

Report for Assignment 1

- 0. Group member
- 1. Introduction
- 2. The environment for the experiment
 - 2.1 Hardware environment
 - 2.2 Software environment
- 3.Procedures to verify the SF-1 results
 - 3.1 MonetDB
 - 3.1.1 Load data
 - 3.1.2 Run queries
 - 3.1.3 Verify the results
 - 3.2 MySQL
 - 3.2.1 Load data
 - 3.2.2 Run queries
 - 3.2.3 Verify the results
- 4. Compare query execution
- 5. Implementation of the queries in Python
 - 5.1 Implement q06.sql and q01.sql on SF-1 dataset
 - 5.1.1 Methods
 - 5.1.2 Verify the results of q01.sql implementation
 - 5.2 Implement q06.sql and q01.sql on SF-3 dataset
- 6. Performance comparison between DBMS and Python implementation
 - 6.1 Visualisation
 - 6.2 Analysis of the performance
- *Appendix
 - *.1 MonetDB
 - *.1.1 Configuration
 - *.1.2 Commands

Report for Assignment 1

0. Group member

The group number is 6.

Name	Student Number	Tasks
Jie Chen	S4162315	All code & report

1. Introduction

This task is using the TPC-H benchmark tool to test the performance of MonetDB and one other DBMS (I use MySQL here). All of the dataset and sql scripts are provided. Also have to choose one of the programming language to implement the same query as the sql scripts (I choose Python here).

2. The environment for the experiment

2.1 Hardware environment

Here is the hardware environment of my experiment, and you don't need to use the same environment as mine. I just use my laptop MacBook pro 14-inch, 2021.

- Chip
Apple M1 Pro chip, 200GB/s memory bandwidth
- CPU
Clock rate: 2064-3220MHz, 24MB Level 3 Cache
- Main memory
32GB
- Disk
512GB SSD, 4900 MB/s read speed and 3951 MB/s write speed

2.2 Software environment

Download all of the software and config them properly to repeat the experiment.

- MonetDB
Version: v11.51.3 (Aug2024)
- MySQL
Version: Ver 8.3.0 for macos14.2 on arm64
- Python
Version: 3.12.2
- DBeaver – as the client to MonetDB and MySQL
Version: 24.2.0.202409011551

3.Procedures to verify the SF-1 results

First, I download the `MonetDB` and `MySQL`, and follow the official instruction to set up the username, password and port. Then I use the tool `DBeaver` to connect to the database of `MonetDB` and `MySQL`. `DBeaver` is a user friendly database client tool, and I will execute all of the sql script through it.

I use the provided scripts and data from BrightSpace directly, and follow the instructions in the task 1.

3.1 MonetDB

3.1.1 Load data

Create 2 databases called `SF1` and `SF3`.

Use the DBeaver to connect to these 2 databases separately.

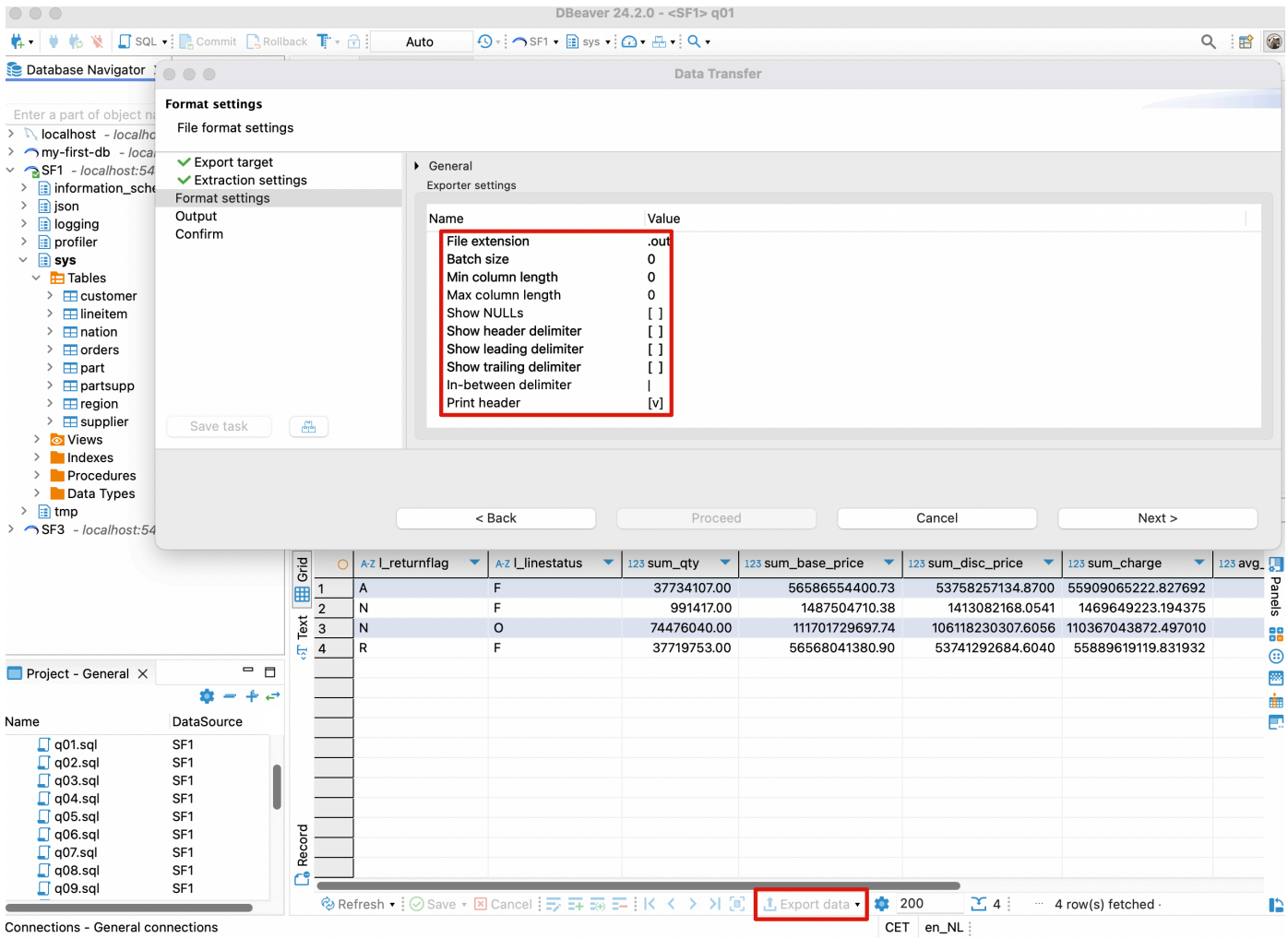
Import the create table script `.../dbgen/MonetDB/0-create_tables.sql` into the `SF1`, and then run the script to create table.

