NoSQL Database

ACID Properties

- **Atomicity** – All of the work in a transaction completes or none of it completes
- **Consistency**– A transaction transforms the database from one consistent state to another consistent state.
- **Isolation** – The results of any changes made during a transaction are not visible until the transaction has committed.
- **Durability** – The results of a committed transaction survive failures

Why NoSQL?

RDBMS - Scalability Issues - Big Data

- Issues with scaling up when the dataset is just too big e.g. Big Data.
- Not designed to be distributed.

NoSQL provides a solution to the scalability issue as they can run on clusters or multi-node database solution.

Different approaches include:

- Master-slave
- Sharding

Master-Slave:

- All writes are written to the master. All reads are performed against the replicated slave databases
- Critical reads may be incorrect as writes may not have been propagated down
- Large data sets can pose problems as master needs to duplicate data

Sharding:

- Any DB distributed across multiple machines needs to know in what machine a piece of data is stored or must be stored
- A sharding system makes this decision for each row, using its key

BASE Properties

- Basically Available – Prioritizing availability than consistency.NoSQL databases will ensure availability of data by spreading and replicating it across the nodes of the database cluster.
- Soft State – Due to the lack of immediate consistency, NoSQL databases enforces its own consistency delegating that responsibility to developers and so the state of the system could change over time.
- Eventually Consistent – The system will become consistent once it stops receiving input.

CAP Theorem:

CAP theorem – At most two properties on three can be addressed

1. Consistency : Each client has the same view of the the data
2. Availability : Each client can always read and write
3. Partition tolerance : System works well across distributed physical networks

All NoSQL databases provide two of the three above properties but none guarantees all three. We have to choose the database according to our requirements.

Distinguishing Characteristics

- Large data volumes (Google’s “big data”)
- Scalable replication and distribution (Thousands of machines-worldwide)
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE
- Open source development

Types of NOSQL Databases:

- **Column Based:** Optimized for queries over large datasets, and store columns of data together, instead of rows.Each row can have different columns. E.g. Cassandra , Amazon DynamoDB
- **Document Based:** Pairs each key with a complex data structure known as a document like tree data structure consisting of maps and scalar values . E.g.: MongoDB
- **Key-Value Pair Based:** Every single item in the database is stored as an attribute name (or 'key'), together with its value. It is designed for processing dictionary,which is basically a collection of records having fields containing data . E.g.: CouchDB
- **Graph Based:** Those databases are used when data can be represented as graph, for example, social networks.E.g. : Neo4J, Infinite Graph

MongoDB:

- NoSQL database developed in C++. First public release in 2009
- It is a non-relational database, which features the richest and most like the relational database
- Supports complex data types: bjson data structures to store complex data types
- Powerful query language: it allows most functions like query in a single table of relational databases, and also supports index.

```
spartan@IMS-089MBA ~ % mongosh
Current Mongosh Log ID: 6448c40b07393d5712931a2e
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.8.0
Using MongoDB:          6.0.5
Using Mongosh:          1.8.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-04-24T15:31:46.013-07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2023-04-24T15:31:46.013-07:00: Soft rlimits for open file descriptors too low
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongorc.js will not be loaded.
  You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.js.
test>
```

InfluxDB:

- An open-source schemaless time series database with optional closed-sourced components developed by InfluxData.
- Written in Go programming language and it is optimized to handle time series data.
- SQL-like query language.
- Supports Sharding - Stores data in shard groups, which are organized and store data with timestamps that fall within a specific time interval.
- Uses its in-house built data structure, the Time Structured Merge Tree (TSM Tree).

```
[spartan@IMS-089MBA ~ % influx
Connected to http://localhost:8086 version 1.11.0
InfluxDB shell version: 1.11.0
```

TimeScaleDB:

- An open-source time series database developed by Timescale Inc.
- Written in C and extends PostgreSQL.
- TimescaleDB supports standard SQL queries and is a relational database.
- SQL functions and table structures provide support for time series data oriented towards storage, performance, and analysis facilities for data-at-scale.
- Data partitioning provides for improved query execution and performance when used for time-oriented applications.

```
effective_cache_size = 5888MB
maintenance_work_mem = 1005007kB
work_mem = 5025kB
timescaledb.max_background_workers = 16
max_worker_processes = 23
max_parallel_workers_per_gather = 2
max_parallel_workers = 4
wal_buffers = 16MB
min_wal_size = 512MB
default_statistics_target = 500
random_page_cost = 1.1
checkpoint_completion_target = 0.9
max_locks_per_transaction = 64
autovacuum_max_workers = 10
autovacuum_naptime = 10
effective_io_concurrency = 200
timescaledb.last_tuned = '2023-03-18T21:35:07Z'
timescaledb.last_tuned_version = '0.14.3'
Saving changes to: /var/lib/postgresql/data/postgresql.conf

waiting for server to shut down...2023-03-18 21:35:07.163 UTC [103] LOG:  received fast shutdown request
.2023-03-18 21:35:07.164 UTC [103] LOG:  aborting any active transactions
2023-03-18 21:35:07.164 UTC [143] FATAL:  terminating connection due to administrator command
2023-03-18 21:35:07.165 UTC [117] FATAL:  terminating connection due to administrator command
2023-03-18 21:35:07.173 UTC [103] LOG:  background worker "logical replication launcher" (PID 118) exited with exit code 1
2023-03-18 21:35:07.177 UTC [103] LOG:  background worker "TimescaleDB Background Worker Launcher" (PID 117) exited with exit code 1
2023-03-18 21:35:07.181 UTC [103] LOG:  background worker "TimescaleDB Background Worker Scheduler" (PID 143) exited with exit code 1
2023-03-18 21:35:07.182 UTC [109] LOG:  shutting down
2023-03-18 21:35:07.266 UTC [103] LOG:  database system is shut down
 done
server stopped

PostgreSQL init process complete; ready for start up.

2023-03-18 21:35:07.440 UTC [1] LOG:  starting PostgreSQL 12.14 on x86_64-pc-linux-musl, compiled by gcc (Alpine 12.2.1_git20220924-r4) 12.2.1 20220924, 64-bit
2023-03-18 21:35:07.442 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2023-03-18 21:35:07.443 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2023-03-18 21:35:07.444 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-03-18 21:35:07.543 UTC [176] LOG:  database system was shut down at 2023-03-18 21:35:07 UTC
2023-03-18 21:35:07.562 UTC [1] LOG:  database system is ready to accept connections
2023-03-18 21:35:07.580 UTC [186] LOG:  TimescaleDB background worker launcher connected to shared catalogs
2023-03-18 21:45:26.951 UTC [1] LOG:  received fast shutdown request
2023-03-18 21:45:26.953 UTC [1] LOG:  aborting any active transactions
2023-03-18 21:45:26.957 UTC [192] FATAL:  terminating connection due to administrator command
2023-03-18 21:45:26.966 UTC [186] FATAL:  terminating connection due to administrator command
2023-03-18 21:45:26.978 UTC [1] LOG:  background worker "logical replication launcher" (PID 187) exited with exit code 1
2023-03-18 21:45:26.980 UTC [1] LOG:  background worker "TimescaleDB Background Worker Launcher" (PID 186) exited with exit code 1
2023-03-18 21:45:26.984 UTC [1] LOG:  background worker "TimescaleDB Background Worker Scheduler" (PID 192) exited with exit code 1
2023-03-18 21:45:26.985 UTC [178] LOG:  shutting down
2023-03-18 21:45:27.081 UTC [1] LOG:  database system is shut down
spartan@IMS-089MBA ~ % docker container start ba1dc221bcf656798f06818c787163acb23f0847c3c9d41d51601225d5f425d0

ba1dc221bcf656798f06818c787163acb23f0847c3c9d41d51601225d5f425d0
spartan@IMS-089MBA ~ %
spartan@IMS-089MBA ~ %
spartan@IMS-089MBA ~ % docker container start project_timescaledb

project_timescaledb
spartan@IMS-089MBA ~ % psql -U postgres -h localhost
psql (15.2, server 14.7 (Homebrew))
Type "help" for help.

postgres=# 
```

```
spartan@IMS-089MBA ~ % brew services
Name              Status  User    File
cassandra         started spartan ~/Library/LaunchAgents/homebrew.mxcl.cassandra.plist
emacs             none
hbase             none
influxdb@1        started spartan ~/Library/LaunchAgents/homebrew.mxcl.influxdb@1.plist
mongodb-community started spartan ~/Library/LaunchAgents/homebrew.mxcl.mongodb-community.plist
postgresql@14     started spartan ~/Library/LaunchAgents/homebrew.mxcl.postgresql@14.plist
questdb           started spartan ~/Library/LaunchAgents/homebrew.mxcl.questdb.plist
unbound           none
spartan@IMS-089MBA ~ %
```

