Big Data Assignment 2: Benchmarking

2. Database initialization and loading:

a. time mysql -uroot -p -D flight --local-infile=1 < air_ddl.sql

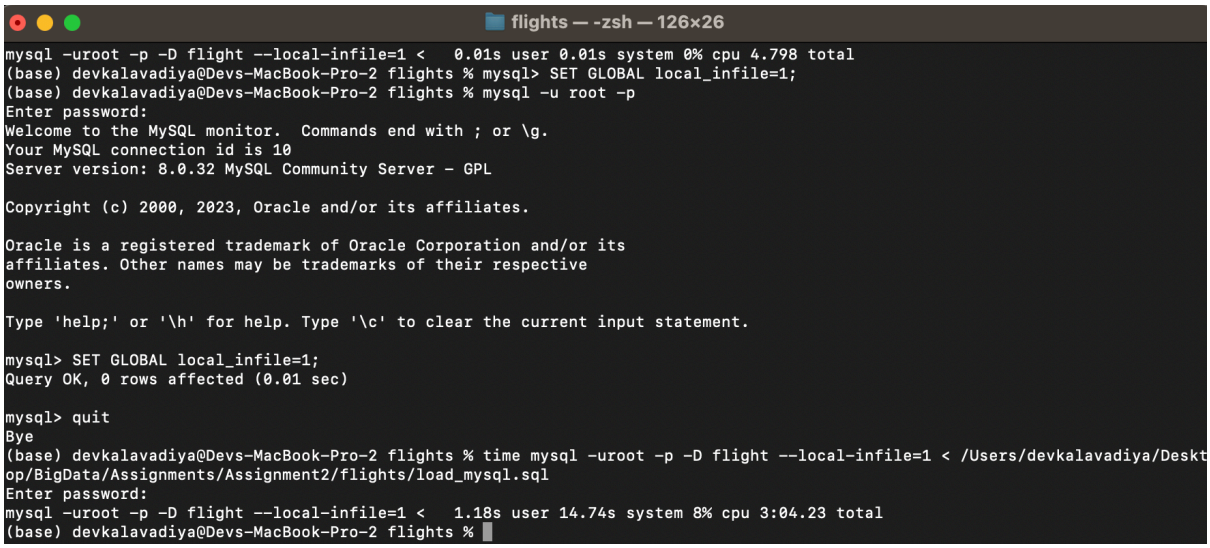
```
mysql> CREATE DATABASE flight;
Query OK, 1 row affected (0.03 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| classicmodel       |
| flight             |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| university         |
+--------------------+
7 rows in set (0.06 sec)

mysql>
mysql> time mysql -uroot -p -D flight --local-infile=1 < air_ddl.sql
    ->
    ->
    -> ^C
mysql> ^C
mysql> ^C
mysql> ^C
mysql>
zsh: suspended  mysql -u root -p
(base) devkalavadiya@Devs-MacBook-Pro-2 flights % time mysql -uroot -p -D flight --local-infile=1 < air_ddl.s
ql

Enter password:
mysql -uroot -p -D flight --local-infile=1 < air_ddl.sql  0.01s user 0.01s system 0% cpu 4.213 total
(base) devkalavadiya@Devs-MacBook-Pro-2 flights %
```

b. time mclient -u monetdb -d flight < air_ddl.sql

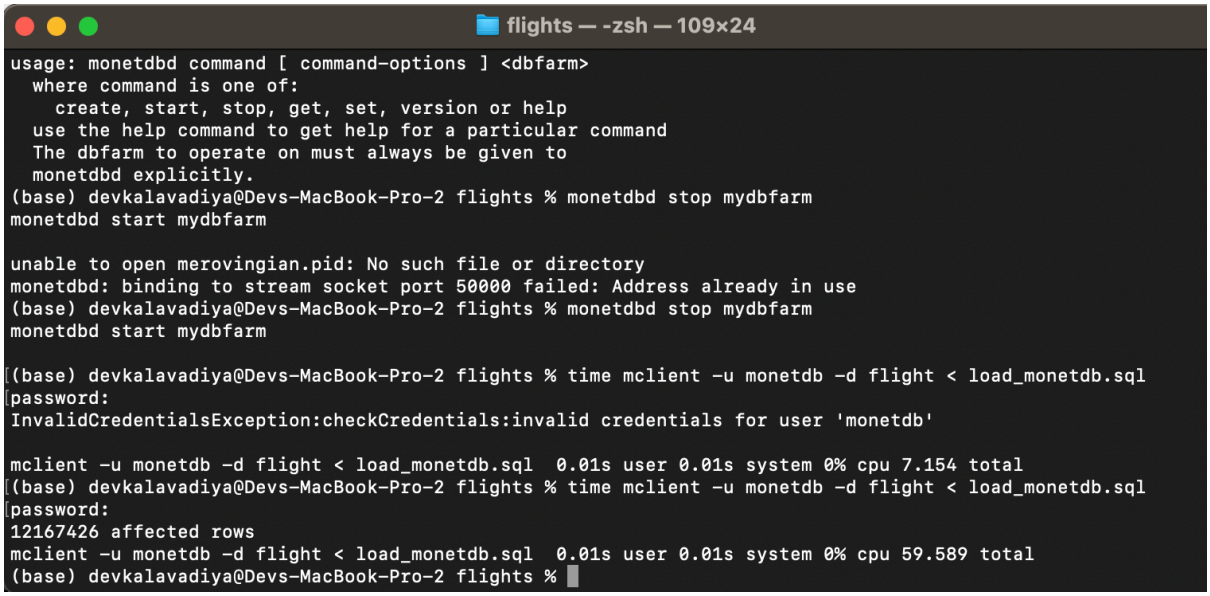
```
flights — -zsh — 109×24
Last login: Fri Mar  3 13:45:52 on ttys000
(base) devkalavadiya@Devs-MacBook-Pro-2 ~ % cd Desktop/Big\ Data/Assignments/Assignment2
(base) devkalavadiya@Devs-MacBook-Pro-2 Assignment2 % ls
Assignment 2_ Benchmarking.pdf          flights-20230228T113927Z-001.zip
Big Data Assignment 2.docx              mydbfarm
flights                                 ~$g Data Assignment 2.docx
(base) devkalavadiya@Devs-MacBook-Pro-2 Assignment2 % cd flights
(base) devkalavadiya@Devs-MacBook-Pro-2 flights % time mclient -u monetdb -d flight < air_ddl.sql
password:
operation successful
mclient -u monetdb -d flight < air_ddl.sql  0.00s user 0.01s system 0% cpu 3.703 total
(base) devkalavadiya@Devs-MacBook-Pro-2 flights %
```