Import the load data script `.../dbgen/MonetDB/1-load_data.SF-1.sql` into the `SF1`. I change the data location to the absolute file path (`.../dbgen/SF-1/data/xxx.tbl`) for each line and then run the script to load the SF-1 data.

Finally, I import the `add_constraints` script `.../dbgen/MonetDB/2-add_constraints.sql` into the `SF1`, and then run it to set constraints.

3.1.2 Run queries

Import all of the queries into the database `SF1`, and then run these sql scripts one by one. Once we get the results of each script, we can use the tool of `DBeaver` to export the data with the specific format.



Just click on the button on the bottom, and then config the format as the screenshot shows. Then we get all of the output file from the queries. Check the `README` file located in the `.../dbgen/check_answers/` to find the file format requirements, and adjust the configuration in the export file process if necessary.

Make sure the output file format satisfy the requirements and then store them into the folder `/Users/jay/Desktop/monetdb_sf1_out`.

3.1.3 Verify the results

I enter the file path `.../dbgen/check_answers/`, and run the command.

```
# command to verify the results
```

```
./pairs.sh /Users/jay/Desktop/monetdb_sf1_out /Users/jay/Desktop/lessons material/2024-2/Advanced data management/assignment 01/TPC-H_V3.0.1++/dbgen/answers
```

The file path `/Users/jay/Desktop/monetdb_sf1_out` is the place I store the output files of previous queries. And the file path `/Users/jay/Desktop/lessons material/2024-2/Advanced data management/assignment 01/TPC-H_V3.0.1++/dbgen/answers` is the folder to store the official correct output files.

Then the script generate folders including `./data_01`, `./data_management`, `./management`, `./material`. After I check the content of these files, I find all of these folders are used to store the comparison log files and result files which is shown as follows,

```
dbgen > check_answers > data_01 > TPC-H_V3.0.1++ > dbgen > answers > ≡ analysis_1.log
1   Found 0 unacceptable mismatches
2
```

