







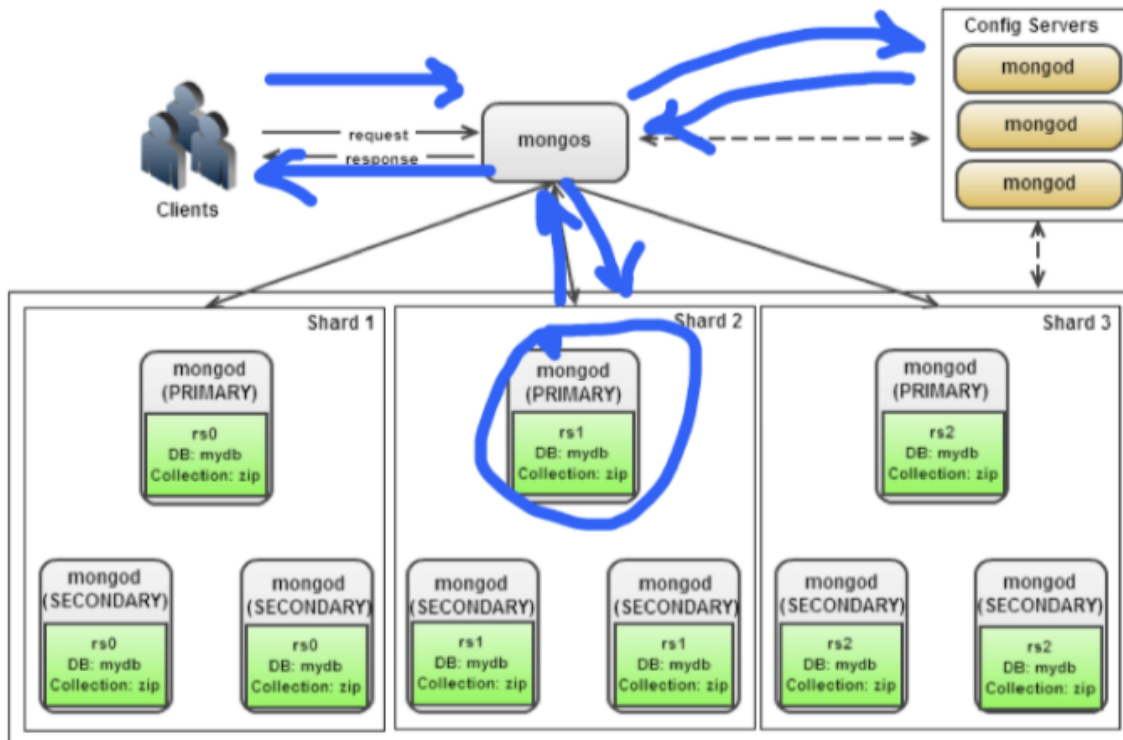
Class7-DB-Dec16

 Assign	
 Status	Completed
 Priority	
 Date Created	@December 16, 2021 7:53 PM
 Due Date	
 Property	

4. MongoDB

mongodb

- no-sql database
- document based datastore
- developed by C++, supports APIs in many computer language: java, python, ruby, perl...



There are three sharding, and each sharding stores partial of all the logical database. Each sharding has three replicas, one primary node and two secondary nodes.

Mongod: represent each instance of the mongoDB

Mongos works like API gateway. When request comes in, the mongos will check **which instance** (mongod) should be routed in, which checks the config Servers.

Config Servers contain all the meta data of all mongods (sharding and replica information). Mongos check config Servers and figure out which sharding should handle the request, and send request to corresponding sharding.

