Performance Improvement of Post-ETL in OMOP CDM

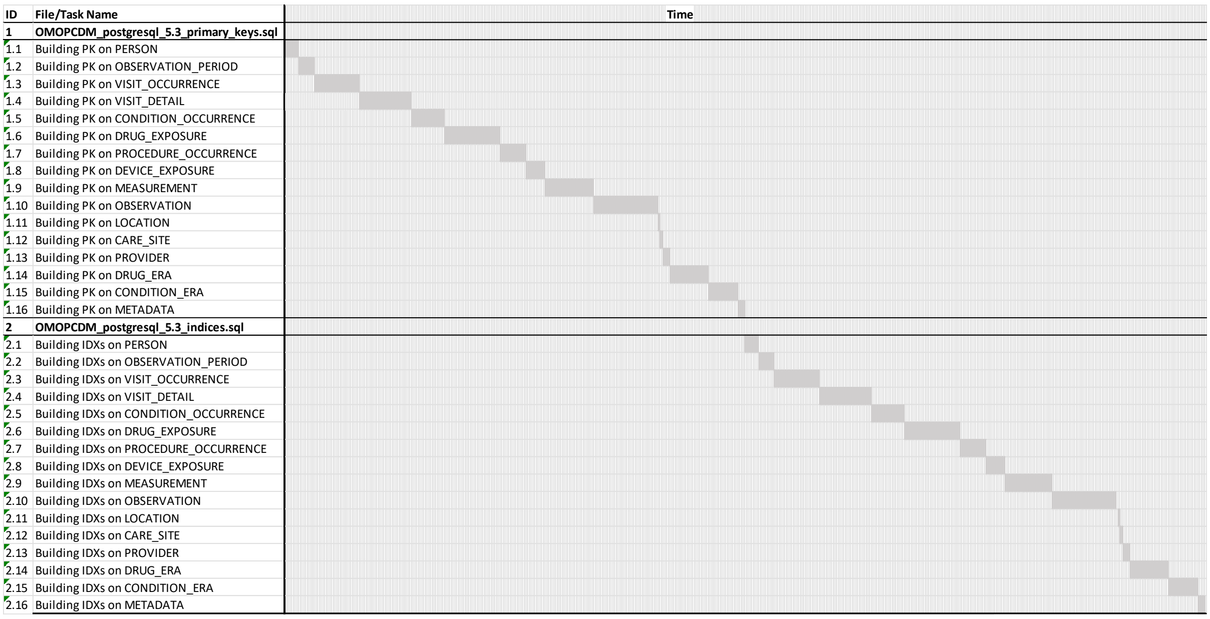
Wai Yi Man[1], Antonella Delmestri[1]
[1]NDORMS, University of Oxford, UK

Background

Transformation of real-world data into the Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) is not simply an Extract-Transform-Load (ETL) process. It also requires the building of primary keys, indexes and constraints using OHDSI provided sequential SQL code before the standardized data can be used. This implementation is a post-ETL operation, whose execution time depends closely on the data dimension, and can be very time consuming. The simpler approach for building primary keys, indexes, and constraints on OMOP CDM tables is running one after the other the OMOP CDM GitHub provided SQL scripts, one for PK, one for indexes and one for constraints. However, this operation could become significantly more efficient by splitting and merging these files in a way that allows concurrency to be used.

