DateTime & Pooling

“Simplicity, patience, compassion.
These three are your greatest treasures…”
- Lao Tzu

RONIN™
ENGINEER

Outline

1. Datetime
2. Connection Pool
3. Thread Pool

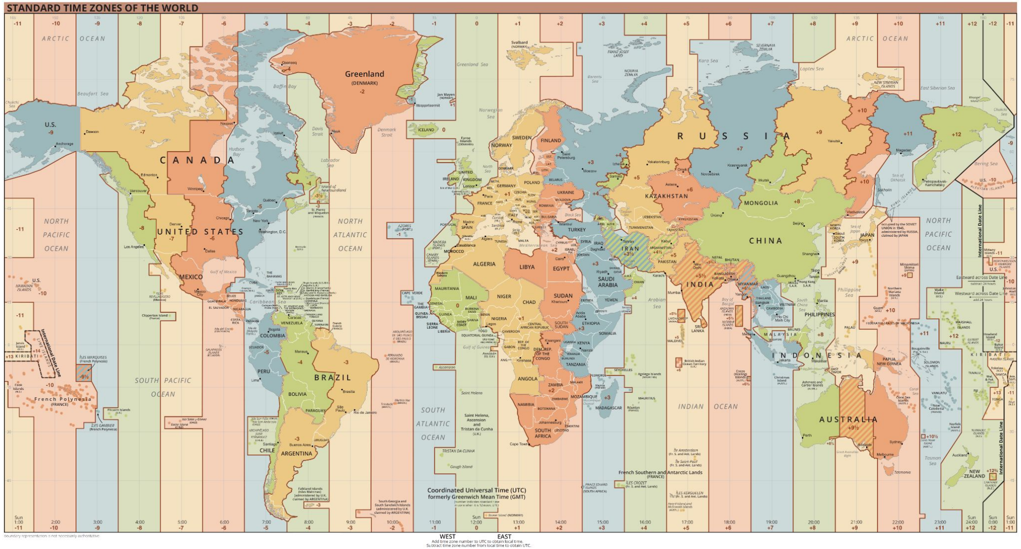
1. Datetime

1.1. How do computers store time?

- **UNIX timestamp**: the number of seconds that have elapsed since 1970-01-01 at 00:00:00 UTC
- Example:
  - Timestamp in seconds: 1695718262
  - Timestamp in milliseconds: 1695718262342
- Facts:
  - In older 32bit systems, the signed integer will **overflow in 2038**
  - Why 1970? The Unix OS was created in the late 1960s and early 1970s. Unix engineers arbitrarily selected 1970

1.2. Timezone

- Why do we need time zones?
- Time zones maintain logical order and regulate day and night across the globe
- UTC = GMT = Time zone offset of 0



1.3. Datetime Format

- **ISO 8601**: 2023-09-25T16:20:52+07:00, 2023-09-25T09:20:52Z
    - Z = +00:00
- Vietnam: 26/09/2023
- RFC 2822: Mon, 25 Sep 2023 09:16:58 +0000
- Javascript: YYYY-MM-DDTHH:mm:ss.sssZ
- Java: Tue Sep 26 16:05:37 ICT 2023 (+0700)
- RFC 3339
- **Why choose YYYY-MM-DDThh:mm:ss+TZ as ISO format?**

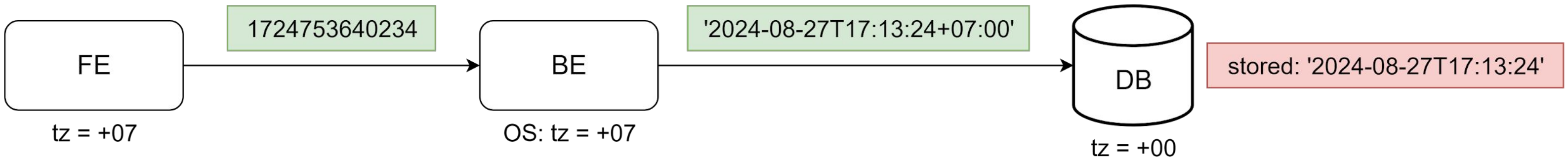
1.4. Exercise

- **ISO 8601**
- Time in Hanoi: 2023-09-26T07:15:52+07:00 → Time in Tokyo (+09), Paris (+01), California (-08)?