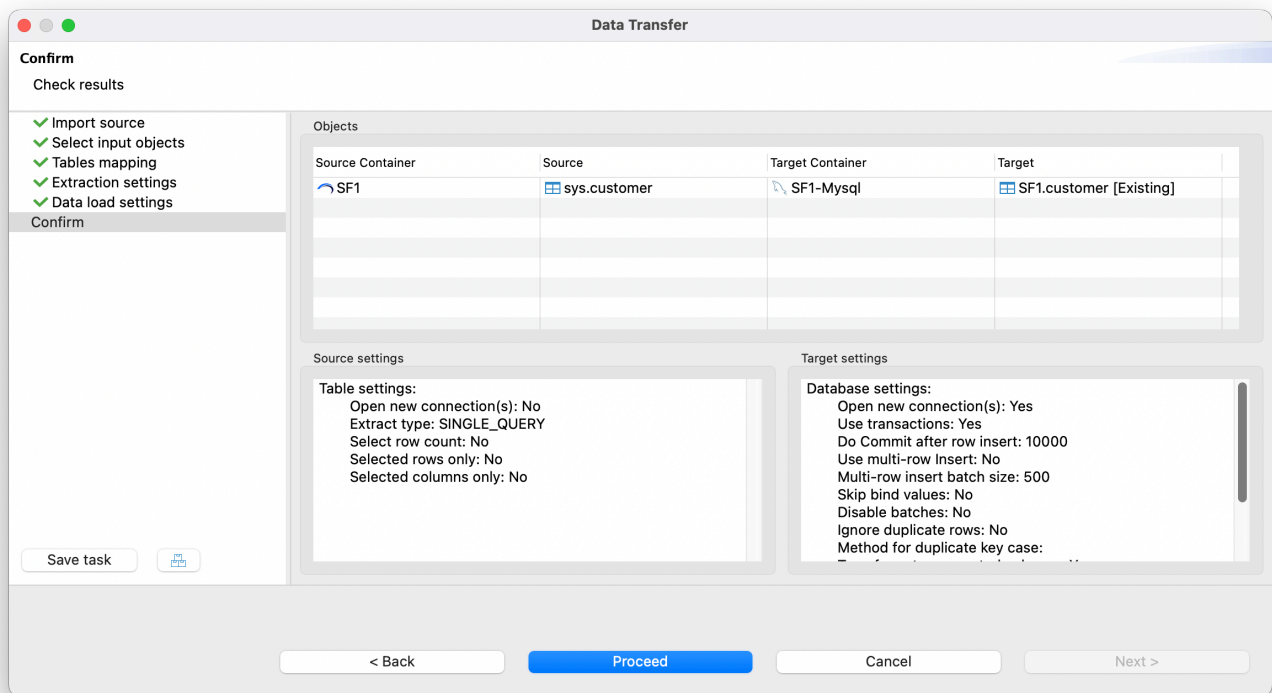
At the mean time, the terminal shows all of these logs which indicate all the results of 22 queries find 0 unacceptable mismatches. And we can verify that all the results are correct.

```
Comparing /Users/jay/Desktop/monetdb_sf1_out to 01/TPC-H_V3.0.1++/dbgen/answers
Query 1 0 unacceptable mismatches
Query 2 0 unacceptable mismatches
Query 3 0 unacceptable mismatches
Query 4 0 unacceptable mismatches
Query 5 0 unacceptable mismatches
Query 6 0 unacceptable mismatches
Query 7 0 unacceptable mismatches
Query 8 0 unacceptable mismatches
Query 9 0 unacceptable mismatches
Query 10 0 unacceptable mismatches
Query 11 0 unacceptable mismatches
Query 12 0 unacceptable mismatches
Query 13 0 unacceptable mismatches
Query 14 0 unacceptable mismatches
Query 15 0 unacceptable mismatches
Query 16 0 unacceptable mismatches
Query 17 0 unacceptable mismatches
Query 18 0 unacceptable mismatches
Query 19 0 unacceptable mismatches
Query 20 0 unacceptable mismatches
Query 21 0 unacceptable mismatches
Query 22 0 unacceptable mismatches
```

3.2 MySQL

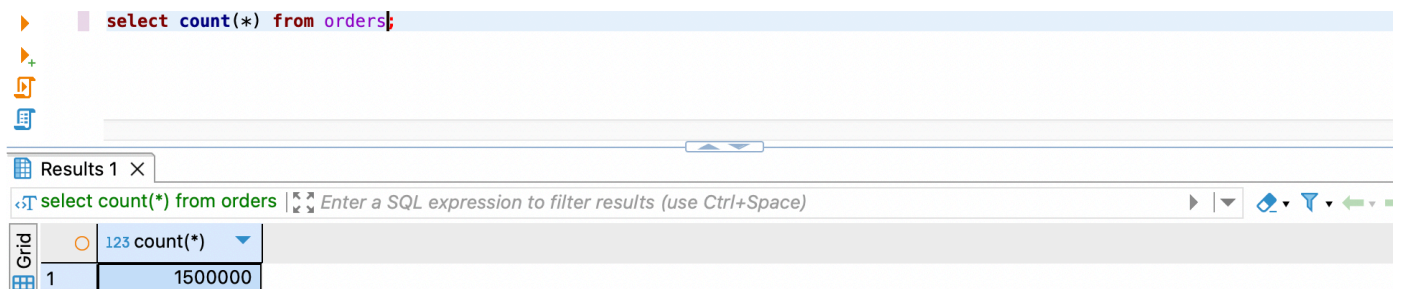
3.2.1 Load data

When I try to run the load data scripts in the MySQL database, I find some errors and constraints. So I choose to copy import the data from MonetDB tables to MySQL tables with the DBeaver. An example is shown as below.



To make sure all of the data have been successfully imported, I run the query command to check the total row in these tables.

For example, I find 1500000 rows in table orders in the mysql database which is same as that in MonetDB.



Finally, I import the add_constraints script `.../dbgen/MonetDB/2-add_constraints.sql` into the `SF1`, and then run it to set constraints. But it takes far more time in MySQL than in MonetDB, almost 537s!

3.2.2 Run queries

Use the same sql scripts as the MonetDB to execute the queries. Several sql scripts need to change the syntax to MySQL 8.0. I have to change the syntax in `q01.sql`, `q15.sql`. Then I store all of the output files into the folder `/Users/jay/Desktop/mysql_sf1_out`

3.2.3 Verify the results

I use the same script to verify the results. I also move the answers into the folder `/Users/jay/Desktop/answers`.