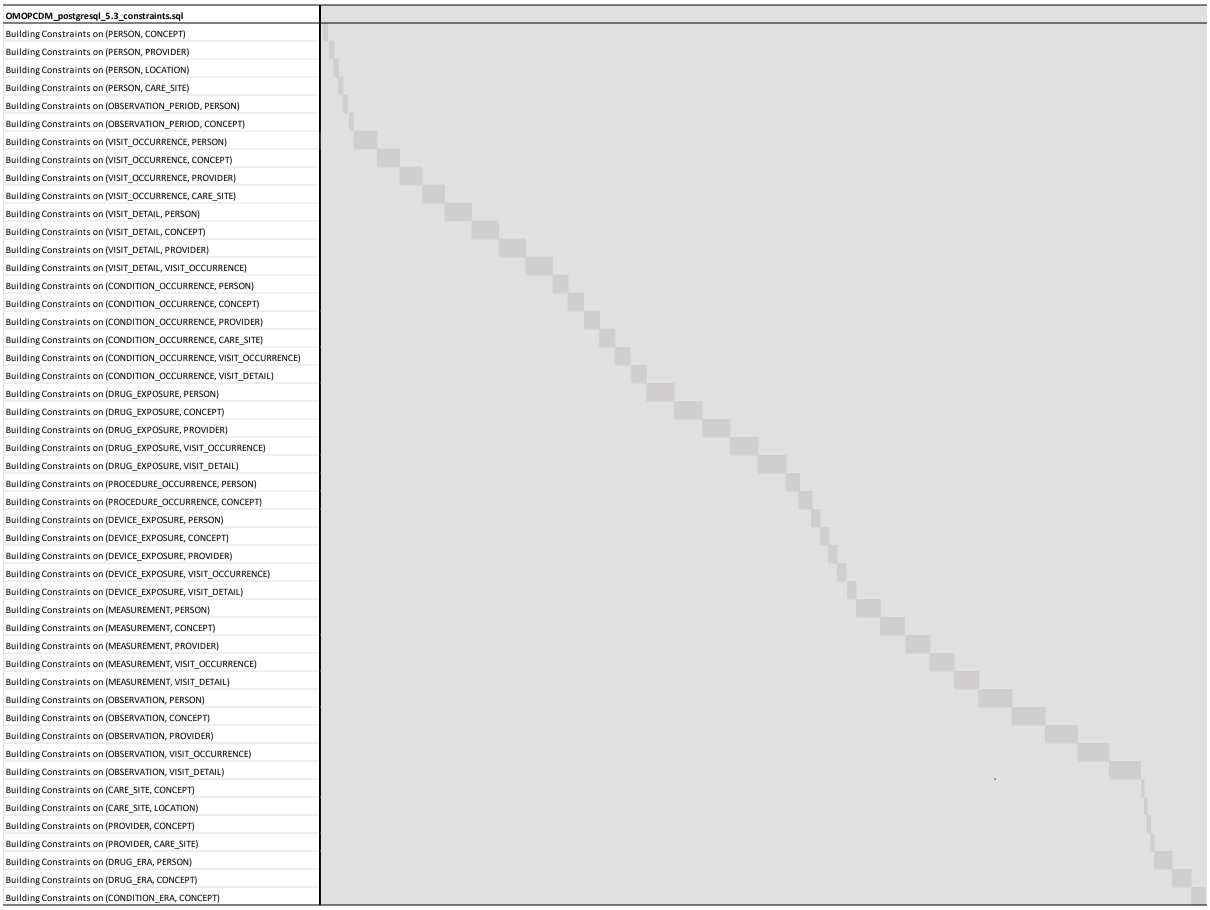
Methods

The native sequential approach requires waiting for the completion of building all PKs in all tables before starting to create the indexes and then the constraints as Figures 1 and 2 show.



ID	File/Task Name	Time
1	**OMOPCDM_postgresql_5.3_primary_keys.sql**	
1.1	Building PK on PERSON	
1.2	Building PK on OBSERVATION_PERIOD	
1.3	Building PK on VISIT_OCCURRENCE	
1.4	Building PK on VISIT_DETAIL	
1.5	Building PK on CONDITION_OCCURRENCE	
1.6	Building PK on DRUG_EXPOSURE	
1.7	Building PK on PROCEDURE_OCCURRENCE	
1.8	Building PK on DEVICE_EXPOSURE	
1.9	Building PK on MEASUREMENT	
1.10	Building PK on OBSERVATION	
1.11	Building PK on LOCATION	
1.12	Building PK on CARE_SITE	
1.13	Building PK on PROVIDER	
1.14	Building PK on DRUG_ERA	
1.15	Building PK on CONDITION_ERA	
1.16	Building PK on METADATA	
2	**OMOPCDM_postgresql_5.3_indices.sql**	
2.1	Building IDXs on PERSON	
2.2	Building IDXs on OBSERVATION_PERIOD	
2.3	Building IDXs on VISIT_OCCURRENCE	
2.4	Building IDXs on VISIT_DETAIL	
2.5	Building IDXs on CONDITION_OCCURRENCE	
2.6	Building IDXs on DRUG_EXPOSURE	
2.7	Building IDXs on PROCEDURE_OCCURRENCE	
2.8	Building IDXs on DEVICE_EXPOSURE	
2.9	Building IDXs on MEASUREMENT	
2.10	Building IDXs on OBSERVATION	
2.11	Building IDXs on LOCATION	
2.12	Building IDXs on CARE_SITE	
2.13	Building IDXs on PROVIDER	
2.14	Building IDXs on DRUG_ERA	
2.15	Building IDXs on CONDITION_ERA	
2.16	Building IDXs on METADATA	

Figure 1: Running OMOP CDM .sql files sequentially to build PKs and Indexes.