MongoDB has different components (?) mongod

Mongod: **database instance**

Mongos: sharding processes (look as a router), could be several mongos(multi-gateway)

- analogous to a database router
- process all the request

- decide how many and which mongods should receive the query

Mongos is not only routing all the requests, but also **cache some the request.**

e.g there is 1000 requests come in, mongos only can handle 500 requests, and it will cache other(remained) 500 requests

~~Mongo: interactive shell (manipulate MongoDB like Terminal)~~

Functionality of mongoDB

- Dynamic schema
 - Document data source, **every Document is allowed have different fields** (different elements in a JSON Object) BSON → binary JSON → BSON file
- document based database
- secondary indexes ?
- primary-second node with automated failover
- built in **horizontal scaling** via automated ranged-based partitioning of data(sharding)
- follows CP

5. Redis

Redis(remote directory server) in most situation, Redis used as cache

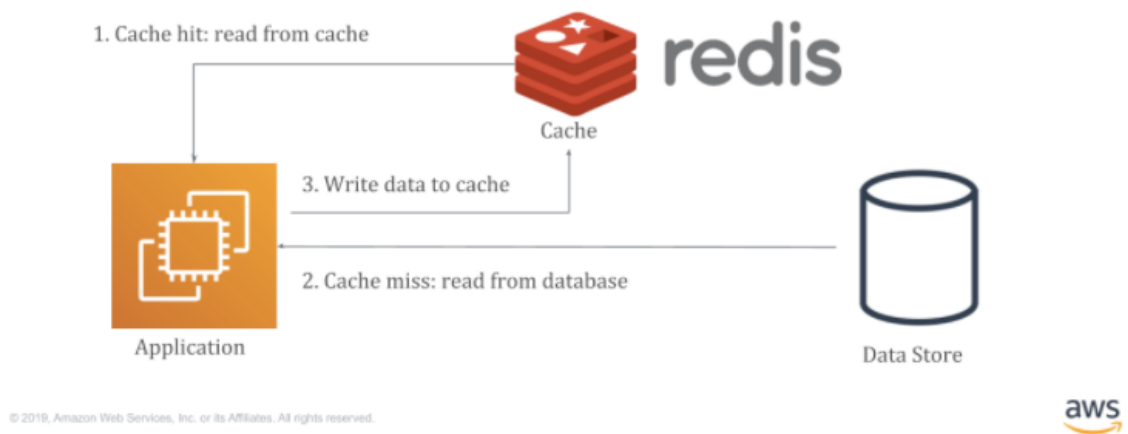
Redis is an **in-memory data structure store**, used as a distributed, in-memory key-value database, cache and message broker, with optional durability.

- in memory(use RAM to store data, which's faster than SSDs and Disks, but lose power, lose data)
- key value data store
- support different kinds of data structure
 - String
 - List (double LinkedList, message system)

- Sets (HashSet)
- Sorted Sets
 - each set has a score, redis will sort all set elements, set entry base on the score
- Hashes(HashMap, redis calls it Hashes)

Cache-Aside

- Modular
- Cache failure **is not** critical
- Data models can be different
- Cached data can get stale (TTLs)
- Code changes required



- cache hit, cache miss
 - cache miss: if redis has not required data, that's cache miss, and then redis will ask DB to get(popular) the data.
 - cache hit: find the data at redis, called cache hit

Why Redis is powerful?

Because support two kinds of **persistence mechanisms**

- RDB(redis database): the RDB persistence performs point-in-time snapshots of dataset at specific intervals
 - store snapshots at specific intervals

- AOF(append only file): The AOF persistence logs every write operation received by the server, what will be played again at server startup, reconstructing the original dataset
 - when the cache service down and restart, it will re-implement (replay) all the write operations from snapshots

start service → copy snapshot → run write operation again stored in AOF(logs)

LRU: Least Recently Used cache(I) — also OOP Question

allowing you to quickly identify which item hasn't been used for the longest amount of time.

how to write a LRU cache by LinkedHashMap

LinkedList + hashMap

Memcache

AWS: Elastic Cache

- two cache engine
 - **Memcache** (not single thread, auto ??, maintain the number of index)
 - **redis**(single thread)

Primary cache tech uses redis, and second memcache

Homework2-1:

- Memcache
- compare redis and memcache

AWS: Elastic Cache

redis	memcache
Persistence by RDB and AOF	Not persistence
Multiple data type: String, List...	Only String
Single thread	Muti-threads
Support Master-Slave replication	Not support any replication
Persistent data	Not use persistent data

Q: Why the redis is single thread, we still use it?

concepts only

Other usgae:

- cache
- distributed lock
- message queue(not suggest)
- store configuration information