Run the command:

```
# command to verify the results
./pairs.sh /Users/jay/Desktop/mysql_sf1_out /Users/jay/Desktop/answers
```

Similarly, I get the results from terminal, which means all the results are correct.

```
Comparing /Users/jay/Desktop/mysql_sf1_out to /Users/jay/Desktop/answers
Query 1 0 unacceptable mismatches
Query 2 0 unacceptable mismatches
Query 3 0 unacceptable mismatches
Query 4 0 unacceptable mismatches
Query 5 0 unacceptable mismatches
Query 6 0 unacceptable mismatches
Query 7 0 unacceptable mismatches
Query 8 0 unacceptable mismatches
Query 9 0 unacceptable mismatches
Query 10 0 unacceptable mismatches
Query 11 0 unacceptable mismatches
Query 12 0 unacceptable mismatches
Query 13 0 unacceptable mismatches
Query 14 0 unacceptable mismatches
Query 15 0 unacceptable mismatches
Query 16 0 unacceptable mismatches
Query 17 0 unacceptable mismatches
Query 18 0 unacceptable mismatches
Query 19 0 unacceptable mismatches
Query 20 0 unacceptable mismatches
Query 21 0 unacceptable mismatches
Query 22 0 unacceptable mismatches
```

4. Compare query execution

Repeat the previous steps of `SF1`, and load all of the data and scripts into database `SF3` of MonetDB and MySQL separately.

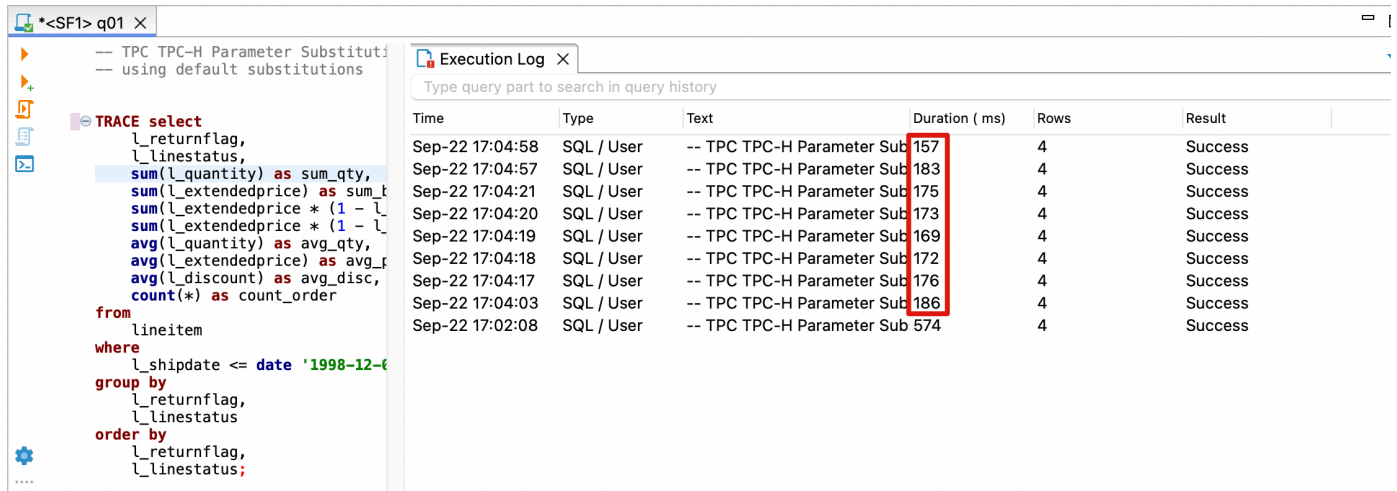
Then loading data process is really slow in MySQL database, and the table `lineitem` costs almost 32min to complete.

Then I run all of these scripts and record the execution time for each. For each query, I add a `TRACE` to obtain the execution time, for example:

```
TRACE select
  l_returnflag,
  l_linestatus,
  sum(l_quantity) as sum_qty,
  sum(l_extendedprice) as sum_base_price,
  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
  sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
  avg(l_quantity) as avg_qty,
  avg(l_extendedprice) as avg_price,
  avg(l_discount) as avg_disc,
  count(*) as count_order
from
  lineitem
where
  l_shipdate <= date '1998-12-01' - interval '90' day (3)
group by
  l_returnflag,
  l_linestatus
order by
  l_returnflag,
```

```
l_linestatus;
```

To run the experiment in **warm memory**, I run each query for more than five times and then calculate the mean execution time (except the first execution time) of the query as is shown in the following figure. The first execution is the **cold run**, and we should set all the experiment in **hot run**.