Questdb

- An open-source time series database
- A relational database optimized for speed and low latency
- Supports SQL querying and indexing
- Provides full ACID compliance and transaction support
- Provides columnar storage format and has a built-in time series extension that provides time-series functionality

```
spartan@IMS-089MBA ~ % questdb start


  ___                  _   ____  ____
 / _ \ _   _  ___  ___| |_|  _ \| __ )
| | | | | | |/ _ \/ __| __| | | |  _ \
| |_| | |_| |  __/\__ \ |_| |_| | |_) |
 \__\_\\__,_|\___||___/\__|____/|____/
                        www.questdb.io

JAVA: /opt/homebrew/opt/openjdk@17/libexec/openjdk.jdk/Contents/Home/bin/java
spartan@IMS-089MBA ~ % Reading log configuration from /opt/homebrew/var/questdb/conf/log.conf
spartan@IMS-089MBA ~ % brew services
```

Time Series Database Suite:

- A collection of GO programs
- Use cases : CPU-only, DevOps and IOT
- Benchmark read and write performance of various databases.
- Various Databases Supported
- Various Queries supported (depending on the use case)

Data Generation

- Data is generated randomly, but it’s deterministic if we supply the same PRNG value for each database.
- The variables defining the generated data are :

1. Use case (CPU-only, devops, or iot)
2. PRNG seed for deterministic generation E.g:123
3. The number of devices E.g: 4000 (This will determine the size of the dataset)
4. A start time E.g: 2023-04-01T00:00:00Z
5. An end time E.g: 2023-04-02T00:00:00Z
6. Time between each reading E.g:10s
7. Target Database E.g: mongo (name determined for MongoDB)

Example:

tsbs_generate_data --use-case="cpu-only" --seed=123 --scale=100 --timestamp-start="2023-04-01T00:00:00Z" --timestamp-end="2023-04-02T00:00:00Z" --log-interval="10s" --format="mongo" | gzip > /Users/spartan/tmp/mongo-data.gz

Output : mongo-data zip file . 780 mb of data (Zipped to save space)

Mongo DB vs MySQL time series data format comparison

```
test> show dbs
admin       40.00 KiB
benchmark   11.21 MiB
config      72.00 KiB
local       72.00 KiB
test> use benchmark
switched to db benchmark
benchmark> show collections
point_data                    [time-series]
system.buckets.point_data
system.views
benchmark> db.point_data.find()
[
  {
    time: ISODate("2023-04-01T00:00:00.000Z"),
    tags: {
      arch: 'x86',
      datacenter: 'eu-central-1a',
      hostname: 'host_0',
      os: 'Ubuntu15.10',
      rack: '6',
      region: 'eu-central-1',
      service: '19',
      service_environment: 'test',
      service_version: '1',
      team: 'SF'
    },
    usage_irq: 63,
    usage_guest: 80,
    usage_idle: 24,
    usage_softirq: 6,
    usage_nice: 61,
    usage_system: 2,
    usage_guest_nice: 38,
    usage_user: 58,
    usage_iowait: 22,
    measurement: 'cpu',
    usage_steal: 44,
    _id: ObjectId("642f25ea78c015e22abcf619")
  },
  {
    time: ISODate("2023-04-01T00:00:00.000Z"),
    tags: {
      arch: 'x64',
      datacenter: 'us-west-1a',
      hostname: 'host_1',
      os: 'Ubuntu15.10',
      rack: '41',
      region: 'us-west-1',
      service: '9',
      service_environment: 'staging',
      service_version: '1',
      team: 'NYC'
    },
    usage_idle: 53,
    usage_guest: 53,
    usage_nice: 87,
    usage_softirq: 54,
    usage_guest_nice: 74,
    usage_system: 11,
    usage_irq: 20,
    usage_user: 84,
    usage_iowait: 29,
```

MongoDB – JSON like format

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	time	usage_irq	usage_guest	usage_idle	usage_softirq	usage_nice	usage_system	usage_quest_nice	usage_user	usage_iowait	measurement	usage_steal	id:ObjectId
2	2023-04-01T00:00:00.000Z	63	80	24	6	61	2	38	58	22	cpu	44	642f25ea78c015e22abcf619
3	2023-04-01T00:00:00.000Z	20	53	53	54	87	11	74	84	29	cpu	77	642f25ea78c015e22abcf61a
4													