B. Data Loading

a. time mysql -uroot -p -D flight --local-infile=1 < load_mysql.sql

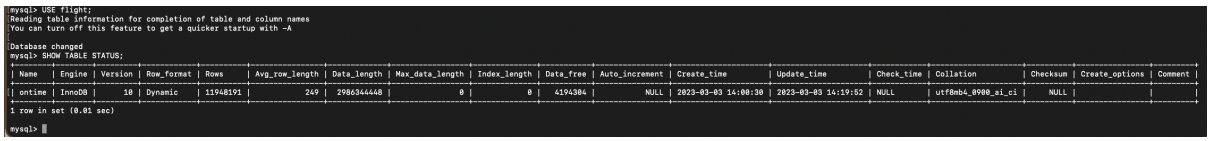


b. time mclient -u monetdb -d flight < load_monetdb.sql



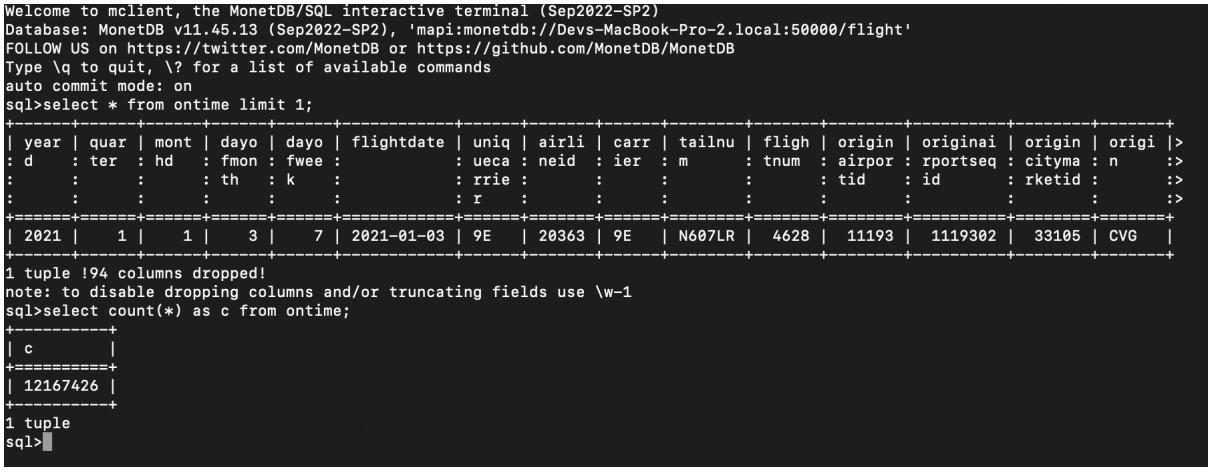
- Get the table status in both system: **(0.5 point)**