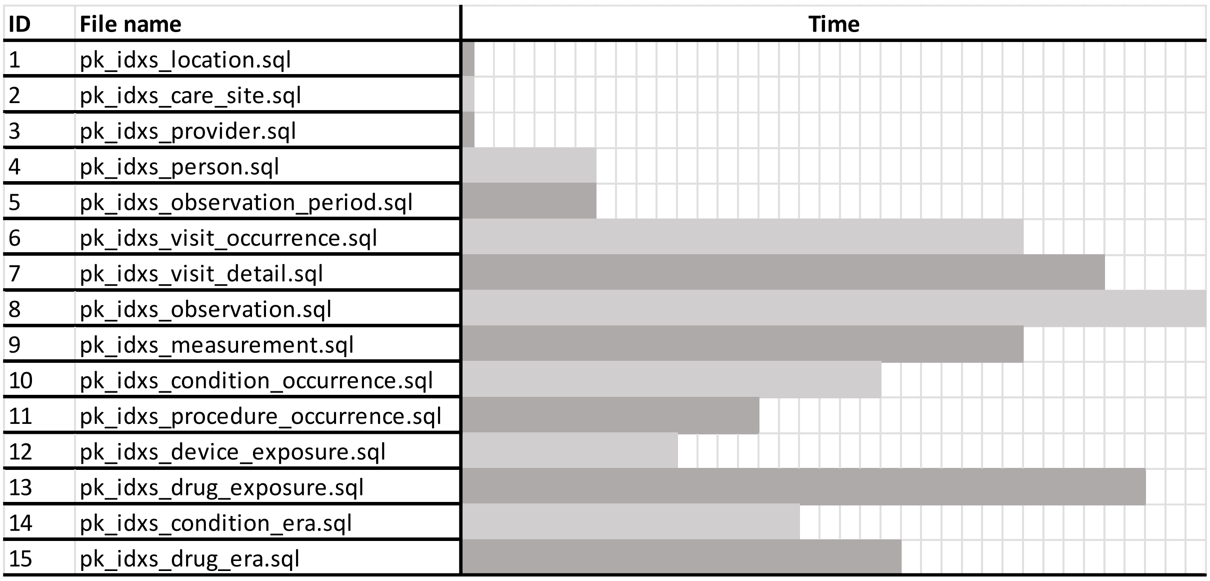


OMOPCDM_postgresql_5.3_constraints.sql

Building Constraints on (PERSON, CONCEPT)
Building Constraints on (PERSON, PROVIDER)
Building Constraints on (PERSON, LOCATION)
Building Constraints on (PERSON, CARE_SITE)
Building Constraints on (OBSERVATION_PERIOD, PERSON)
Building Constraints on (OBSERVATION_PERIOD, CONCEPT)
Building Constraints on (VISIT_OCCURRENCE, PERSON)
Building Constraints on (VISIT_OCCURRENCE, CONCEPT)
Building Constraints on (VISIT_OCCURRENCE, PROVIDER)
Building Constraints on (VISIT_OCCURRENCE, CARE_SITE)
Building Constraints on (VISIT_DETAIL, PERSON)
Building Constraints on (VISIT_DETAIL, CONCEPT)
Building Constraints on (VISIT_DETAIL, PROVIDER)
Building Constraints on (VISIT_DETAIL, VISIT_OCCURRENCE)
Building Constraints on (CONDITION_OCCURRENCE, PERSON)
Building Constraints on (CONDITION_OCCURRENCE, CONCEPT)
Building Constraints on (CONDITION_OCCURRENCE, PROVIDER)
Building Constraints on (CONDITION_OCCURRENCE, CARE_SITE)
Building Constraints on (CONDITION_OCCURRENCE, VISIT_OCCURRENCE)
Building Constraints on (CONDITION_OCCURRENCE, VISIT_DETAIL)
Building Constraints on (DRUG_EXPOSURE, PERSON)
Building Constraints on (DRUG_EXPOSURE, CONCEPT)
Building Constraints on (DRUG_EXPOSURE, PROVIDER)
Building Constraints on (DRUG_EXPOSURE, VISIT_OCCURRENCE)
Building Constraints on (DRUG_EXPOSURE, VISIT_DETAIL)
Building Constraints on (PROCEDURE_OCCURRENCE, PERSON)
Building Constraints on (PROCEDURE_OCCURRENCE, CONCEPT)
Building Constraints on (DEVICE_EXPOSURE, PERSON)
Building Constraints on (DEVICE_EXPOSURE, CONCEPT)
Building Constraints on (DEVICE_EXPOSURE, PROVIDER)
Building Constraints on (DEVICE_EXPOSURE, VISIT_OCCURRENCE)
Building Constraints on (DEVICE_EXPOSURE, VISIT_DETAIL)
Building Constraints on (MEASUREMENT, PERSON)
Building Constraints on (MEASUREMENT, CONCEPT)
Building Constraints on (MEASUREMENT, PROVIDER)
Building Constraints on (MEASUREMENT, VISIT_OCCURRENCE)
Building Constraints on (MEASUREMENT, VISIT_DETAIL)
Building Constraints on (OBSERVATION, PERSON)
Building Constraints on (OBSERVATION, CONCEPT)
Building Constraints on (OBSERVATION, PROVIDER)
Building Constraints on (OBSERVATION, VISIT_OCCURRENCE)
Building Constraints on (OBSERVATION, VISIT_DETAIL)
Building Constraints on (CARE_SITE, CONCEPT)
Building Constraints on (CARE_SITE, LOCATION)
Building Constraints on (PROVIDER, CONCEPT)
