1. Spark cluster load data từ database như nào?

- Nếu file nhỏ thì load bình thường không cần chia partition.

- Nếu file lớn như bảng admissions thì phải chia thành partition để tận dụng khả năng load song song của spark cluster. Ví dụ:

In [ ]:
```
            "admissions": {
                "partition_column": "hadm_id",
                "partition_bounds": (20000000, 30000000),
                "num_partitions": 10,
                "fetch_size": "50000"
            },
```

- Giải thích:
  - patition_column: cột để chia partition.
  - Khoảng giá trị min/max của cột hadm_id
  - num_partitions: số partition được seperate ra

- Cụ thể:
  - Như ví dụ trên, cột hadm_id có miền giá trị nằm trong khoảng (20 triệu - 30 triệu) => Range = 10 triệu.
  - Chia thành 10 partition thì mỗi partition sẽ tương ứng với khoảng 100000 dòng.
    - Partition 1: 20 triệu <= hadm_id < 21 triệu
    - Partition 2: 21 triệu <= hadm_id < 22 triệu
    - ...
    - Partition 10: 29 triệu <= hadm_id <= 30 triệu
  - Sau đó spark sẽ sinh ra 10 query song song kiểu:
    - SELECT * FROM admissions WHERE hadm_id >= 20000000 AND hadm_id < 21000000;
    - SELECT * FROM admissions WHERE hadm_id >= 21000000 AND hadm_id < 22000000;
    - ...

    -> Mỗi query được coi là 1 task và được gắn cho 1 core trong 1 spark worker node. Core này sẽ đảm nhiệm vai trò hoàn thành task này.

2. Nếu chỉ có 3 spark worker mà có đến 10 partition thì sao?

- Đầu tiên, cần biết mỗi spark worker có bao nhiêu core. Giả sử, mỗi worker có 2 cores thì tổng số core sẽ là 6.
- 10 partition sẽ được chia thành 10 task.

=> **Spark cluster sẽ phân phối như sau:**

- Lập lịch để thực thi 10 task này: 6 tasks đầu phân cho 6 cores thực thi trước -> sau đó, cái nào xong trước thì sẽ được phân task cho thực hiện tiếp đến khi hết task thì thôi.

3. Nên chia số lượng partition như nào cho hợp lý:

- Nếu bảng quá nhỏ không cần chia task.

- Nếu bảng lớn (tính bằng GB trở lên):
    - Số lượng partition (khuyến nghị) = 3-5 x số core. Ví dụ: có 6 core thì nên chia ít nhất là 18 tasks. Lý do: Để Spark có thể lập lịch, thực hiện từ 3-5 lượt là đẹp.

  ! Không nên chia ra quá nhiều partition, vì:
    - Công sức thằng spark-master nó quản lý, phân phối task cũng quá tội.
    - Số lượt đọc/ghi nhiều hơn (đọc/ghi rất tốn time)
    - 1 số trường hợp cần tách ra rồi ghép lại (map/reduce) thì cũng phức tạp, tốn tài nguyên hơn.

  ! Nhưng cũng không nên chia ra quá ít partition, vì:
    - Sẽ lãng phí thời gian và năng lực tính toán. Ví dụ: Có 6 cores nhưng mà chỉ có 2 partitions, thì tức là có 2 cores làm việc thôi còn 4 cores sẽ ngồi chơi (như vậy sẽ không tận dụng được 4 thằng này để làm giảm thời gian)

4. Postgres nhận cùng lúc 6-10 query (hoặc nhiều hơn) từ các spark worker thì nó có bị bottle neck không?

- PostgreSQL không biết gì về Spark, nó chỉ thấy có 10 client connections (từ Spark executors) gửi query song song -> Nên nó sẽ tạo 10 luồng để đáp ứng nhu cầu của 10 connection này **(miễn là máy đủ tài nguyên)**.

