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.

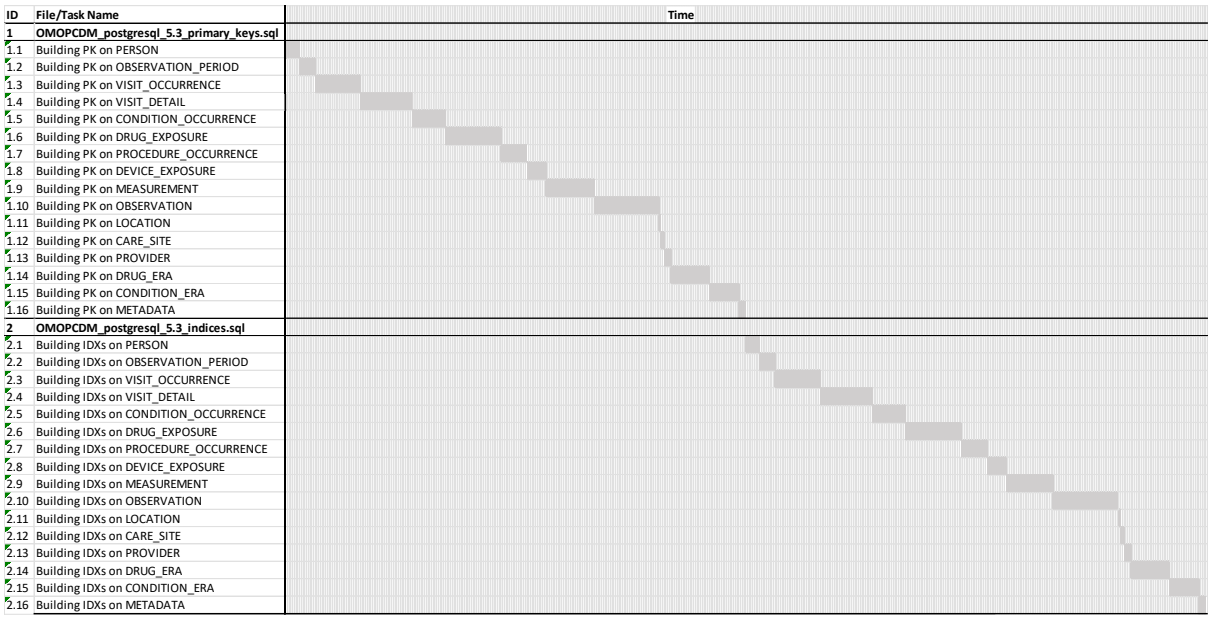

Figure 1: Running OMOP CDM .sql files sequentially to build PKs and Indexes.