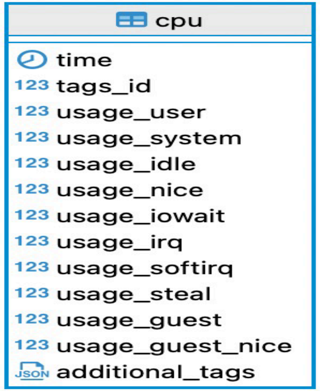
MYSQL -- Table format

PostgreSQL Data Evaluation

```
postgres=# \c benchmark
psql (15.2, server 14.7 (Homebrew))
You are now connected to database "benchmark" as user "postgres".
benchmark=# \dt
         List of relations
 Schema | Name | Type  |  Owner
--------+------+-------+----------
 public | cpu  | table | postgres
 public | tags | table | postgres
(2 rows)

benchmark=# SELECT * FROM cpu;
          time          | tags_id | usage_user | usage_system | usage_idle | usage_nice | usage_iowait | usage_irq | usage_softirq | usage_steal | usage_guest | usage_guest_nice | additional_tags
------------------------+---------+------------+--------------+------------+------------+--------------+-----------+---------------+-------------+-------------+------------------+-----------------
 2023-03-31 17:00:00-07 |       1 |         58 |            2 |         24 |         61 |           22 |        63 |             6 |          44 |          80 |               38 |
 2023-03-31 17:00:00-07 |       2 |         84 |           11 |         53 |         87 |           29 |        20 |            54 |          77 |          53 |               74 |
 2023-03-31 17:00:00-07 |       3 |         29 |           48 |          5 |         63 |           17 |        52 |            60 |          49 |          93 |                1 |
 2023-03-31 17:00:00-07 |       4 |          8 |           21 |         89 |         78 |           30 |        81 |            33 |          24 |          24 |               82 |
 2023-03-31 17:00:00-07 |       5 |          2 |           26 |         64 |          6 |           38 |        20 |            71 |          19 |          40 |               54 |
 2023-03-31 17:00:00-07 |       6 |         76 |           40 |         63 |          7 |           81 |        20 |            29 |          55 |          20 |               15 |
 2023-03-31 17:00:00-07 |       7 |         44 |           70 |         20 |         67 |           65 |        11 |             7 |          92 |           0 |               31 |
 2023-03-31 17:00:00-07 |       8 |         92 |           35 |         99 |          9 |           31 |         1 |             2 |          24 |          96 |               69 |
 2023-03-31 17:00:00-07 |       9 |         21 |           77 |         90 |         83 |           41 |        84 |            26 |          60 |          43 |               36 |
 2023-03-31 17:00:00-07 |      10 |         90 |            0 |         81 |         28 |           25 |        44 |             8 |          89 |          11 |               76 |
 2023-03-31 17:00:00-07 |      11 |         76 |           85 |         24 |          0 |           44 |        88 |            90 |          40 |          72 |               63 |
 2023-03-31 17:00:00-07 |      12 |         80 |           11 |         50 |         72 |           52 |        18 |            68 |          88 |          54 |               50 |
 2023-03-31 17:00:00-07 |      13 |         64 |           81 |         55 |         54 |           89 |        81 |            69 |          33 |          53 |               25 |
 2023-03-31 17:00:00-07 |      14 |         48 |            0 |         64 |         91 |           13 |        88 |            79 |          41 |          48 |                4 |
```

```
benchmark=# SELECT * FROM tags;
 id | hostname |     region      |    datacenter     | rack |      os        | arch | team | service | service_version | service_environment
----+----------+-----------------+-------------------+------+----------------+------+------+---------+-----------------+---------------------
  1 | host_0   | eu-central-1    | eu-central-1a     | 6    | Ubuntu15.10    | x86  | SF   | 19      | 1               | test
  2 | host_1   | us-west-1       | us-west-1a        | 41   | Ubuntu15.10    | x64  | NYC  | 9       | 1               | staging
  3 | host_2   | sa-east-1       | sa-east-1a        | 89   | Ubuntu16.04LTS | x86  | LON  | 13      | 0               | staging
  4 | host_3   | us-west-2       | us-west-2b        | 12   | Ubuntu15.10    | x64  | CHI  | 18      | 1               | production
  5 | host_4   | sa-east-1       | sa-east-1c        | 74   | Ubuntu16.10    | x86  | SF   | 7       | 0               | staging
  6 | host_5   | us-west-1       | us-west-1b        | 18   | Ubuntu16.10    | x64  | CHI  | 14      | 0               | staging
  7 | host_6   | ap-southeast-1  | ap-southeast-1b   | 49   | Ubuntu16.10    | x86  | CHI  | 7       | 0               | staging
  8 | host_7   | eu-west-1       | eu-west-1c        | 44   | Ubuntu16.10    | x64  | LON  | 7       | 1               | test
  9 | host_8   | eu-west-1       | eu-west-1a        | 17   | Ubuntu16.04LTS | x64  | LON  | 2       | 0               | test
 10 | host_9   | ap-southeast-2  | ap-southeast-2a   | 0    | Ubuntu16.04LTS | x86  | CHI  | 18      | 0               | production
 11 | host_10  | sa-east-1       | sa-east-1a        | 95   | Ubuntu16.10    | x64  | LON  | 8       | 0               | staging
 12 | host_11  | us-west-2       | us-west-2b        | 66   | Ubuntu15.10    | x64  | NYC  | 6       | 1               | production
 13 | host_12  | eu-central-1    | eu-central-1a     | 79   | Ubuntu16.10    | x86  | CHI  | 6       | 1               | production
 14 | host_13  | eu-west-1       | eu-west-1a        | 79   | Ubuntu16.10    | x64  | CHI  | 19      | 1               | staging
 15 | host_14  | us-west-1       | us-west-1b        | 8    | Ubuntu15.10    | x64  | LON  | 3       | 0               | production
 16 | host_15  | ap-southeast-2  | ap-southeast-2a   | 67   | Ubuntu16.10    | x86  | LON  | 11      | 1               | production
 17 | host_16  | ap-southeast-2  | ap-southeast-2a   | 51   | Ubuntu16.10    | x64  | NYC  | 6       | 1               | staging
 18 | host_17  | ap-southeast-1  | ap-southeast-1b   | 79   | Ubuntu16.10    | x86  | NYC  | 6       | 0               | production
 19 | host_18  | ap-southeast-1  | ap-southeast-1b   | 0    | Ubuntu15.10    | x86  | LON  | 19      | 1               | test
 20 | host_19  | ap-northeast-1  | ap-northeast-1a   | 70   | Ubuntu16.04LTS | x64  | CHI  | 19      | 0               | staging
```