Building Constraints on (PROVIDER, CARE_SITE)
Building Constraints on (DRUG_ERA, PERSON)
Building Constraints on (DRUG_ERA, CONCEPT)
Building Constraints on (CONDITION_ERA, CONCEPT)

Figure 2: Running OMOP CDM .sql files sequentially to build constraints.

Since primary keys and indexes building on different tables are completely independent, we have created one file for each OMOP CDM table (i.e. *pk_idx_<tbl>.sql*) which includes both PK and indexes for that table, and then we have run these SQL files simultaneously, as Figure 3 shows.



ID	File name	Time
1	pk_idxs_location.sql	
2	pk_idxs_care_site.sql	
3	pk_idxs_provider.sql	
4	pk_idxs_person.sql	
5	pk_idxs_observation_period.sql	
6	pk_idxs_visit_occurrence.sql	
7	pk_idxs_visit_detail.sql	
8	pk_idxs_observation.sql	
9	pk_idxs_measurement.sql	
10	pk_idxs_condition_occurrence.sql	
11	pk_idxs_procedure_occurrence.sql	
12	pk_idxs_device_exposure.sql	
13	pk_idxs_drug_exposure.sql	
14	pk_idxs_condition_era.sql	
15	pk_idxs_drug_era.sql	

Figure 3: Running PK and indexes .sql files in parallel.

In order to run these .sql files in parallel we have created a Python program that benefits from modern concurrency methods implemented in an imported Python multiprocessing module (Brownlee, 2022).

We used the *ProcessPoolExecutor* class and tested each process for successful completion. The number of tasks running simultaneously should be based on the system resources, i.e., CPU number, CPU speed, random access memory, and database management system.

Building constraints in parallel requires more attention, however, this can be achieved by splitting the native sequential SQL script of constraints by table pair since each constraint specifies a relationship between a target table, and a parent table (Obe and Hsu, 2012). More constraints can belong to one table pair, and some tables are parents of several target tables.

This approach allows for the creation of distinct .sql files (i.e. *fk_<target_tbl>__<parent_tbl>.sql*) whose names, which reveal the tables involved in the included code, can be used to select those that can run in parallel without interfering with each other. For example, the same five parent tables *concept*, *person*, *provider*, *visit_detail* and *visit_occurrence* can be combined with the following eight target tables to create groups of five files that can run in parallel.

