Modular Data Analytics

Spinellis, Diomidis

DOI
10.1109/MS.2024.3409988

Publication date
2024

Document Version
Final published version

Published in
IEEE Software

Editor: Diomidis Spinellis
dds@aueb.gr

Modular Data Analytics

Diomidis Spinellis

A SOPHISTICATED ANALYSIS of data is often based on relational analytical processing (ROLAP) methods. These involve using SQL queries on a relational database system to perform on the data operations such as slice and dice, drill down, and roll up. Such queries can be complex, involving tens of tables and many intermediate steps. On large datasets they can also be expensive to run, taking hours or days to complete. Modularizing the SQL queries can make them more readable, testable, and amenable to incremental execution.

SQL offers two mechanisms to modularize queries. First, the SQL *common table expression* syntax (WITH *name* AS …) allows the naming of intermediate result sets within the query. However, such elements cannot be developed, tested, or run in isolation because they are part of a larger, often unwieldy, query. Second, SQL *views* (CREATE VIEW AS …) allow the permanent establishment of an alias for the results that a query would return. A view can be developed and tested as a separate unit. However, both mechanisms incur the query's cost every time it is evaluated, leading to monolithic, wasteful, and slow computations. On the other hand, they offer the possibility of advanced cross-query optimizations. *Materialized views*, which cache the results of a view's query, address the performance problem, but their automatic refresh when the base data change is not well supported in popular open source relational database management systems, such as PostgreSQL and MySQL/MariaDB.

Handling Dependencies

To facilitate the development of maintainable, time-efficient, and testable ROLAP queries, I developed *simple-rolap* (github.com/dspinellis/simple-rolap/), a small open source software framework that automates the dependency analysis and orchestrates the execution of multiple modular queries. I have used it for tens of queries[1,2] on the GHTorrent[3] GitHub metadata and the Alexandria3k[4] bibliographic records. The framework's design is based on *convention over configuration* centered on two databases. A main database contains the primary queried data, which are assumed to be infrequently modified. A secondary database is used for caching derived intermediate ROLAP results. Users split complex queries into simple ones that either create intermediate tables or report results; see the example in Figure 1. Both types of queries can be easily unit tested with RDBUnit.[5]

The running of queries in the appropriate order is orchestrated by *GNU make*, the GNU implementation of the Unix *make* tool,[6] which was originally devised to automate the building of programs.[5] The *make* tool works by reading a file (named Makefile by default) that specifies build rules. Each rule is written as a target file followed by a colon and (optionally) some prerequisite files that are required to build that target. Any following tab-indented lines contain the commands that must be executed for building the target. The *make* tool will read the rules and establish a dependency graph according to the targets and their prerequisites. It will then execute, in the appropriate order, all commands needed to build each target if it depends on prerequisite files that are missing or that have been modified after the target's modification time.

As an example, the following two rules specify that to create the sales-statistics.txt file (using the *ministat*

Digital Object Identifier 10.1109/MS.2024.3409988
Date of current version: 13 August 2024

program) the file sales.csv must have been fetched through the specified *curl* command. (The >$@ incantation requires further explanation. The > sign redirects the command's output to the file named on its right, $ is the prefix for *make*'s variables, and @ is the name of a built-in variable that is set to the value of the rule's target, so in the first rule sales-statistics.txt.)

```
sales-statistics.txt: sales.csv
    ministat sales.csv >$@
sales.csv:
  curl https://example.com/sales.csv>$@
```

Although *make* was originally devised to automate program builds, it can accommodate any task in which files must be updated when some others are missing or change. For instance, I am using it to maintain my website's content, to format course notes, and to typeset articles and books.

In the case of *simple-rolap*, *make* starts by running a Unix shell script that analyzes all SQL queries residing in a project's directory and creates a list of their dependencies so that they can be run in the appropriate order. It stores the dependencies into a file that the *simple-rolap* Makefile includes, if it exists. Having read the query dependencies, *make* (re)runs the queries whose results are outdated or missing.

For *make* to work, all dependencies must be mapped to files. Query results satisfy this requirement because *simple-rolap* stores them in a directory named reports. On the other hand, database tables are not directly visible to *make* as files. To address this, when *simple-rolap* executes a table creation query, it creates or updates the modification time of an empty file named after the corresponding table in a directory named tables, which *make* can then readily use for dependency management.

A Makefile *pattern* rule (one matching any specified file) for all files with a .sql suffix invokes another script that executes each SQL query. For queries that output results, the script stores them in a file named after the query under the reports directory. For queries that create an intermediate table, the script creates or updates the associated time stamp file in the tables directory. Both types of files are the entities that *make* uses to determine which targets are missing or outdated and need to be (re)created.

