Scaling Database Systems
Winter Term 2025/26
Chair of Scalable Database Systems
Prof. Dr.-Ing. S. Scherzinger

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

Milestone 4: MapReduce Optimization

Optimizing miniHive

In the fourth milestone, we optimize *miniHive*. The goal is to reduce the total amount of data stored in intermediary files within HDFS.

1 Evaluation Data

We use data from the TPC-H benchmark, describing customers and their orders. This is a popular benchmark where synthetic data can be generated based on a scale factor (SF).

Figure 1 describes the database schema and has been taken from the official benchmark description. We assume that all data adhere to this schema.

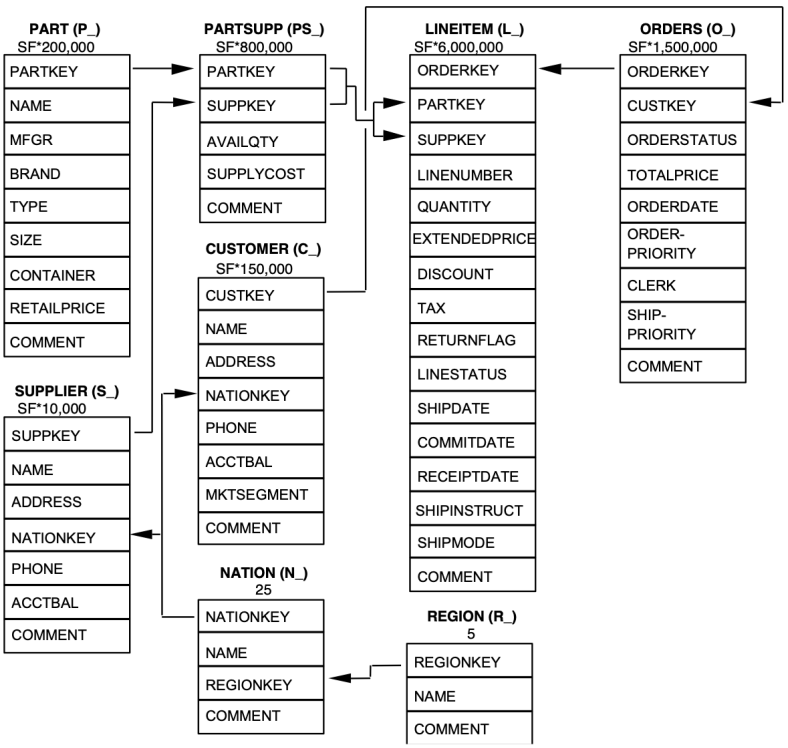

Figure 1: The TCP-H benchmark data [Source: www.tpc.org.]

The parentheses following each table name contain the prefix of the column names for that table. The arrows point in the direction of the one-to-many relationships between

tables. The number/formula below each table name represents the cardinality (number of rows) of the table. Some are factored by the scale factor SF.

2 Calling miniHive

For evaluating a single SQL query on the command-line, *miniHive* gets

- an optional flag telling miniHive to switch Milestone 4 optimizations on,
- the optional scale factor, allowing you to derive the *approximate* sizes of tables,
- whether the input files are stored in HDFS or available as local files,
- the optional execution environment (LOCAL or HDFS), set to HDFS by default,
- the SQL query over TPC-H data to evaluate.

This is how we call the optimized *miniHive* for execution on local files:

```
python3 miniHive.py --O --SF 1 --env LOCAL "select distinct N_NAME from NATION"
```

If called in `LOCAL` mode, this outputs an approximation of the HDFS storage costs (more on this later). **Make sure your submitted version does not write any other information to standard out, as this will confuse the test scripts.**

3 Material in StudIP

- A sample dataset that was generated with scale factor 0.01 (`data_0_01.zip`).
- The file `miniHive.py`. You may alter this file.
- The file `costcounter.py`. **You must not alter this file.**
- A list of SQL test queries in `miniHive.q`.