- Nếu không đủ tài nguyên thì các query sẽ tranh nhau CPU/RAM/I/O... (cái này trong môn Vi xử lý ở đại học có dạy, lúc nào đọc lại)

👉 **Nói nôm na: song song ở Spark → song song ở PostgreSQL, miễn là server chịu nổi.**

5. Cách chọn partitionColumn như nào?

- **Không phải cột nào cũng chọn làm partitionColumn được, cần ít nhất 3 tiêu chí:**
  - **Nên là dạng Numeric hoặc Date/Timestamp**: vì cần chia khoảng.
  - **Phân bố đều là tốt nhất (ít Null càng tốt) tránh partition bị lệch**. Ví dụ: Giả sử, cột hadm_id ở trên đa số bản ghi có hadm_id nằm trong khoảng 25 - 26 triệu, các khoảng còn lại có rất ít bản ghi. Thì sẽ có 1 thằng cores được giao task đọc các bản ghi có hadm_id nằm trong khoảng 25-26 triệu sẽ phải đọc cật lực, đọc rất lâu. Trong khi các thằng cores khác làm xong việc từ lâu, ngồi chơi rồi => Không tối ưu.
  - **Ít lặp giá trị**: Thử tưởng tượng dùng 1 cột flag chỉ có giá trị 0, 1 để chia partition đã thấy nó dở hơi rồi. Chả chia ra được bao nhiêu partition cả.

=> Thường chọn các cột ID, hoặc các cột liên quan đến Date/Time

6. Mỗi partition cũng tương ứng với 1 cái file .parquet nhỏ đúng không?

- **Đúng luôn :)))** Nếu có 10 partition -> 10 task được gán cho Spark -> Spark xử lý 1 task (hay 1 partition) xong thì sẽ ghi luôn nó thành 1 file .parquet nhỏ.
- **Tóm lại: số partition = số parquet (kiểu có bảng admissions chia thành 10 partitions thì đọc/ghi 1 hồi thì sẽ tạo ra 10 file .parquet**

```
/data/admissions_parquet/
 ├── part-00000-xxxx.snappy.parquet
 ├── part-00001-xxxx.snappy.parquet
 ├── part-00002-xxxx.snappy.parquet
 ...
 ├── part-00009-xxxx.snappy.parquet
 └── _SUCCESS
```

=> **Cái việc chia partition theo khoảng rồi ghi thành parquet như này có 1 lợi ích siêu to lớn. Đó là, các file .parquet được tạo ra bản thân nó cũng đã được partition.**

- Ví dụ, nếu có cần 1 bản ghi có hadm_id nằm trong khoảng 21-22 triệu thì chỉ cần đọc đúng file part-00001-xxxx-.snapy.parquet lên và tìm trong đó là được. Tiết kiệm rất nhiều tài nguyên máy (I/O/RAM/Bus), lại còn siêu tiết kiệm thời gian.
- Về phần làm sao biết được file .parquet này là chứa dữ liệu tương ứng với khoảng giá trị nào, thì yên tâm metadata của file parquet to nó có lưu lại thông tin min/max value của mỗi khoảng rồi.

7. Còn fetch_size thì sao, nó là gì và dựa vào đâu để định giá trị cho nó?

- **Postgres không thể đọc data từ disk rồi chuyển cho Spark luôn được**, mà nó cần phải load lên RAM trước, xong mới chuyển sang cho Spark.

- **Vì sao lại phải load lên RAM trước?** Lý do là, nó cần phải áp dụng filter, join, index scan,... kiểu như là phải lọc dữ liệu theo điều kiện where lên RAM trước, rồi dữ liệu đó mới stream sang cho Spark được.