  - MySQL



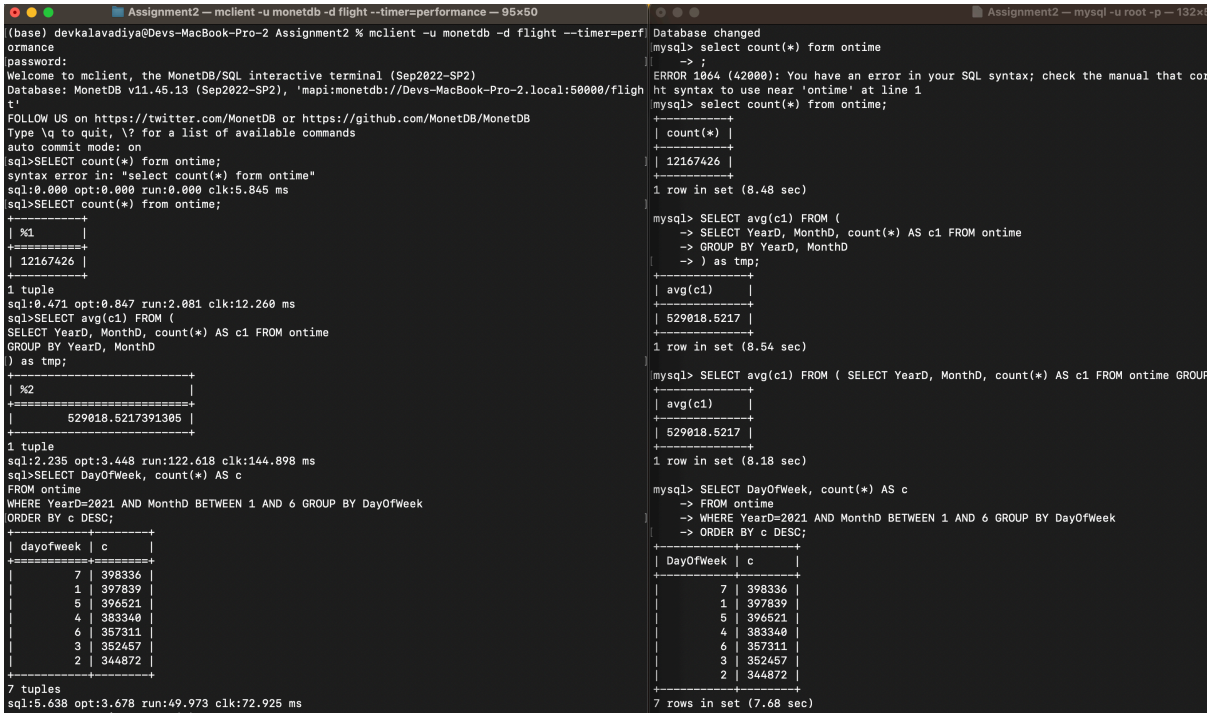
```
mysql> USE flight;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SHOW TABLE STATUS;
+--------+--------+---------+------------+---------+----------------+-------------+-----------------+-----------+----------------+---------------------+---------------------+---------------------+------------+--------------------+----------+----------------+---------+
| Name   | Engine | Version | Row_format | Rows    | Avg_row_length | Data_length | Max_data_length | Index_length | Data_free | Auto_increment | Create_time         | Update_time         | Check_time | Collation          | Checksum | Create_options | Comment |
+--------+--------+---------+------------+---------+----------------+-------------+-----------------+--------------+-----------+----------------+---------------------+---------------------+------------+--------------------+----------+----------------+---------+
| ontime | InnoDB |      10 | Dynamic    | 11948191 |            249 |  2986344448 |               0 |            0 |   4194304 |           NULL | 2023-03-03 14:00:30 | 2023-03-03 14:19:52 | NULL       | utf8mb4_0900_ai_ci |     NULL |                |         |
+--------+--------+---------+------------+---------+----------------+-------------+-----------------+--------------+-----------+----------------+---------------------+---------------------+------------+--------------------+----------+----------------+---------+
1 row in set (0.01 sec)

mysql>
```

  - MonetDB



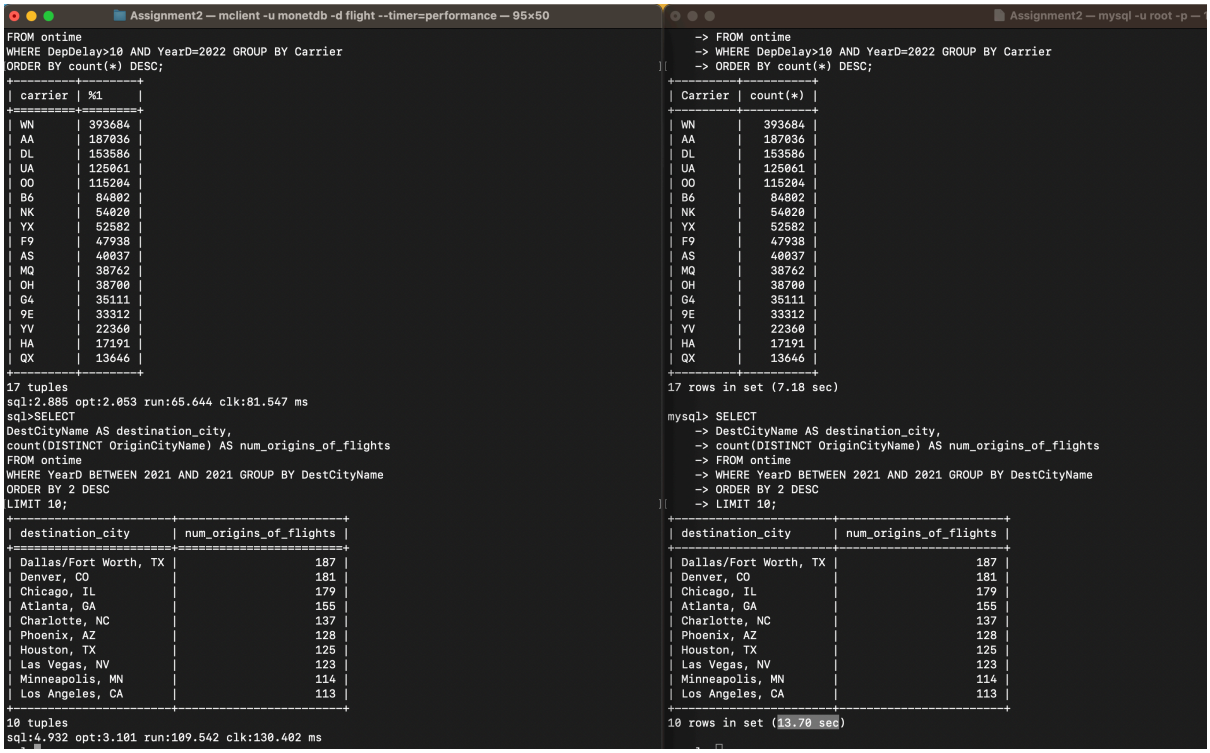
```
Welcome to mclient, the MonetDB/SQL interactive terminal (Sep2022-SP2)
Database: MonetDB v11.45.13 (Sep2022-SP2), 'mapi:monetdb://Devs-MacBook-Pro-2.local:50000/flight'
FOLLOW US on https://twitter.com/MonetDB or https://github.com/MonetDB/MonetDB
Type \q to quit, \? for a list of available commands
auto commit mode: on
sql>select * from ontime limit 1;
+------+------+------+------+------+------------+------+-------+------+--------+-------+--------+----------+--------+-------+
| year | quar | mont | dayo | dayo | flightdate | uniq | airli | carr | tailnu | fligh | origin | originai | origin | origi |>
: d    : ter  : hd   : fmon : fwee :            : ueca : neid  : ier  : m      : tnum  : airpor : rportseq : cityma : n     :>
:      :      :      : th   : k    :            : rrie :       :      :        :       : tid    : id       : rketid :       :>
:      :      :      :      :      :            : r    :       :      :        :       :        :          :        :       :>
+======+======+======+======+======+============+======+=======+======+========+=======+========+==========+========+=======+
| 2021 |    1 |    1 |    3 |    7 | 2021-01-03 | 9E   | 20363 | 9E   | N607LR |  4628 |  11193 |  1119302 |  33105 | CVG   |
+------+------+------+------+------+------------+------+-------+------+--------+-------+--------+----------+--------+-------+
1 tuple !94 columns dropped!
note: to disable dropping columns and/or truncating fields use \w-1
sql>select count(*) as c from ontime;
+----------+
| c        |
+==========+
| 12167426 |
+----------+
1 tuple
sql>
```