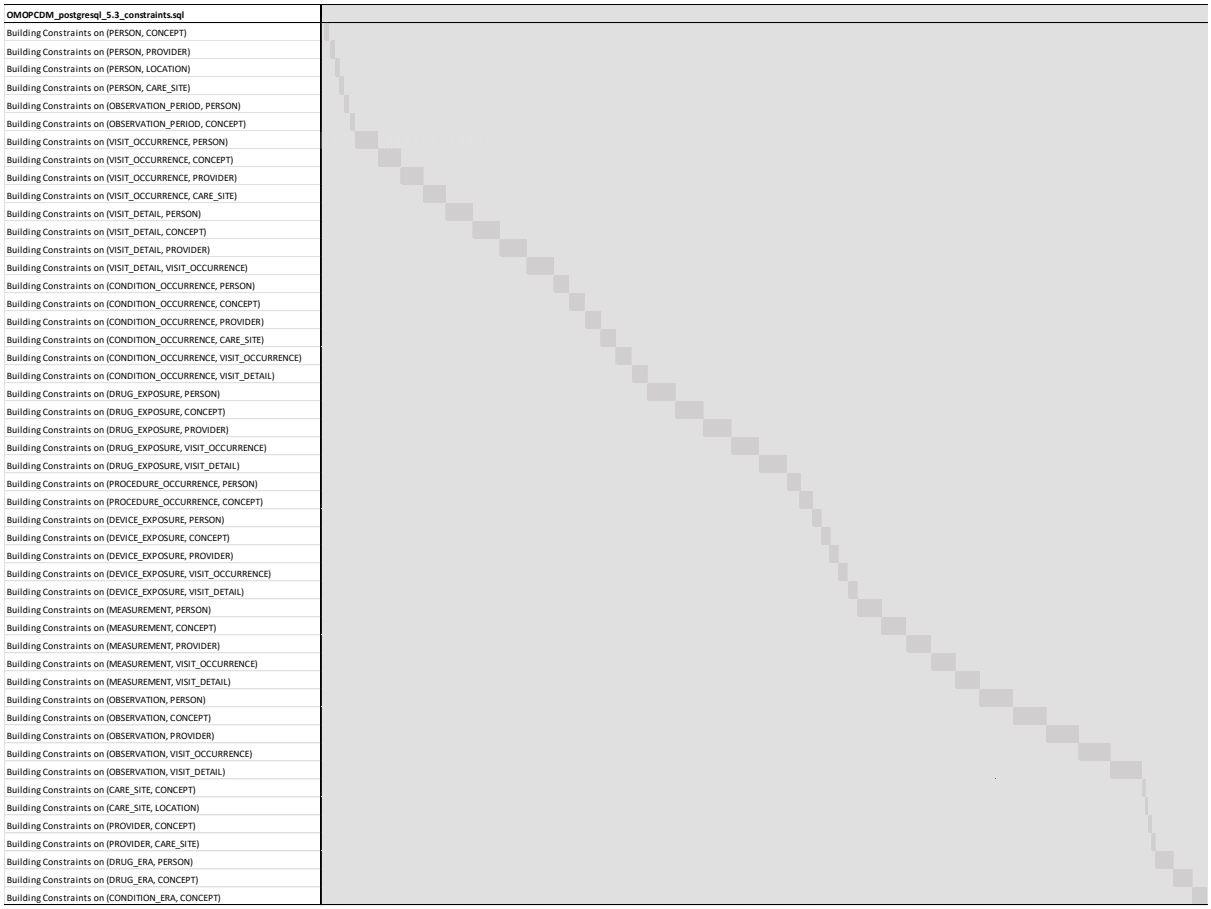

Figure 2: Running OMOP CDM .sql files sequentially to build constraints.

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.

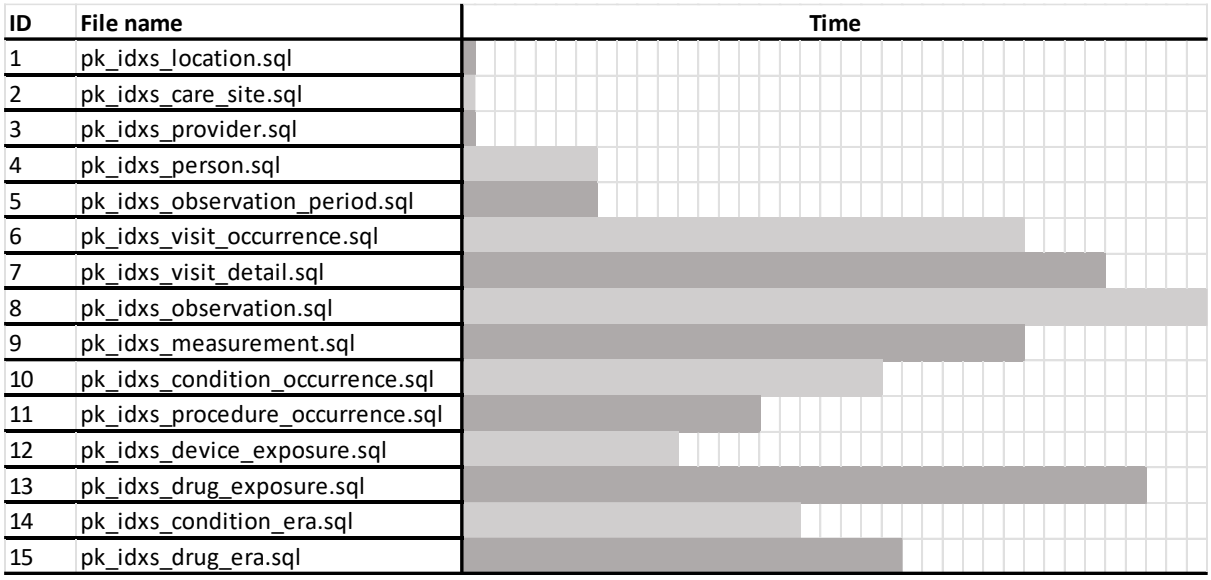

Figure 3: Running PK and indexes .sql files in parallel.