6. SQL database vs no-sql database

sql	no-sql
relational database	non-relational database
pre-defined schema	dynamic schema
vertical scaling (scale up)	horizontal scaling (scale out)
ACID	CAP
not suited for hierarchical data store	suited for hierarchical data store

vertical scaling: more power(cpu, memory) in single machine

horizontal scaling: separate to multiple machine

SQL hope data in a single table → normalization rules

Hierarchical(分级的) data store is a tree-like data store

Homework2-2:

- vertical scaling vs horizontal scaling
- hierarchical data store
- BASE

BASE principle

7. index

indexing is a way to optimize the performance of a database by minimizing the number of the disk accesses required when a query is processed.

which for quick locate and access in DB

- clustered index - primary index
- non-clustered index - secondary index

cluster index: (come with DB, **when create DB first time**, DB engine will create index automatically, **only one cluster index in each table**)

default order is when insert data in table is by primary key

- defines the order in which data is physically stored
- only one clustered index per table

```
INSERT INTO student
```

```
VALUES
```

```
(6, 'Kate', 'Female', '1985-05-05', 500, 'Liverpool'),  
(2, 'Jon', 'Male', '1971-02-02', 545, 'Manchester'),  
(9, 'Wise', 'Male', '1987-04-16', 499, 'Manchester'),  
(3, 'Sara', 'Female', '1988-05-25', 600, 'Leeds'),  
(1, 'Jolly', 'Female', '1989-03-14', 500, 'London'),  
(4, 'Laura', 'Female', '1963-01-03', 400, 'Liverpool'),  
(7, 'Joseph', 'Male', '00982-02-02', 643, 'London'),  
(5, 'Alan', 'Male', '1993-10-25', 500, 'London'),  
(8, 'Mice', 'Male', '1998-01-01', 543, 'Liverpool'),  
(10, 'Elis', 'Female', '1996-03-08', 400, 'Leeds');
```

id	name	gender	DOB	total_score	city
1	Jolly	Female	1989-03-14 00:00:00	500	London
2	Jon	Male	1971-02-02 00:00:00	545	Manchester
3	Sara	Female	1988-05-25 00:00:00	600	Leeds
4	Laura	Female	1963-01-03 00:00:00	400	Liverpool
5	Alan	Male	1993-10-25 00:00:00	500	London
6	Kate	Female	1985-05-05 00:00:00	500	Liverpool
7	Joseph	Male	0982-02-02 00:00:00	643	London
8	Mice	Male	1998-01-01 00:00:00	543	Liverpool
9	Wise	Male	1987-04-16 00:00:00	499	Manchester
10	Elis	Female	1996-03-08 00:00:00	400	Leeds
NULL	NULL	NULL	NULL	NULL	NULL

non-cluster index: (create a virtual table/index)

- as many as non cluster index
- doesn't sort the physical data inside the table

like create an index based on **name** column

IX_tblStudent_Name Index Data

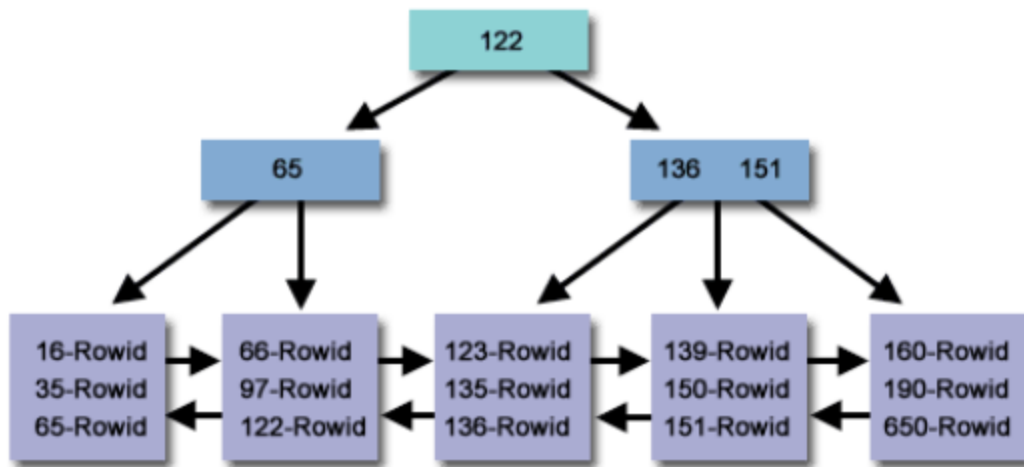
name	Row Address
Alan	Row Address
Ellis	Row Address
Jolly	Row Address
Jon	Row Address
Joseph	Row Address
Kate	Row Address
Laura	Row Address
Mice	Row Address
Sara	Row Address
Wise	Row Address