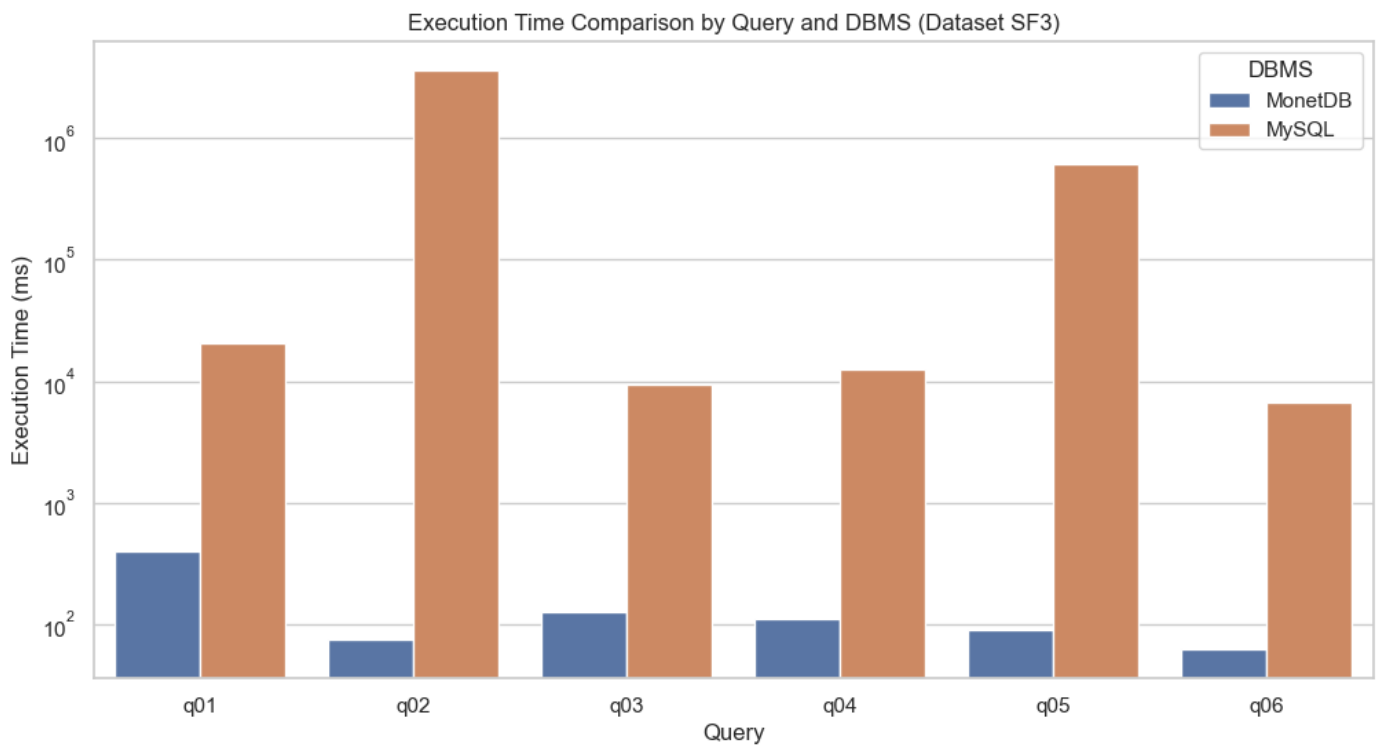
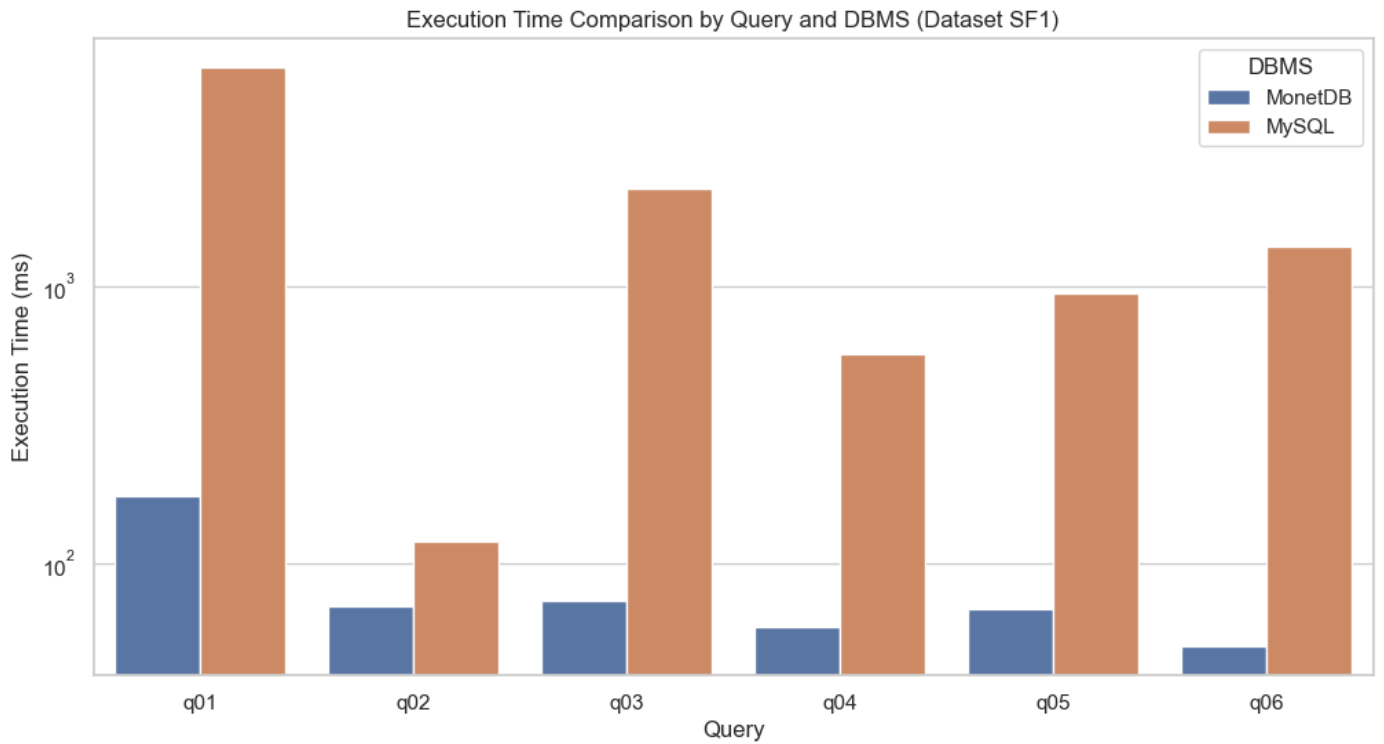
The screenshot shows a database interface with a SQL query editor on the left and an execution log on the right. The query is a complex SELECT statement with multiple aggregations and joins. The execution log shows a series of successful queries with their execution times and row counts. The first execution is highlighted with a red box, indicating it is the cold run.