1.5. When to use DateTime, Timestamp?

- **Timestamp: to record a (more or less) fixed point in time.**
    - Example: created_at
- **Datetime: time can be set and changed arbitrarily.**
    - Example: schedule time for appointments

1.5. Timezone in System



- **Backend, DB uses UTC time zone. Recommend to store TZ**
- Frontend uses local datetime (show timezone) and send TZ if needed
- **Be careful: JVM timezone != OS timezone → set JVM timezone by using env var**

- FE insert: '2024-08-27T17:13:25.123+07:00'
- BE return: '2024-08-27T17:13:25.000+07:00'
- Why?

1.6. Precision

- FE insert: '2024-08-27T17:13:25.123+07:00' to MySQL DB
- BE return: '2024-08-27T17:13:25.000+07:00'
- Why?

- Because the precision of time data type in MySQL is in second by default.
- Cause: Data type: Datetime
- **Use the fractional digits** → accuracy
  - datetime(3)
  - timestamp(3)
- Timestamp in Postgres use 6 fractional digits by default.

1.6. How to Work with Datetime Data Type?

- **MySQL datetime data types does not store time zone info**
  → Recommended to store the time zone info.
- For example in Postgres:
  - Column \`started_at\`: timestamptz(3)
  - Column \`tz\`: smallint
  - TZ: UTC (+00)

1.6. Best Practices

- **ISO 8601**
- **Backend, DB uses UTC time zone**
- Frontend uses local datetime (show timezone)
- DB store as timestamp if possible
- Store time zone
- **MySQL: use a fractional digitals → accuracy**
- **JVM timezone != OS timezone**

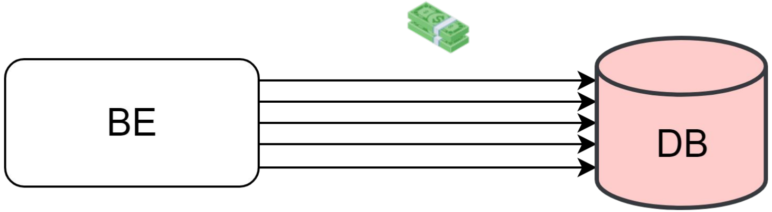
Be careful with:

- **Time zone**
- **Datetime format**
- **Precision: Fractional digitals**
- Một số case ma giáo khác

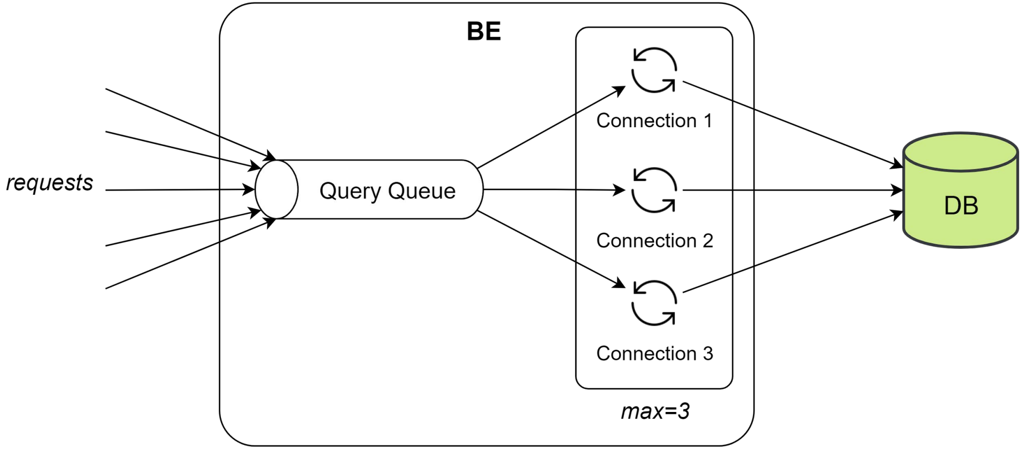
2. Connection Pool

2.1. Problem

- The native way: for each query, client creates a new connection to DB
- Problem:
  - Costly
    - **CPU to establish connections**
    - **Time-consuming**
    - Memory to keep connections
    - File Descriptor: ID of connections
    - Time to create and free up connections
  - If there is a high traffic, high number of concurrent connection to DB
    → we can lose DB and BE app.