Query Execution Screenshots



```
Assignment2 — mclient -u monetdb -d flight --timer=performance — 95×50
(base) devkalavadiya@Devs-MacBook-Pro-2 Assignment2 % mclient -u monetdb -d flight --timer=perf
ormance
password:
Welcome to mclient, the MonetDB/SQL interactive terminal (Sep2022-SP2)
Database: MonetDB v11.45.13 (Sep2022-SP2), 'mapi:monetdb://Devs-MacBook-Pro-2.local:50000/fligh
t'
FOLLOW US on https://twitter.com/MonetDB or https://github.com/MonetDB/MonetDB
Type \q to quit, \? for a list of available commands
auto commit mode: on
sql>SELECT count(*) form ontime;
syntax error in: "select count(*) form ontime"
sql:0.000 opt:0.000 run:0.000 clk:5.845 ms
sql>SELECT count(*) from ontime;
+----------+
| %1       |
+==========+
| 12167426 |
+----------+
1 tuple
sql:0.471 opt:0.847 run:2.081 clk:12.260 ms
sql>SELECT avg(c1) FROM (
SELECT YearD, MonthD, count(*) AS c1 FROM ontime
GROUP BY YearD, MonthD
) as tmp;
+--------------------------+
| %2                       |
+==========================+
|        529018.5217391305 |
+--------------------------+
1 tuple
sql:2.235 opt:3.448 run:122.618 clk:144.898 ms
sql>SELECT DayOfWeek, count(*) AS c
FROM ontime
WHERE YearD=2021 AND MonthD BETWEEN 1 AND 6 GROUP BY DayOfWeek
ORDER BY c DESC;
+-----------+--------+
| dayofweek | c      |
+===========+========+
|         7 | 398336 |
|         1 | 397839 |
|         5 | 396521 |
|         4 | 383340 |
|         6 | 357311 |
|         3 | 352457 |
|         2 | 344872 |
+-----------+--------+
7 tuples
sql:5.638 opt:3.678 run:49.973 clk:72.925 ms
```