Time	Type	Text	Duration (ms)	Rows	Result
Sep-22 17:04:58	SQL / User	-- TPC TPC-H Parameter Sub	157	4	Success
Sep-22 17:04:57	SQL / User	-- TPC TPC-H Parameter Sub	183	4	Success
Sep-22 17:04:21	SQL / User	-- TPC TPC-H Parameter Sub	175	4	Success
Sep-22 17:04:20	SQL / User	-- TPC TPC-H Parameter Sub	173	4	Success
Sep-22 17:04:19	SQL / User	-- TPC TPC-H Parameter Sub	169	4	Success
Sep-22 17:04:18	SQL / User	-- TPC TPC-H Parameter Sub	172	4	Success
Sep-22 17:04:17	SQL / User	-- TPC TPC-H Parameter Sub	176	4	Success
Sep-22 17:04:03	SQL / User	-- TPC TPC-H Parameter Sub	186	4	Success
Sep-22 17:02:08	SQL / User	-- TPC TPC-H Parameter Sub	574	4	Success

Some queries may cost many more times, and I will run for fewer times and also calculate the mean execution time. Then I get the execution time table.

Query	DBMS	Dataset	Execution time/ms
q01	MonetDB	SF1	174
q02	MonetDB	SF1	70
q03	MonetDB	SF1	73
q04	MonetDB	SF1	59
q05	MonetDB	SF1	68
q06	MonetDB	SF1	50
q01	MonetDB	SF3	395
q02	MonetDB	SF3	75
q03	MonetDB	SF3	127
q04	MonetDB	SF3	112
q05	MonetDB	SF3	89
q06	MonetDB	SF3	62
q01	MySQL	SF1	6120
q02	MySQL	SF1	119
q03	MySQL	SF1	2250
q04	MySQL	SF1	568
q05	MySQL	SF1	943
q06	MySQL	SF1	1383
q01	MySQL	SF3	20760
q02	MySQL	SF3	longer than 1h, use 3600000
q03	MySQL	SF3	9413
q04	MySQL	SF3	12457
q05	MySQL	SF3	617194
q06	MySQL	SF3	6683

Then I plot the results to compare the execution time with different queries and different DBMS. The value of execution time spread widely, and I use the log scale to put all these data into one figure.



From the figures above, we can find that for MonetDB, the query 1 is the most time consuming. For MySQL, the query performance are very unstable on different datasets. For example, in the SF1 dataset, the query02 consumes least time, but in SF3, the query02 consumes the most time. And there is no specific pattern between the queries.

Compare the DBMS, it is obvious that the performance of MonetDB is far more efficient than MySQL in both dataset. As the dataset size increasing, the performance gap between these two DBMSs become larger. The division of the consuming time on SF3 is 10^3 and on SF1 is only 10. With larger dataset, the advantages of MonetDB is greater than MySQL.

5. Implementation of the queries in Python

I use Python to implement the q01.sql and q06.sql on SF-1 and SF-3 dataset. All of the code can be seen in the attachment named as `implementation.ipynb`.

5.1 Implement q06.sql and q01.sql on SF-1 dataset

SF-1 dataset has a relatively small size, so I implement these sql scripts on SF-1 dataset.

5.1.1 Methods

I use the similar method to implement q06.sql and q01.sql.