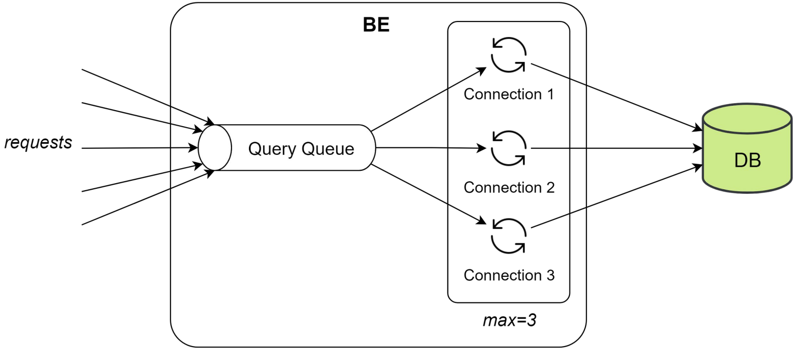
2.2. Solution: Connection Pool



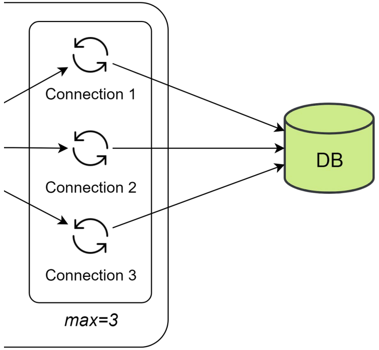
- When the query is complete, **instead of terminating the connection, the pooler places the connection back in the pool** so that it can potentially be reused by a subsequent query.
- If there is **no idle connection, requests will be enqueue**, wait for an available connection.
- Not every implementation of connection pool is same → choose the right one.

2.3. Connection Pool Configuration

- Question: How many connections in a client connection pool?
- Answer:
    - **Pool sizing is ultimately very specific to deployments.**
    - We **have to tune**.
    - For example, systems with a mix of long running transactions and very short transactions are the most difficult to tune.