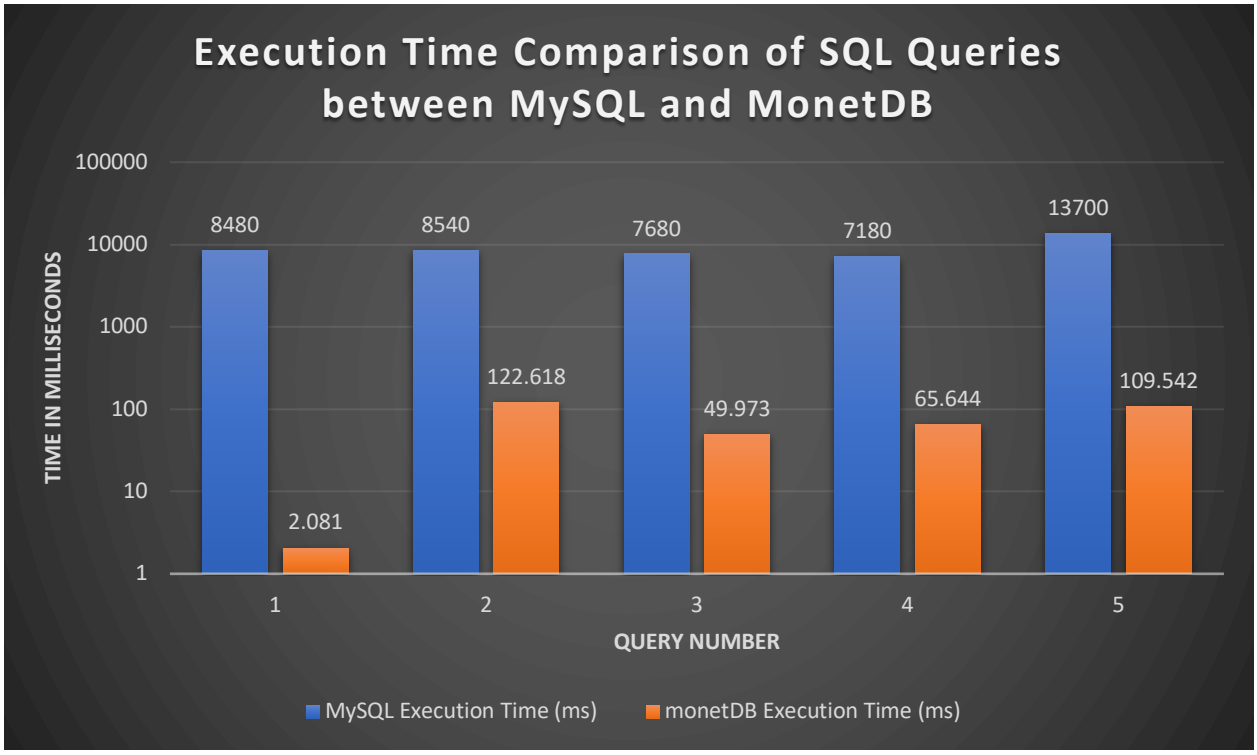
```
Assignment2 — mysql -u root -p — 132×
Database changed
mysql> select count(*) form ontime
    -> ;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that co
ht syntax to use near 'ontime' at line 1
mysql> select count(*) from ontime;
+----------+
| count(*) |
+----------+
| 12167426 |
+----------+
1 row in set (8.48 sec)

mysql> SELECT avg(c1) FROM (
    -> SELECT YearD, MonthD, count(*) AS c1 FROM ontime
    -> GROUP BY YearD, MonthD
    -> ) as tmp;
+-------------+
| avg(c1)     |
+-------------+
| 529018.5217 |
+-------------+
1 row in set (8.54 sec)

mysql> SELECT avg(c1) FROM ( SELECT YearD, MonthD, count(*) AS c1 FROM ontime GROUP
+-------------+
| avg(c1)     |
+-------------+
| 529018.5217 |
+-------------+
1 row in set (8.18 sec)

mysql> SELECT DayOfWeek, count(*) AS c
    -> FROM ontime
    -> WHERE YearD=2021 AND MonthD BETWEEN 1 AND 6 GROUP BY DayOfWeek
    -> ORDER BY c DESC;
+-----------+--------+
| DayOfWeek | c      |
+-----------+--------+
|         7 | 398336 |
|         1 | 397839 |
|         5 | 396521 |
|         4 | 383340 |
|         6 | 357311 |
|         3 | 352457 |
|         2 | 344872 |
+-----------+--------+
7 rows in set (7.68 sec)
```



```
Assignment2 — mclient -u monetdb -d flight --timer=performance — 95×50
FROM ontime
WHERE DepDelay>10 AND YearD=2022 GROUP BY Carrier
ORDER BY count(*) DESC;
+---------+--------+
| carrier | %1     |
+=========+========+
| WN      | 393684 |
| AA      | 187036 |
| DL      | 153586 |
| UA      | 125061 |
| OO      | 115204 |
| B6      |  84802 |
| NK      |  54020 |
| YX      |  52582 |
| F9      |  47938 |
| AS      |  40037 |
| MQ      |  38762 |
| OH      |  38700 |
| G4      |  35111 |
| 9E      |  33312 |
| YV      |  22360 |
| HA      |  17191 |
| QX      |  13646 |
+---------+--------+
17 tuples
sql:2.885 opt:2.053 run:65.644 clk:81.547 ms
sql>SELECT
DestCityName AS destination_city,
count(DISTINCT OriginCityName) AS num_origins_of_flights
FROM ontime
WHERE YearD BETWEEN 2021 AND 2021 GROUP BY DestCityName
ORDER BY 2 DESC
LIMIT 10;
+----------------------+-------------------------+
| destination_city     | num_origins_of_flights  |
+======================+=========================+
| Dallas/Fort Worth, TX |                     187 |
| Denver, CO           |                     181 |
| Chicago, IL          |                     179 |
| Atlanta, GA          |                     155 |
| Charlotte, NC        |                     137 |
| Phoenix, AZ          |                     128 |
| Houston, TX          |                     125 |
| Las Vegas, NV        |                     123 |
| Minneapolis, MN      |                     114 |
| Los Angeles, CA      |                     113 |
+----------------------+-------------------------+
10 tuples
sql:4.932 opt:3.101 run:109.542 clk:130.402 ms
```

```
Assignment2 — mysql -u root -p —
    -> FROM ontime
    -> WHERE DepDelay>10 AND YearD=2022 GROUP BY Carrier
    -> ORDER BY count(*) DESC;
+---------+----------+
| Carrier | count(*) |
+---------+----------+
| WN      |   393684 |
| AA      |   187036 |
| DL      |   153586 |
| UA      |   125061 |
| OO      |   115204 |
| B6      |    84802 |
| NK      |    54020 |
| YX      |    52582 |
| F9      |    47938 |
| AS      |    40037 |
| MQ      |    38762 |
| OH      |    38700 |
| G4      |    35111 |
| 9E      |    33312 |
| YV      |    22360 |
| HA      |    17191 |
| QX      |    13646 |
+---------+----------+
17 rows in set (7.18 sec)

mysql> SELECT
    -> DestCityName AS destination_city,
    -> count(DISTINCT OriginCityName) AS num_origins_of_flights
    -> FROM ontime
    -> WHERE YearD BETWEEN 2021 AND 2021 GROUP BY DestCityName
    -> ORDER BY 2 DESC
    -> LIMIT 10;
+----------------------+-------------------------+
| destination_city     | num_origins_of_flights  |
+----------------------+-------------------------+
| Dallas/Fort Worth, TX |                     187 |
| Denver, CO           |                     181 |
| Chicago, IL          |                     179 |
| Atlanta, GA          |                     155 |
| Charlotte, NC        |                     137 |
| Phoenix, AZ          |                     128 |
| Houston, TX          |                     125 |
| Las Vegas, NV        |                     123 |
| Minneapolis, MN      |                     114 |
| Los Angeles, CA      |                     113 |
+----------------------+-------------------------+
10 rows in set (13.70 sec)
```