- Step1
Load data to a data frame from `lineitem.tbl` file to make sure the data exist in the memory instead of the disk.
- Step 2
Use the filter conditions in data frame which is the same as in the sql script. Filter the data and calculate the final results.
- Step 3
Set start time before the query and end time after the query. The execution time will be the deviation of start time and end time.
- Result
Finally, I get the same results as in the correct result file.
- Execution time
The best execution time of q06.sql implementation is about 60ms which is very close to the MonetDB. And the excution time of q01.sql implementation is about 1580ms.

5.1.2 Verify the results of q01.sql implementation

I store the result of q01.sql implementation in a data frame.

- Step 1
Load the correct results into the data frame from the csv file.
- Step 2
Transfer all of the number elements in both data frames into 2 decimal numbers.
- Step 3
Compare all of the values of the correct results and query results. If all value are equal, I can verity the results of q01.sql implementation is correct.

5.2 Implement q06.sql and q01.sql on SF-3 dataset

For SF-3 dataset, I use the same methods, and also get the correct results as the correct results. The best execution time of q06.sql implementation is about 159ms. And the excution time of q01.sql implementation is about 6427ms.

6. Performance comparison between DBMS and Python implementation

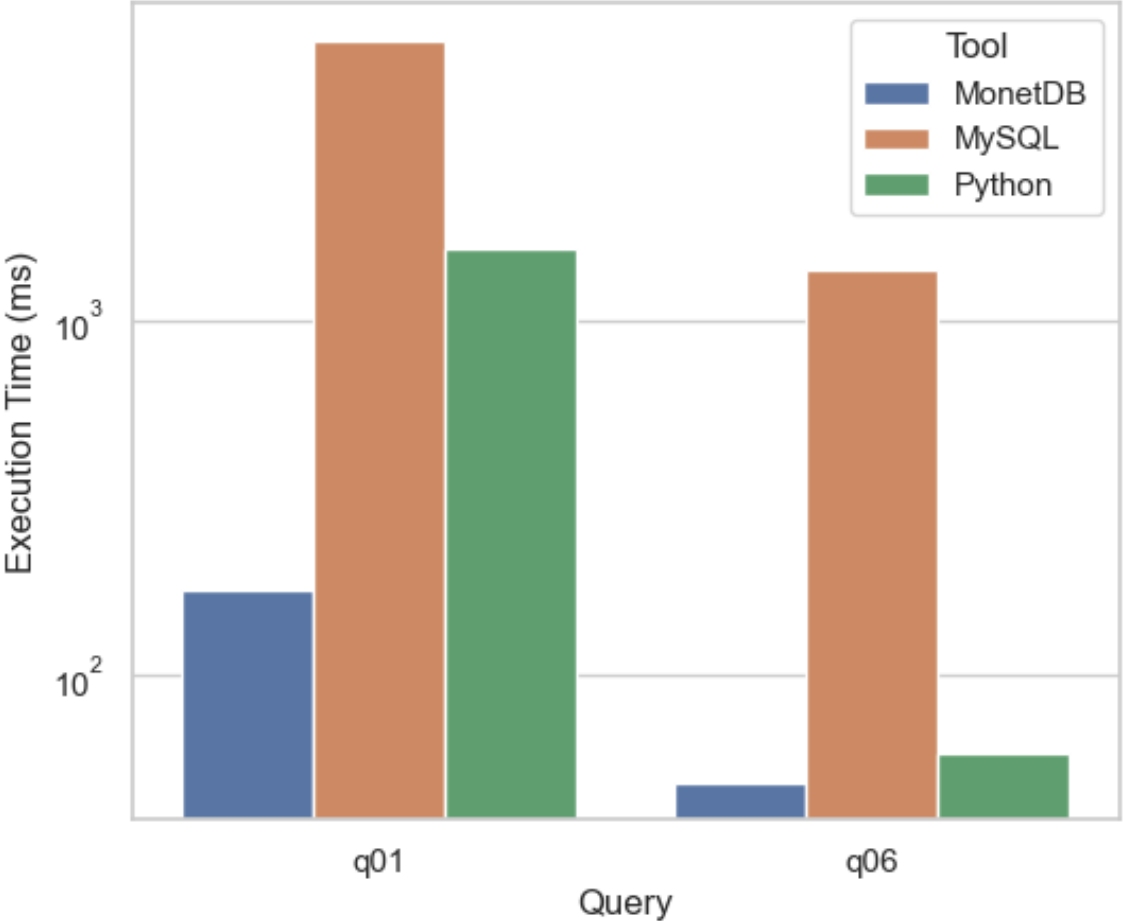
6.1 Visualisation

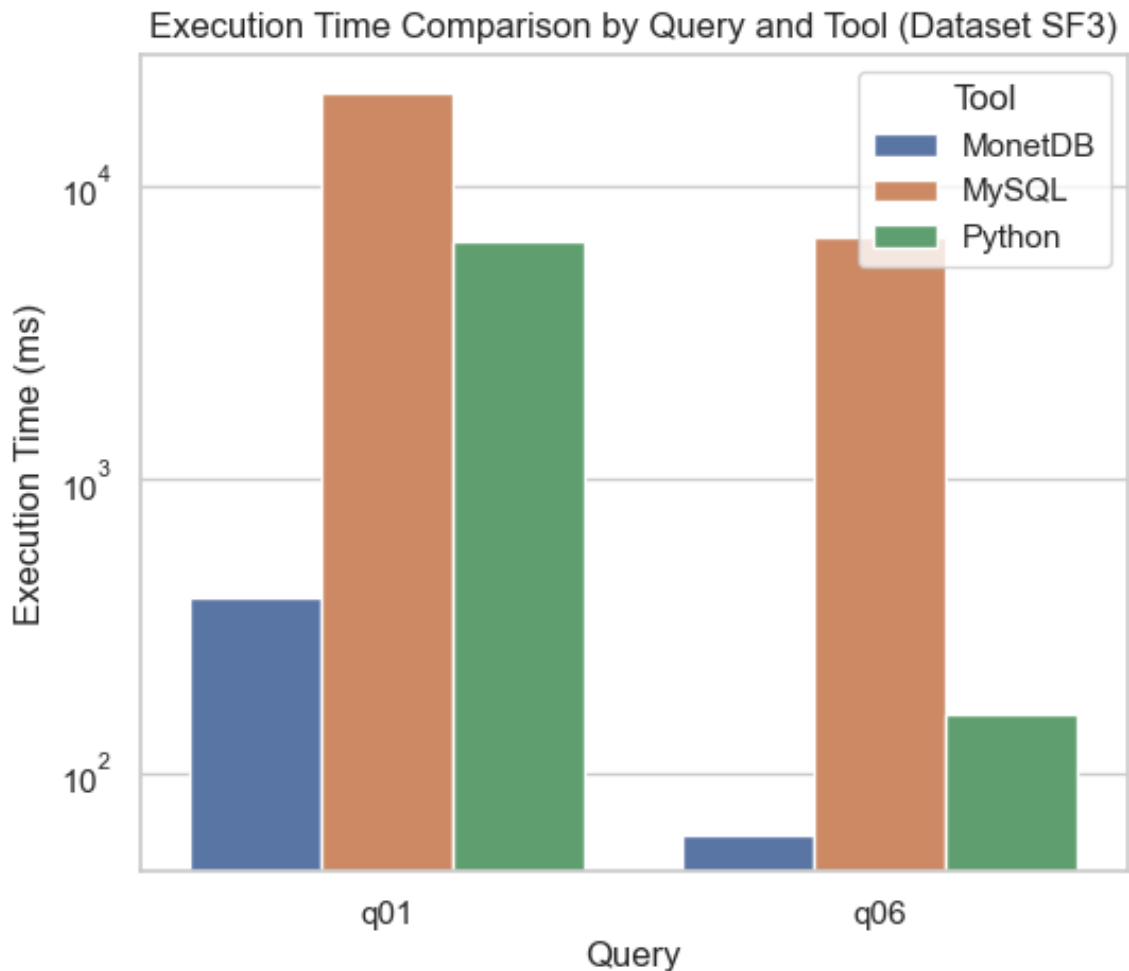
Create the table with all of the previous

Query	Tool	Dataset	Execution time/ms
q01	MonetDB	SF1	174
q06	MonetDB	SF1	50
q01	MonetDB	SF3	395
q06	MonetDB	SF3	62
q01	MySQL	SF1	6120
q06	MySQL	SF1	1383
q01	MySQL	SF3	20760
q06	MySQL	SF3	6683
q01	Python	SF1	1580
q06	Python	SF1	60
