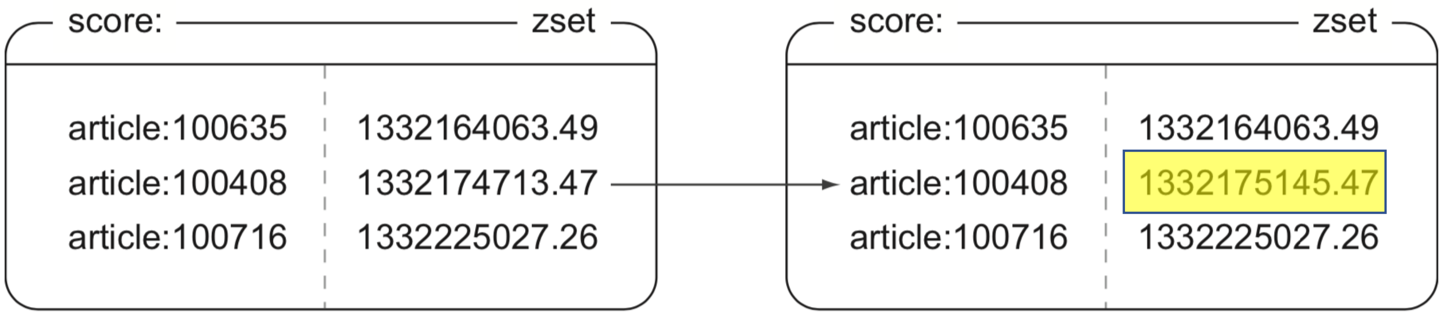
is article published within last week?
YES=add +1 to votes



article:92617 hash

title	Go to statement considered harmful
link	http://goo.gl/kZUSu
poster	user:83271
time	1331382699.33
votes	528

user:115423 votes for *article:100408*



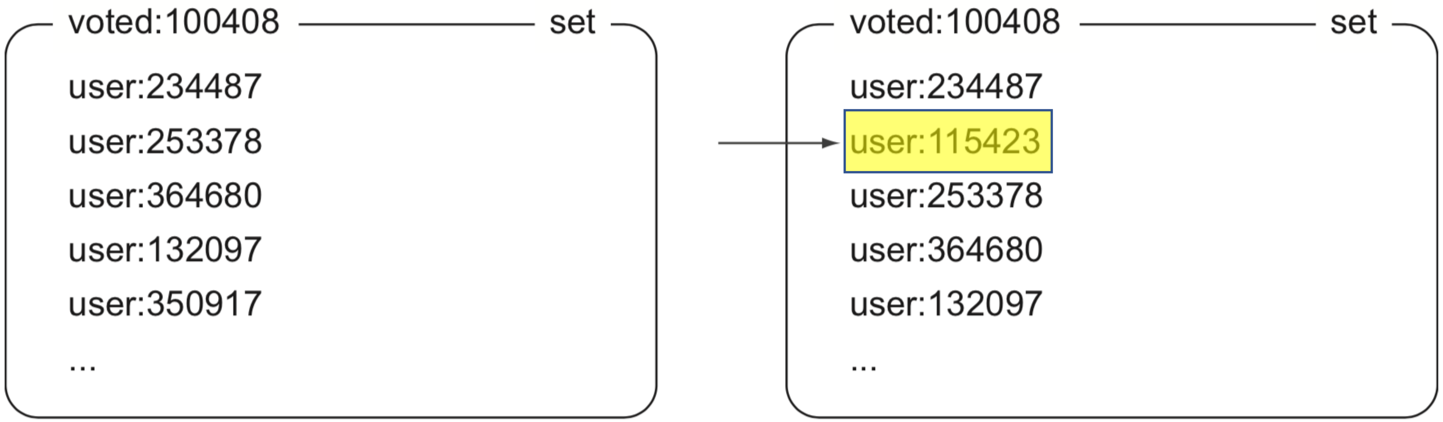
score: zset

article:100635	1332164063.49
article:100408	1332174713.47
article:100716	1332225027.26

score: zset

article:100635	1332164063.49
article:100408	1332175145.47
article:100716	1332225027.26

Article 100408 got a new vote, so its score was increased.



voted:100408 set

user:234487
user:253378
user:364680
user:132097
user:350917
...

voted:100408 set

user:234487
user:115423
user:253378
user:364680
user:132097
...

Since user 115423 voted on the article, they are added to the voted SET.

Why new score is not +1?

SCORE= number of seconds in a day (86.400) /
number of votes required (200) to last a full day

SCORE = 432

User wants to vote for an article

- Verify that article was posted within last week in article HASH
  - time:<article id> must be less than one week old
- Add the user to article voted SET
  - voted:<article id>, add <user id>
- Increment the score of the article in score SET
  - score:<article id>, = score + SCORE INCREMENT (432)
- Update vote count in article HASH
  - article:<article id>, votes = votes + 1

check if article is less than a week old

- get article creation time from <time:myarticle> ZSET
- get current time with time.time()
- creation time = ZSCORE(<time:myarticle>)
- IS (current time – creation time) < one week in seconds?
    - YES: article can receive votes

- Now we must check if user has not voted for this article before

check if user has voted for myarticle

- value=SISMEMBER voted:<myarticle> user
  - if value = 1 user has already voted
  - If value = 0

    add user to voters list
    - SADD(voted:<myarticle>, user)

    update article score
    - ZINCRBY(score:<myarticle>, 432)

    update vote count
    - HINCRBY(article:<myarticle>, votes, 1)

article vote implementation

```
ONE_WEEK_IN_SECONDS = 7 * 86400
VOTE_SCORE = 432
ctime=round(time.time())

cutoff = ctime - ONE_WEEK_IN_SECONDS
if conn.zscore('time:', article) > cutoff:
    article_id = article.partition(':')[-1]
    conn.sadd('voted:' + article_id, user):
        conn.zincrby('score:’, myarticle, VOTE_SCORE})
        conn.hincrby(myarticle, 'votes', 1)
```

Build TOP 10 articles by popularity

- Fetch current top scoring articles

- ZRANGE to fetch articles
- with highest decreasing score
- from position 0 to 9

```
ids = zrange(“score:”, 0, 9, REV)
```

Build TOP 10 articles by popularity

- Fetch most recent articles
- HGETALL to get info about each article

```
ids = zrange(“score:”, 0, 9, REV)
for id in ids:
     article_data = hgetall(id)
```

```
//GET TOP ARTICLES BY SCORE

ids = conn.zrange(‘score:’, 0, top-1, ‘REV’)
articles = []
for id in ids:
    article_data = conn.hgetall(id)
    article_data[‘id’] = id
    articles.append(article_data)
return articles
```

Next tutorial

build a simple web service with Redis: structures + functions