i.e follow the Binary tree to search a name, $O(\lg n)$

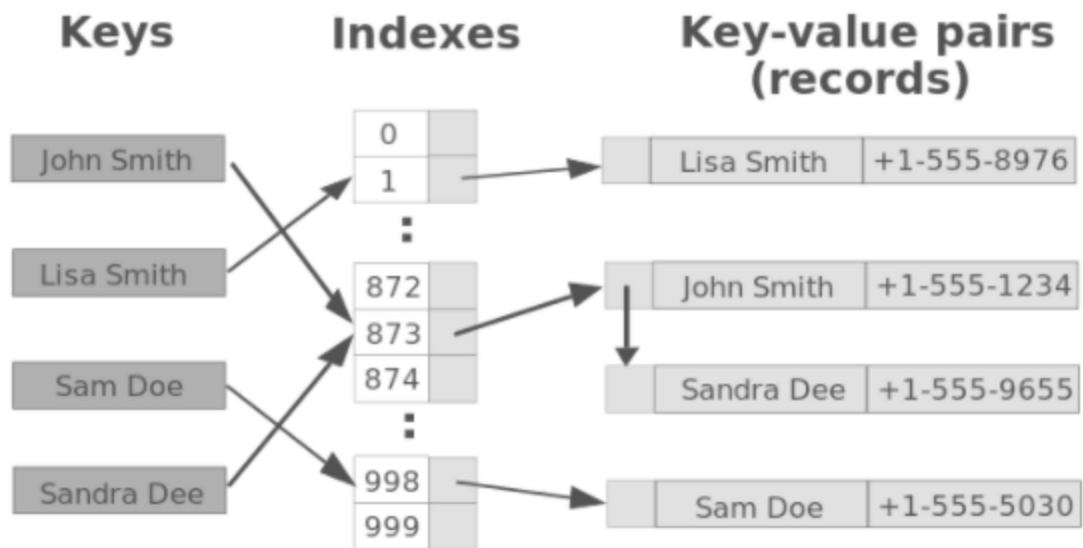
Row Address → like a reference to a row

index is data structure technic

- b tree(default)
- bitmap
- has table
- r tree
- ...



HashMap index



HashMap →

```
select *  
from table  
where name == john;
```

Range search → tree $O(\log n)$

```
select *  
from table  
where id <= 100 and id > 50  
50 - 100  $O(n)$ 
```

```
row id of 50  
row id of 100
```

8. SQL Tuning

SQL tuning

1. Using Execution Plan to identify the cause of slowness
 - a. Execution Plan: tool to find out which cause slowness(find out time cost of each components of query)
2. try to reduce joins, remove unused join and join conditions
3. using index to improve join
 - a. when we have to use join(index join algorithm)
4. use Union ALL instead of Union
 - a. Union will remove all duplicated data
 - b. Union ALL keep all original data
5. ~~use limit to do pagination~~
 - a. ~~pagination(页码): partial the result~~
6. View or stored procedure to improve the performance
7. select exact fields instead of *
8. ...

Homework2-3:

- view and stored procedure

application tuning

- check the db query - do the sql tuning
- DB connection, connection pool usage. change the pool size
 - Original: create and destroy connections are also time consuming → connection pool
 - **similar with Java thread pool**

- do the JVM tuning: tools → Jstack, JMap, Jconsole
 - cuz some time we create lots of useless objects → occupied heap area
- **server side**: CPU, Memory usage by using commands like top, ps
- code review to check if the code is not efficient
- check networking, firewall, load balancer