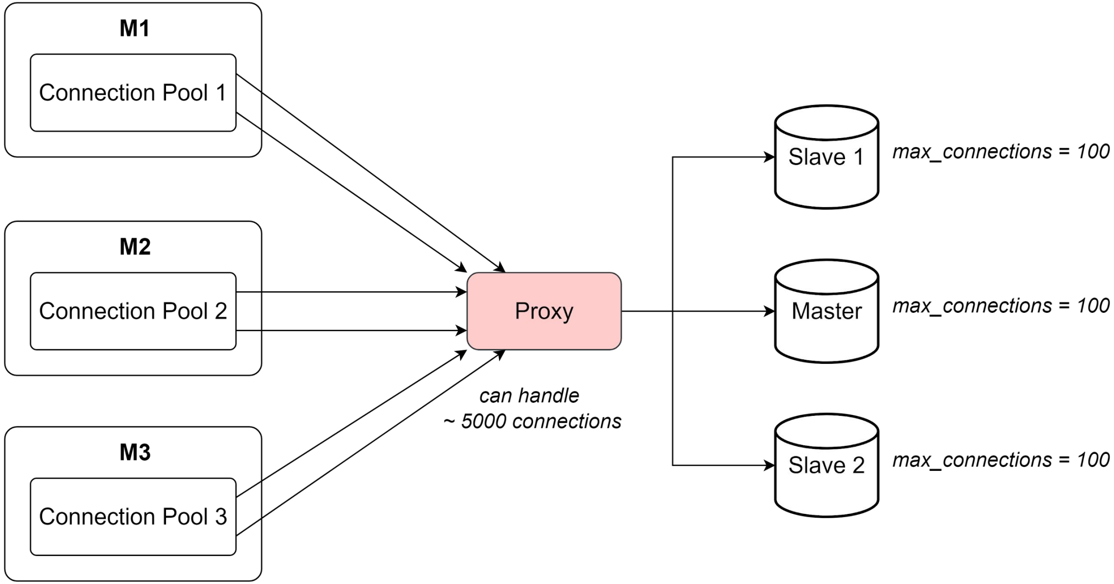


2.4. Tuning on Client Side



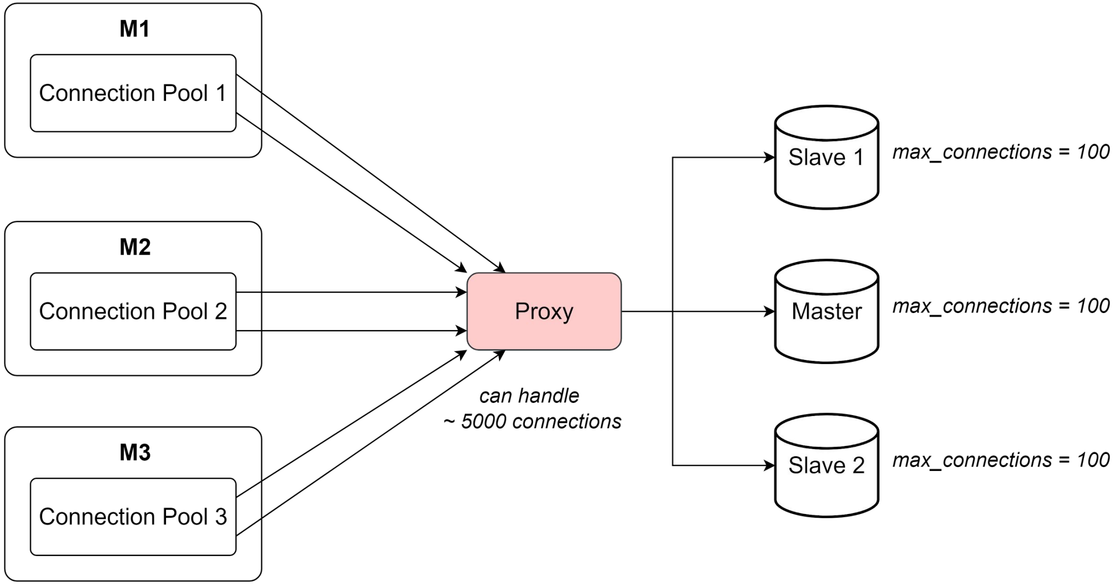
- Start with:
  Pool Size = Number of Core * 2 + Effective Spindle Count
  - Spindle Count:
    - 0: if active data is fully cached
    - 1: for SSD generally
    - > 1: for HDD, number of spindle disks
- Second Try:
  Pool Size = T x (C - 1) + 1
  - T: number of threads
  - C: the max number of concurrent connections held by a single thread.
- Third Try: the value between the first try and the second try.
- Fourth Try: Increase gradually from the second try.

2.5. Proxy



- Proxy routes, load balance requests to master and slaves.
- Proxy is connection pooler that can handle high number of concurrent connections.
  → more difficult to tune the internal pool size.