Other *simple-rolap* scripts create or drop the ROLAP database, run the *RDBUnit* tests, and create editor tags. (An editor "tags" database maps navigable source code entities, such as methods, functions, classes,

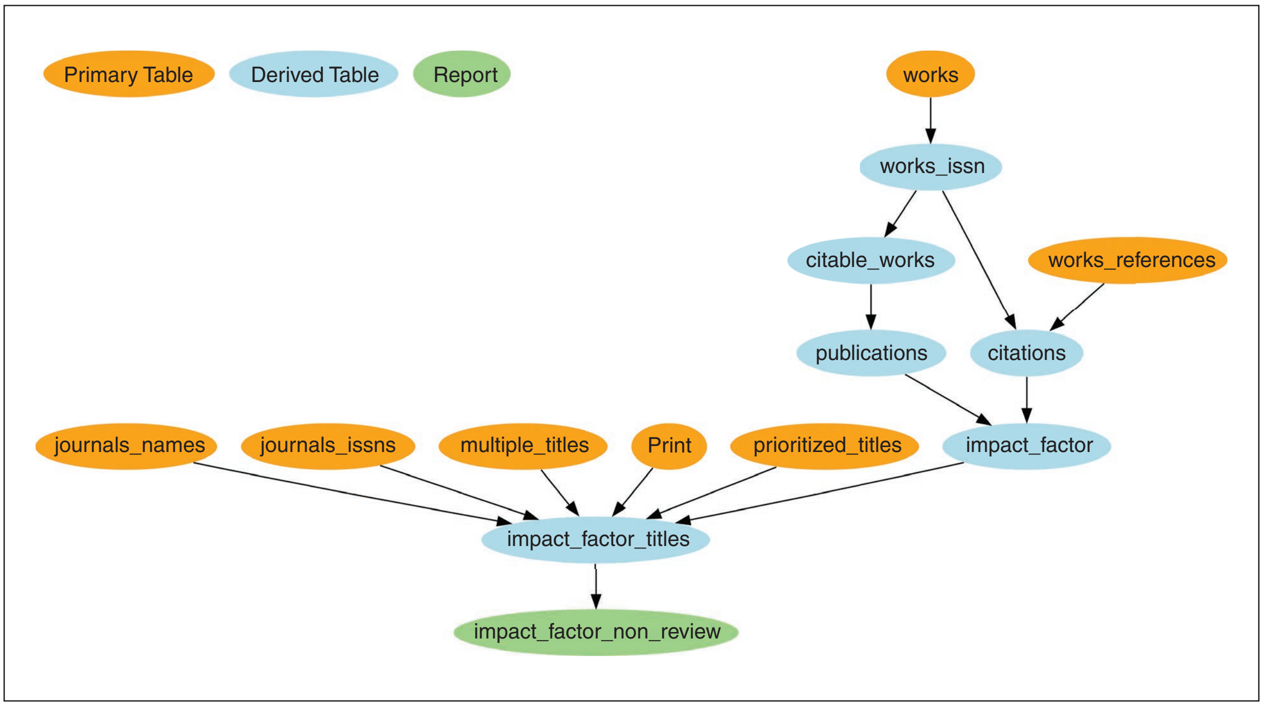

FIGURE 1. Query steps for calculating the journal impact factor.

or query names, in the case of *simple-rolap,* into hyperlinked locations in source code files that the code editor will show when the user asks to see the definition of a given entity.) Most of this functionality is directly coded in the Makefile as single shell commands. The *simple-rolap* framework makes this functionality available through targets named after the created files (as is, for example, the case for the tags target) or, alternatively, a feature of *make* called *phony targets*. These are targets that do not correspond to files but have their commands executed whenever *make* is invoked with that name as the target.

The shell scripts underlying *simple-rolap*'s operation abstract the specific database management systems by calling their command-line interfaces with the appropriate options. In addition, the script that creates the dependency list uses the *sed* stream editor to extract the dependencies from the SQL query files, while the script that runs the SQL queries adds commands from a local configuration and drops tables before issuing the user-supplied creation command.

The help pseudotarget, which lists available *make* targets, involves a trick that has often served me to provide documentation regarding a *make* file's operation. After each target, I add a comment explaining the target's purpose, like this:

```
help: # Help: Show this help message
  [...]
clean: # Help: Drop database and remove generated
files
```

The help target command is a short *sed*-based pipeline that goes through the *make* file and converts all target names that are followed by Help:-prefixed comments into a sorted list of explanatory text. You may want to adopt this rule in your own *make* files.

The *simple-rolap* framework also offers you rules and configuration options to test, visualize, navigate, benchmark, and troubleshoot the workflow's execution.