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.

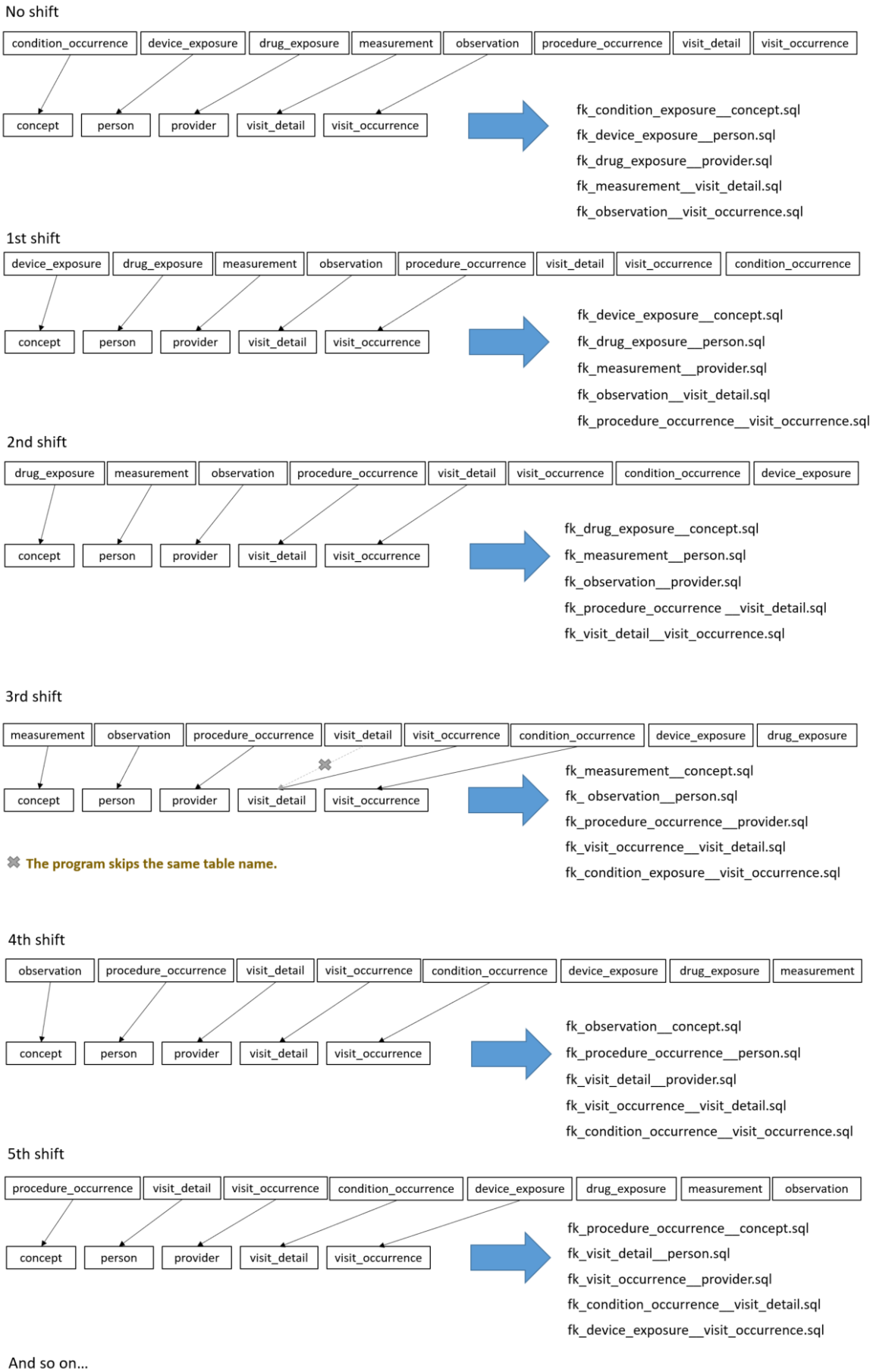

Figure 3: Shift algorithm that identifies .sql files suitable to run in parallel

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:

$$\sum_{i=1}^{n} \left(\sum_{\substack{j=1 \\ T_i \neq P_j}}^{m} O(R(T_i, P_j)) \right)$$

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

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

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.