Query No	MySQL Execution Time	MonetDB Execution Time
1	8480 ms	2.081 ms
2	8540 ms	122.618 ms
3	7680 ms	49.973 ms
4	7180 ms	65.644 ms
5	13700 ms	109.542 ms



Based on the results of the queries, it is clear that MonetDB outperforms MySQL in terms of execution time for all 5 queries. The difference in performance is particularly noticeable in queries 1, 2 and 5, where MonetDB's execution time is significantly lower than MySQL's.

The difference in performance can be attributed to the different data storage models used by the two systems. MonetDB uses a column-oriented storage model, which is optimized for analytical queries that typically involve aggregations and scans over large data sets. However, MySQL uses a row-oriented storage model, which is optimized for transactional queries that typically involve retrieving a small number of rows from a table.

Overall, the performance difference between MonetDB and MySQL highlights the importance of selecting a database system that is optimized for the specific use case. If the focus is on analytical queries that involve aggregations and scans over large data sets, a column-oriented database system like MonetDB may be a better choice. However, if the focus is on transactional queries that involve retrieving a small number of rows from a table, a row-oriented database system like MySQL may be a better choice.

Based on the results, query 5 has the largest difference in execution time between MySQL and MonetDB. To improve the performance of MySQL for this query, here are some possible strategies:

- Index optimization: By ensuring that the relevant columns used in the query are properly indexed. In particular, the columns YearD, DestCityName, and OriginCityName could benefit from indexing.

  ```
  CREATE INDEX idx_YearD ON ontime (YearD);
  ```

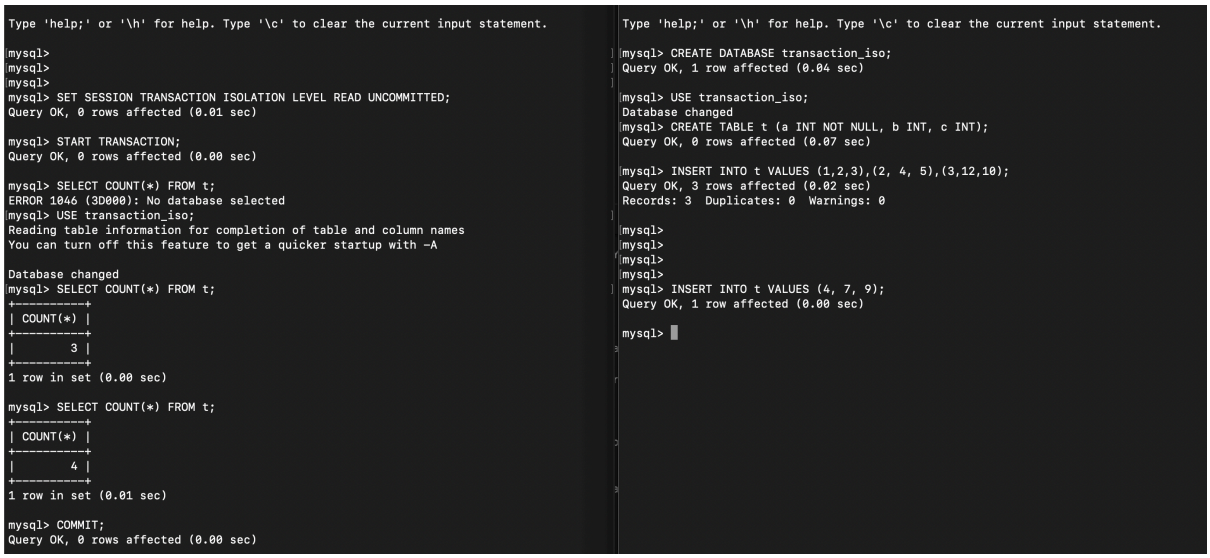
- Query optimization: Review the query and look for ways to simplify or optimize it. For example, instead of using BETWEEN 2021 AND 2021 in the WHERE clause, it could be simplified to YearD = 2021.