- condition_occurrence
- device_exposure
- drug_exposure
- measurement
- observation
- procedure_occurrence
- visit_detail
- visit_occurrence

Each group can be created by a simple list shift, paying attention that target and parent tables are always different, as illustrated in Figure 3.

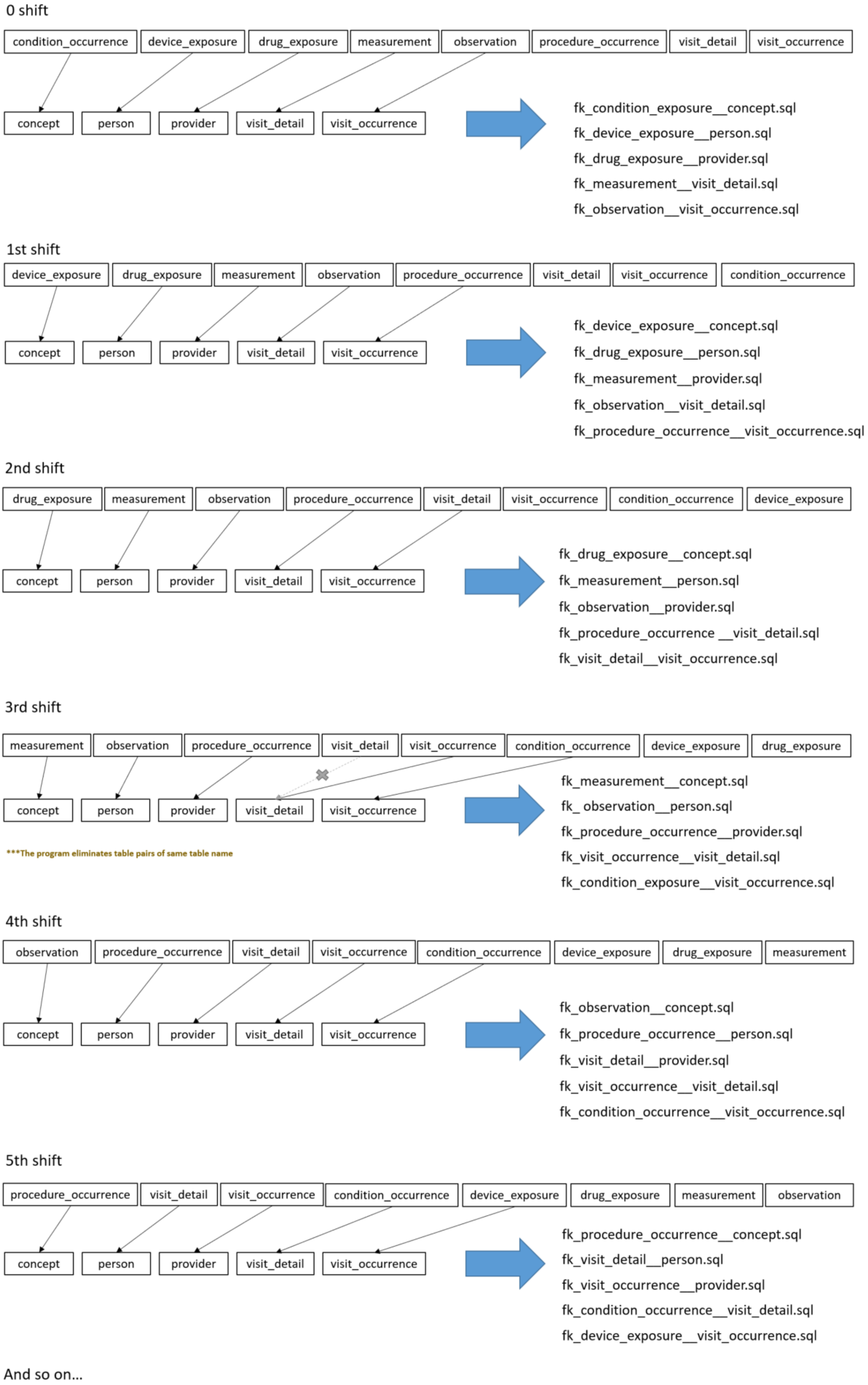

0 shift
condition_occurrence
device_exposure
drug_exposure
measurement
observation
procedure_occurrence
visit_detail
visit_occurrence
concept
person
provider
visit_detail
visit_occurrence
fk_condition_exposure__concept.sql
fk_device_exposure__person.sql
fk_drug_exposure__provider.sql
fk_measurement__visit_detail.sql
fk_observation__visit_occurrence.sql
1st shift
device_exposure
drug_exposure
measurement
observation
procedure_occurrence
visit_detail
visit_occurrence
condition_occurrence
concept
person
provider
visit_detail
visit_occurrence
fk_device_exposure__concept.sql
fk_drug_exposure__person.sql
fk_measurement__provider.sql
fk_observation__visit_detail.sql
fk_procedure_occurrence__visit_occurrence.sql
2nd shift
drug_exposure
measurement
observation
procedure_occurrence
visit_detail
visit_occurrence
condition_occurrence
device_exposure
concept
person
provider
visit_detail
visit_occurrence
fk_drug_exposure__concept.sql
fk_measurement__person.sql
fk_observation__provider.sql
fk_procedure_occurrence __visit_detail.sql
fk_visit_detail__visit_occurrence.sql
3rd shift
measurement
observation
procedure_occurrence
visit_detail
visit_occurrence
condition_occurrence
device_exposure
drug_exposure
concept
person
provider
visit_detail
visit_occurrence
***The program eliminates table pairs of same table name
fk_measurement__concept.sql
fk_ observation__person.sql
fk_procedure_occurrence__provider.sql
fk_visit_occurrence__visit_detail.sql
fk_condition_exposure__visit_occurrence.sql
4th shift
observation
procedure_occurrence
visit_detail
visit_occurrence
condition_occurrence
device_exposure
drug_exposure
measurement
concept
person
provider
visit_detail
visit_occurrence
fk_observation__concept.sql
fk_procedure_occurrence__person.sql
fk_visit_detail__provider.sql