4 What to Submit

- All files to run miniHive: `sql2ra.py`, `raopt.py`, `ra2mr.py` and `miniHive.py`.
- A README file (`README.md`) with a 1-paragraph description of the optimization(s) that you have implemented. The README must have this format:

```
# Author: <your lastname>, <your firstname>

# My approach:
<1-paragraph description of your optimizations>
```

- You can upload the files individually or as ZIP file.

5 Earning Points

Your submission will be tested using a fully automated Python script. **Make sure that you do not break this functionality.**

The script parses the local temporary files, assuming that they have the format familiar from Milestone 3, namely "`some-key json-encoded-value`". **Do not change this format. Do not tinker with the JSON-encoding of the value.**

The script then determines the total number of characters encoding MapReduce values in all output and intermediary files (the input files are the same for everybody, so we ignore them). We use this as a rough proxy for communication costs. **Do not fiddle with the intermediary files.**

The README file must be comprehensible. Your implementation must run in the unoptimized and the optimized mode.

You must implement at least one genuine optimization. We recommend to implement chain folding, but projection pushing (possibly without duplicate elimination) is also a good strategy, as is cost-based join reordering. You may also implement your own optimizations.

Optimizing the format of temporary files (e.g., using compression) will earn zero points and you will be excluded from the bonus points ranking.

The automated scripts must be able to run all test queries from `miniHive.q` with your implementation on HDFS and on LOCAL files with a secret scale factor, and measure the storage costs for temporary files.

In particular, your implementation has to fulfill the following criteria:

- For all queries in `miniHive.q`, your costs must be lower or equal to those of your (unoptimized) implementation of Milestone 3 (all implementations of Milestone 3 should have the same costs). This requirement ensures that your optimization does not actually make things worse.
- For at least half of the queries in `miniHive.q`, your costs must be lower than those of the (unoptimized) implementation of Milestone 3. That means that you cannot just resubmit the same code as in Milestone 3.
- For at least three queries in `miniHive.q`, your optimization must be able to reduce costs by two thirds.

Note: Make sure your miniHive implementation gives correct results in optimized *and* non-optimized mode.

In addition, the best 10 implementations, measured based on the optimization and the produced costs, are earning bonus points on top. See the kickoff slides in Stud.IP for all details. The queries for the ranking will be selected by us and may differ from the queries given in `miniHive.q`. Your solution is only considered for the ranking if it produces a *correct* result for *all* queries.

Remarks on grading: For Milestone 4, all you are required to provide is a correct and clean implementation that

- passes all public tests from Milestone 3 on Praktomat (in unoptimized and optimized mode),
- can execute the queries from the same unit tests in HDFS mode on the miniHive Docker container,
- meets the optimization criteria stated above.

Praktomat will run the unit tests in `test_ra2mr.py` and `test_e2e.py` (from Milestone 3) upon submission, once with `optimize=False` and once with `optimize=True`. Solutions that rely on hard-coding or other non-general methods will not be accepted by us. Secret tests in Praktomat are designed to detect such cases.

Upload your submission as one submission to Praktomat: https://praktomat.sdbs.fim.uni-passau.de (accessible only within the Uni Passau network or via the Uni Passau VPN).

After the Praktomat deadline has passed, your implementation will also be tested on the miniHive Docker container. Your implementation will be executed on Hadoop and data from HDFS will be processed: We recommend that you try this out beforehand (`--exec-environment HDFS`). Just because the unit test pass locally does not mean that the code runs on MapReduce, e.g., when your code relies on global variables.

Deadline: January 30, 2026, 12:00 (noon, daylight). A valid submission must pass all public tests in the `LOCAL` and `HDFS` execution environment, the plagiarism check, and must not be hard-coded.

Note: From January 8, 2026, you can no longer use OpenVPN to access Praktomat and upload your submissions. Instead, you need to use eduVPN. It is your responsibility to ensure that you can hand in your submission timely through Praktomat, either by setting up eduVPN or by accessing Praktomat directly from the campus network (e.g., via eduroam) without a VPN.