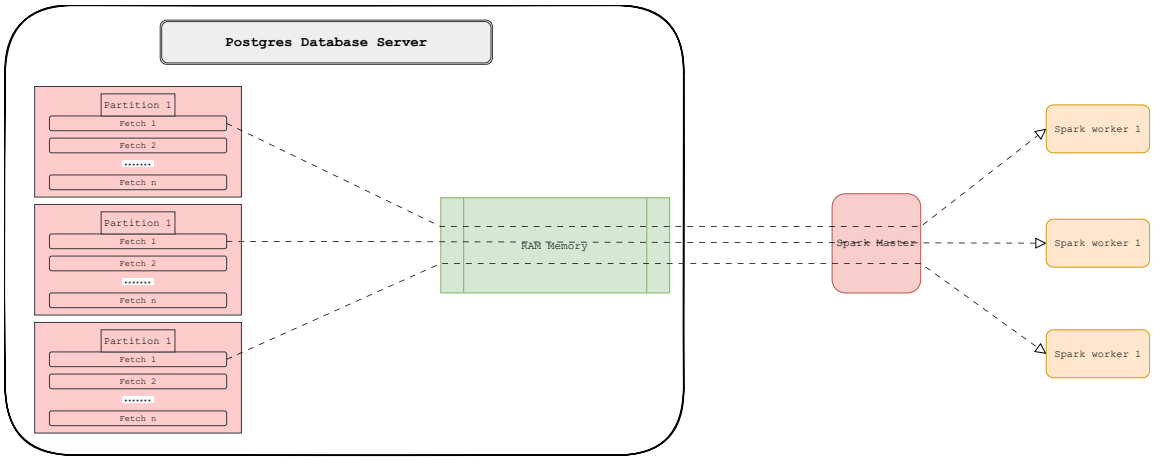
- **Vấn đề xảy là ở đây là**: Đôi khi partition khá to, to hơn cả RAM **mức RAM có thể xử lý tốt**, do đó, cần phải chia thành đơn vị nhỏ hơn là các FETCH để load từ DISK lên RAM rồi stream sang cho Spark thông qua các connection.

! Lưu ý: Fetch_size không tính bằng byte mà tính bằng số record, ví du: fetch_size = 10000. Tức là mỗi fetch sẽ gồm 10000 dòng (records)

- Spark worker **không chờ đến khi gom đủ fetch để thành 1 partition rồi mới xử lý**, mà mỗi Fetch nó nhận được thì nó **xử lý ngay lập tức**.
- Thế nên, về bản chất, là các Spark Worker Core xử lý từng fetch, chứ không phải xử lý cả partition cùng lúc.
- Partition là **giới hạn về mặt logic** thôi, nó giúp đảm bảo:
  - 2 cores khác nhau không được load các fetch của cùng 1 Partition.
  - Mỗi cores xử lý 1 Partition rồi lưu thành .parquet thì .parquet mới giữ được cái khoảng [min, max] tương ứng với khoảng [min, max] của partition ban đầu => Từ đó, mới tìm kiếm được nhanh hơn.
  - Cố định được size mỗi Partition, Partition khai báo gồm những bản ghi nào thì chắc chắn chỉ lấy những bản ghi đó. Còn fetch thì kích thước mỗi fetch hên xui, tuy là có khai báo fetch_size = 10000, nhưng có khi fetch chỉ gồm 9800 dòng, nó tùy thuộc vào RAM khả dụng của Postgres Server. **Tóm lại, fetch_size chỉ là số dòng tối đa của 1 fetch, còn kích thước thì mỗi fetch lại khác nhau.**

Hình ảnh ví dụ:

- 1 Table chia làm 3 Partitions
- Mỗi Partition chia làm nhiều Fetch
- Các Fetch của các Partition khác nhau được load lên RAM, rồi Spark Master điều phối các Fetch này tới các Spark Worker để xử lý (**quá trình này là song song - tức là 3 fetch được load lên RAM cùng lúc và cũng được xử lý cùng lúc như trong hình**)


Postgres Database Server
Partition 1
Fetch 1
Fetch 2
.....
Fetch n
Partition 1
Fetch 1
Fetch 2
.....
Fetch n
Partition 1
Fetch 1
Fetch 2
.....
Fetch n
RAM Memory
Spark Master
Spark worker 1
Spark worker 1
Spark worker 1