q01	Python	SF3	6427
q06	Python	SF3	159

Then I plot all of the execution time with each tool and dataset.

Execution Time Comparison by Query and Tool (Dataset SF1)





6.2 Analysis of the performance

According to the above figures, we can find MonetDB is still the most efficient tool and the performance of Python on q06 are close to MonetDB. But Python perform really bad on q01 which is still better than MySQL.

The bad performance of Python on the q01 has many causes. One of that is the more complex filter conditions in dataframe can largely affect the performance. And at the final step, the result also need to be sorted. The build in sort algorithm of data frame maybe not so efficient.

However, the MySQL is till the worst one compared to Python. I think the main reason can be the data frame treat the data as a matrix, and it also build the column index. In some situation, the data frame can be seen as a column oriented tool which has a better performance than the row oriented tool.

*Appendix

Some notes for the configuration and commands

*.1 MonetDB

*.1.1 Configuration

Description	Configuration
Configuration file location	~/.monetdb
Default username/password	monetdb/monetdb
Port	54321
Language	sql

*.1.2 Commands

Description	Command
Create workspace	<code>monetdbd create ~/my-dbfarm</code>
Check configuration	<code>monetdbd get all ~/my-dbfarm</code>
Start server	<code>monetdbd start ~/my-dbfarm</code>
Create database	<code>monetdb create my-first-db</code>
Start database	<code>monetdb start my-first-db</code>
Check database status	<code>monetdb status</code>
Release database (if locked)	<code>monetdb release my-first-db</code>
Connect to the database	<code>mclient -dmy-first-db</code>
Stop monetdb daemon process completely	<code>monetdbd stop ~/my-dbfarm</code>