Part 2

Creating the new database

```
mysql> CREATE DATABASE transaction_iso;
Query OK, 1 row affected (0.04 sec)

mysql> USE transaction_iso;
Database changed
mysql> CREATE TABLE t (a INT NOT NULL, b INT, c INT);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO t VALUES (1,2,3),(2, 4, 5),(3,12,10);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
```

● *Phantom read.*



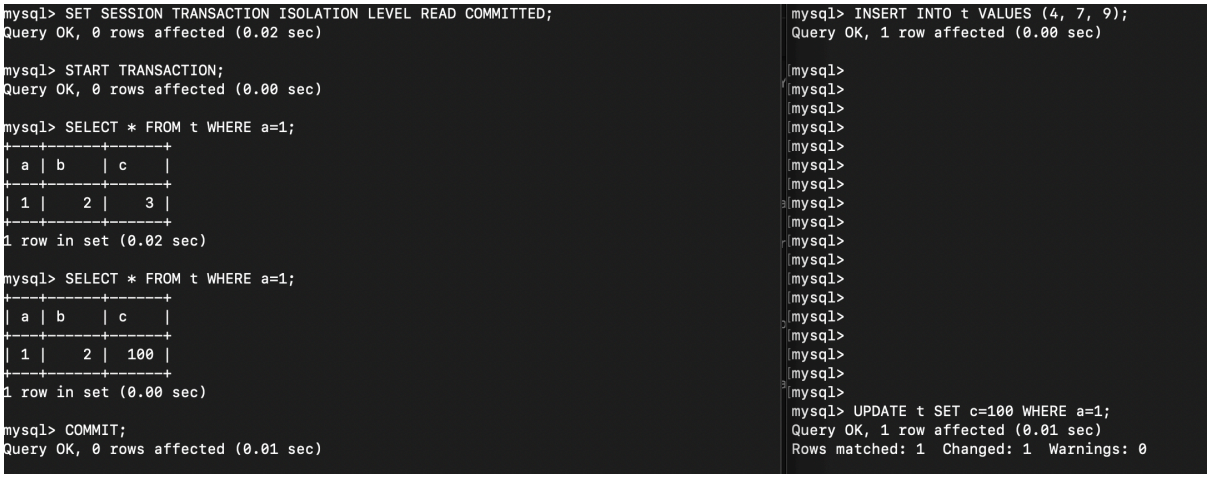
```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
mysql>
mysql>
mysql> SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
Query OK, 0 rows affected (0.01 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT COUNT(*) FROM t;
ERROR 1046 (3D000): No database selected
mysql> USE transaction_iso;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> SELECT COUNT(*) FROM t;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM t;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.01 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE transaction_iso;
Query OK, 1 row affected (0.04 sec)

mysql> USE transaction_iso;
Database changed
mysql> CREATE TABLE t (a INT NOT NULL, b INT, c INT);
Query OK, 0 rows affected (0.07 sec)

mysql> INSERT INTO t VALUES (1,2,3),(2, 4, 5),(3,12,10);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql>
mysql>
mysql>
mysql> INSERT INTO t VALUES (4, 7, 9);
Query OK, 1 row affected (0.00 sec)

mysql>
```

Count is different after inserting in a separate SQL client (on the right) even though the left terminal had a START TRANSACTION, and the commit was made after.

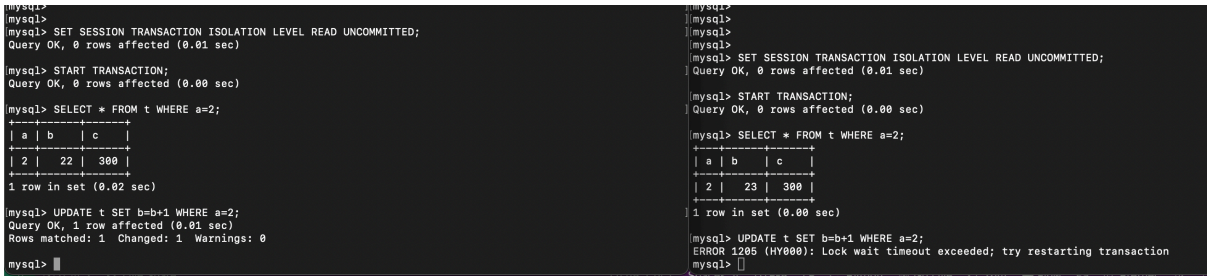
In this example we can see that Terminal 1 starts a transaction and reads the data from the t table where the count is 3. While Terminal 1's transaction is still open, Terminal 2 inserts a new row into the t table, which causes the count to increment to 4. Terminal 1 then makes another read to count the values in t, where it finds the count to be 4 instead of 3, resulting in a phantom read concurrency problem.

● *Unrepeatable read.*



In this scenario, the unrepeatable read problem is simulated by allowing a transaction to read committed data that has been modified by another transaction, resulting in the transaction seeing different data in subsequent reads. As seen in the image, there are 2 Terminals and Terminal 1 starts first as it reads the tuple where a=1. At this point, Terminal 2 updates the value of c to 100 in that same tuple. Hence when Terminal 1 comes to make a second read for the same tuple, it gets 100, resulting in a unrepeatable read concurrency problem.