fk_visit_occurrence__visit_detail.sql
fk_condition_occurrence__visit_occurrence.sql
5th shift
procedure_occurrence
visit_detail
visit_occurrence
condition_occurrence
device_exposure
drug_exposure
measurement
observation
concept
person
provider
visit_detail
visit_occurrence
fk_procedure_occurrence__concept.sql
fk_visit_detail__person.sql
fk_visit_occurrence__provider.sql
fk_condition_occurrence__visit_detail.sql
fk_device_exposure__visit_occurrence.sql
And so on…

Figure 3: Shift algorithm that identifies .sql files suitable to run in parallel

Result

In terms of big *O* notation, the time complexity of sequential execution of PK and indexes OMOP CDM provided .sql files is the sum of $O\big(R(T_i)\big)$, where R is the number of rows in the T_i table, and *n* is the number of tables.

$$O(R(T_1)) + O\big(R(T_2)\big) + \cdots + O\big(R(T_n)\big) = \sum_{i=1}^{n} O(R(T_i))$$

On the other hand, the time used to run the same statements concurrently on different tables will be:

$$\max_{1 \le i \le n} O\big(R(T_i)\big)$$

Regarding foreign keys, when processed on target tables T_i with parent tables P_j sequentially, the time complexity is:

$$\sum_{\substack{0 \le i \le n \\ 0 \le j \le m \\ T_i \ne P_j}} O(R(T_i, P_j))$$

With our concurrent approach, the time complexity becomes as follows:

$$\sum_{z=0}^{n-1} \max_{\substack{1 \le j \le m \\ i = j+z}} O\left(R\big(T_i, P_j\big)\right)$$

Conclusion

Concurrent processing can speed up the post-ETL operations significantly (e.g. hours to minutes, or days to hours) depending on the data volume: the bigger the OMOP CDM tables, the greater the time saving. Providing the post-ETL SQL code in a format suitable for parallel processing would help the OHDSI community to run the central code more efficiently.

References

Brownlee, J. (2022) *Python multiprocessing: The Complete Guide*, *Python Multiprocessing: The Complete Guide*. Super Fast Python. Available at: https://superfastpython.com/multiprocessing-in-python/ (Accessed: April 27, 2023).

Obe, R.O. and Hsu, L.S. (2012) PostgreSQL: up and running. Sebastopol, CA: O'Reilly Media.