Using *simple-rolap*

To use *simple-rolap,* clone its GitHub repository and create a file named Makefile (by convention the file with *make*'s configuration) that includes the *simple-rolap* Makefile and specifies the parameters of your analysis. As a minimum, define the relational database management system you will be using by setting the environment variable RDBMS to one of mysql, postegresql, or sqlite and the names of the main and ROLAP databases through the variables MAINDB and ROLAPDB. For the PostgreSQL and MySQL/MariaDB systems you will also need to set the DBHOST and DBUSER variables and arrange for corresponding database engine authentication and authorization. With that setup you can begin writing your queries, storing them in files suffixed with .sql. Each query can either store its results in an intermediate table (CREATE TABLE ...), which will be dropped and created as needed, or it can obtain a result set (SELECT ...), which will be stored in the reports directory with the query file's base name suffixed by .txt. For dependency tracking to work, place the table names on the same line as the SQL FROM and JOIN clauses, indent nested SELECT clauses (such as those used for creating tables), and prefix ROLAP tables with the corresponding database name.

Sophisticated workflows often have additional processing steps. For example, they may fetch data from remote sources or nonrelational databases, populate tables with them, or postprocess obtained results for statistical analysis, charting, or report formatting. You can specify these through additional *make* rules using the reports files as prerequisites and the table time stamp files as targets or as prerequisites. Consider as an example the following two *make* rules:

```
tables/metrics: metrics.csv
tables/cdindex: tables/base
        calc-index $(ROLAPDB).db >$@
```

The first rule specifies that the metrics table has the file metrics.csv as its prerequisite, presumably because the SQL statement that creates the table imports its data from that file. The second rule specifies that the cdindex table target depends on the (generated) base table prerequisite and is created by running the calc-index program on the ROLAP database (creating the table time stamp file as a side effect) rather than through an automatically detected SQL query. You can also inject prerequisites that need to be satisfied before any other processing (for example, to populate the main database) by specifying them in the DEPENDENCIES variable. Similarly, additional targets can be specified in the ALL variable.

You can control the workflow's execution in several ways. Run make clean to remove all auto-generated files, so that you can start a new analysis from scratch. Remove individual reports or table files to rebuild those and any dependents. You can also put SQL statements that you want to precede each query (for example, MySQL optimizer tuning SET commands) in the file .config.sql.

The *simple-rolap* framework also offers you rules and configuration options to test, visualize, navigate, benchmark, and troubleshoot the workflow's execution.

- Run make test to run any RDBUnit unit tests you may have written. Pass UNIT=*unit-name* to execute only the specified unit test.
- Run make graph.png (or .pdf or .svg) to generate a diagram of the ROLAP queries' dependencies. Use full-graph rather than graph as the target name to also include the main database tables.
- Run make tags to create a tags file that many editors can use to automatically navigate between the queries.
- Run make ordered-dependencies to create a correspondingly named file with the queries listed in the order determined by their dependencies.
- Run make V=1 (verbose) to see the commands executed within the Makefile.
- Run make V=1 TIME=time to see timing information of executions.
- Run make V=2 to also see the commands executed within the *simple-rolap* shell scripts.

Finally, run make help to see a summary of the available targets and configuration options.

This adventure in code taught me two lessons. First, problems that crop up repeatedly can often be profitably addressed through bespoke small-scale tool building. (The source code of *simple-rolap* amounts to about 600 lines.) Second, using existing mature tools, such as *make*, to carry out the heavy lifting simplifies tool building.

ABOUT THE AUTHOR

DIOMIDIS SPINELLIS is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business, Athens 104 34, Greece, and a professor of software analytics in the Department of Software Technology at the Delft University of Technology, 2600 AA Delft, The Netherlands. He is a Senior Member of IEEE. Contact him at dds@aueb.gr.

References

1. D. Spinellis, Z. Kotti, and A. Mockus, "A dataset for GitHub repository deduplication," in *Proc. 17th Int. Conf. Mining Softw. Repositories (MSR)*, New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 523–527, doi: 10.1145/3379597.3387496.
2. D. Spinellis, Z. Kotti, K. Kravvaritis, G. Theodorou, and P. Louridas, "A dataset of enterprise-driven open source software," in *Proc. 17th Int. Conf. Mining Softw. Repositories (MSR)*, New York, NY, USA: Association for Computing, Jun. 2020, pp. 533–537, Machinery. doi: 10.1145/3379597.3387495.
3. G. Gousios and D. Spinellis, "GHTorrent: Github's data from a firehose," in *Proc. 9th IEEE Working Conf. Mining Softw. Repositories (MSR)*, M. Lanza, M. D. Penta, and T. Xie, Eds., Piscataway, NJ, USA: IEEE, Jun. 2012, pp. 12–21, doi: 10.1109/MSR.2012.6224294.
4. D. Spinellis, "Open reproducible scientometric research with Alexandria3k," *PLoS One*, vol. 18, no. 11, Nov. 2023, Art. no. e0294946, doi: 10.1371/journal.pone.0294946.
5. D. Spinellis, "Unit tests for SQL," *IEEE Softw.*, vol. 41, no. 1, pp. 22–26, Jan. 2024, doi: 10.1109/MS.2023.3328788.
6. S. I. Feldman, "Make—A program for maintaining computer programs," *Softw., Pract. Exp.*, vol. 9, no. 4, pp. 255–265, 1979, doi: 10.1002/spe.4380090402.