Query Generation

- The variables used for generating the queries to be benchmarked against the generated data:

As the parameters should match the generated data , use case ,prng ,number of devices and the start date should be kept same. The end date should be kept one second more than the date of the generated date.For my example ,

1. An end time E.g : 2023-04-02T00:00:01Z
2. The number of queries to generate. E.g : 1000
3. The type of query E.g : single-groupby-1-1-1

Example:

```
tsbs_generate_queries --use-case=“cpu-only” --seed=123 --scale=100 \
    --timestamp-start="2023-04-01T00:00:00Z" \
    --timestamp-end="2023-04-02T00:00:01Z" \
    --queries=1000 --query-type="single-groupby-1-1-1" --format=“mongo” \ | gzip > /Users/spartan/tmp/mongo-queries-breakdown-single-groupby-1-1-1.gz
```

Output : mongo-queries zip file . 4 kb of data (Zipped to save space)

Data Loading

The data file generated in the data_generate step can be used to load data as the data is generated in the format supported by the database. For e.g. : MongoDB data file comes in BSON format.
The target database should be installed and configured properly and should be started before performing any steps.

Step 1: Generate a config yaml file for a particular database with default properties

E.g.: tsbs_load config --target=mongo --data-source=FILE

Step 2: Run the tsbs_load with the generated config file

E.g.: tsbs_load load mongo --config=./config.yaml

Query Execution:

This is the last step. So assuming that the database is setup properly , data and query is generated and the data is loaded in the target database , we can use the tsbs_run_queries program to successfully run the target queries with the generated data for a particular database.

If I have zip the generated query ,then i can gzip while running the load command

E.g. :

cat /tmp/queries/mongo-queries-breakdown-single-groupby-1-1-1.gz | \ gunzip | tsbs_run_queries_mongo --workers=8