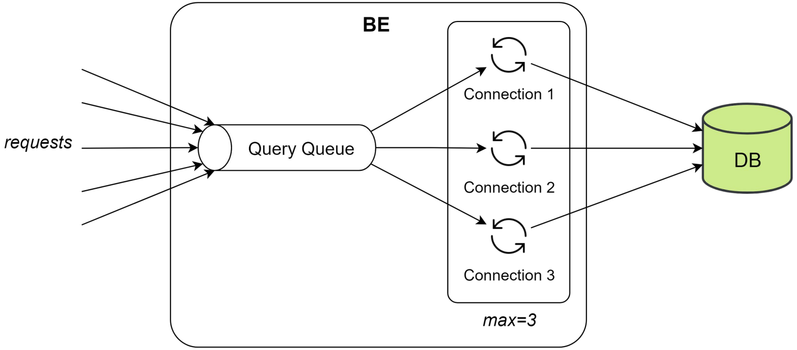
2.5. External Connection Pooler



- Note: Pick a right pooling mode

2.6. Recommended Configuration

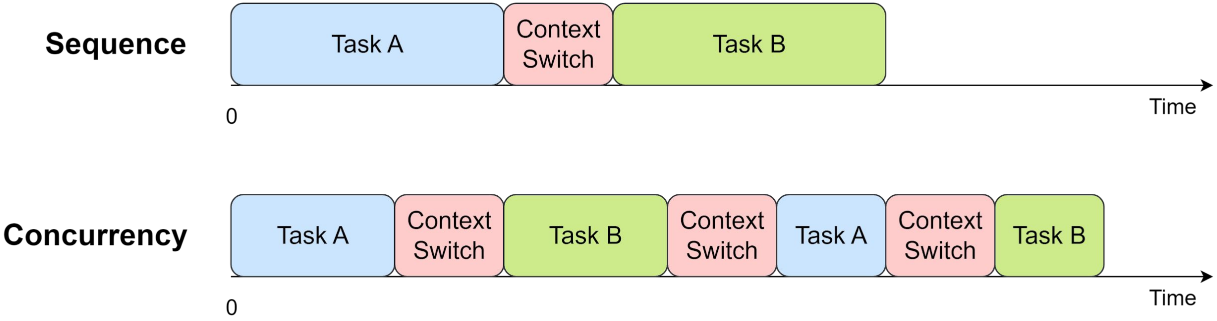
- Min of connections: around 10
- Max of connections: around 20 - 30



3. Thread Pool

2.1. Principle

- For one core, executing A and B **sequentially will always be faster than** executing A and B **concurrently** through time-slicing.



2.2. Pool Size

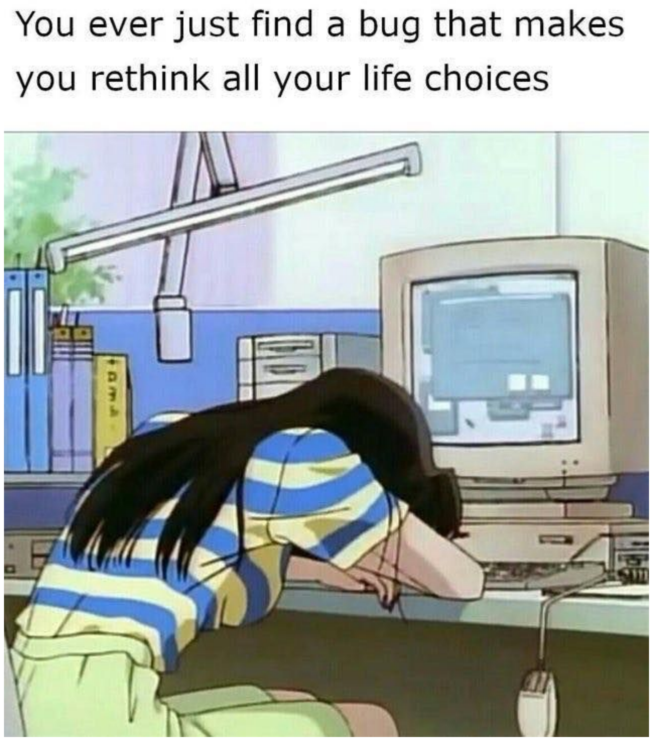
- Case 1: CPU-Bound Tasks
  - Recommend: n + 1
- Case 2: IO-Bound Tasks
  - Recommend: n * 2
- Case 3: Generic
  - n * (1 + average waiting time / average working time)
- n: the number of cores

Recap

- Datetime:
  - Time zone should be stored
  - Precision
- Connection Pool
  - Pool sizing is ultimately very specific to deployments.
- Thread Pool:
  - CPU-Bound Task vs IO-Bound Task

Homework

- Implement API Search Flights


You ever just find a bug that makes you rethink all your life choices

References

- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/13_IOSystems.html
- https://highscalability.com/big-list-of-20-common-bottlenecks/

Thank you 🙏

RONIN™
ENGINEER