● *Dirty read.*



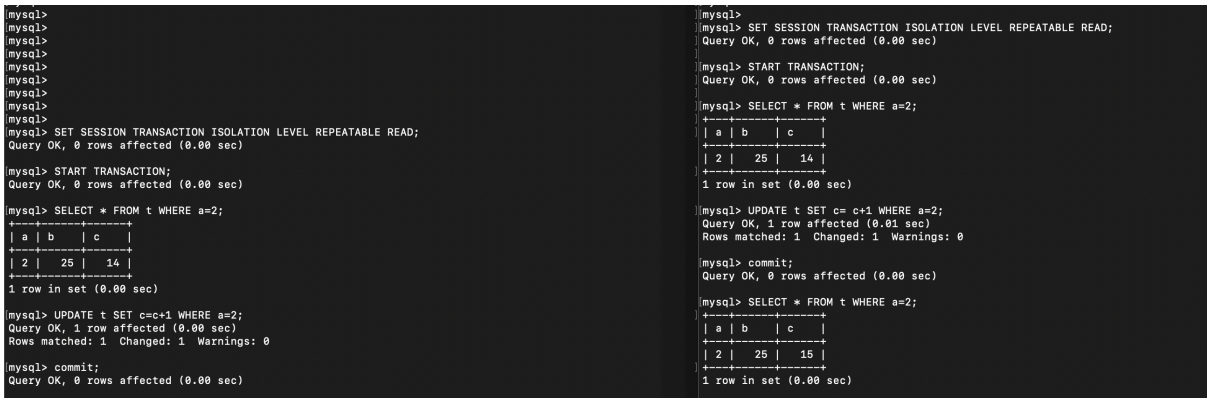
This is an example of a dirty read because Terminal 2 reads uncommitted data that was modified by an uncommitted transaction in Terminal 1.

In Terminal 1, the transaction reads the row where a=2, and then updates the value of b. The update operation has not yet been committed, so the value of b has not yet been changed in the database.

In Terminal 2, the transaction also reads the row where a=2, and since it is set to READ UNCOMMITTED isolation level, it reads the uncommitted data modified by Terminal 1, which is why it sees the value of b as 23 instead of 22.

However, when Terminal 2 tries to update the same row, it encounters a lock wait timeout, because the row has been locked by the uncommitted transaction in Terminal 1. This means that Terminal 2's transaction cannot proceed until Terminal 1's transaction is either committed or rolled back. If Terminal 1 rolls back its transaction, the update in Terminal 2 will never be committed, and Terminal 2 will have read uncommitted data that was never committed.

● *Lost update.*



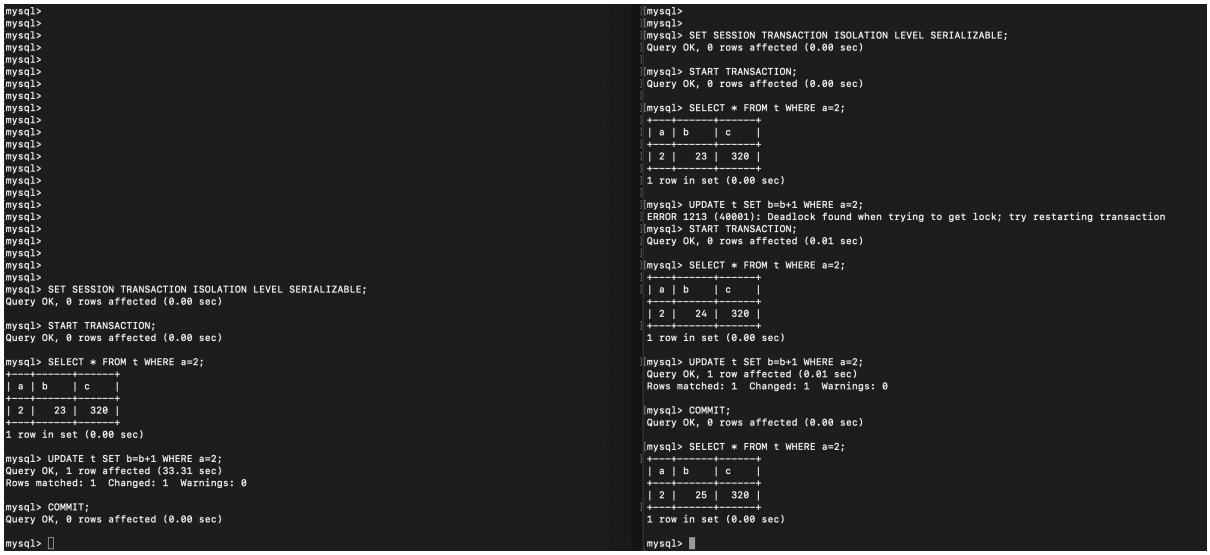
```
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE a=2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 |   25 |   14 |
+---+------+------+
1 row in set (0.00 sec)

mysql> UPDATE t SET c=c+1 WHERE a=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
mysql> SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE a=2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 |   25 |   14 |
+---+------+------+
1 row in set (0.00 sec)

mysql> UPDATE t SET c= c+1 WHERE a=2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE a=2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 |   25 |   15 |
+---+------+------+
1 row in set (0.00 sec)
```

14 + 1 + 1 = 15?

This is an instance of a lost update. In this scenario, both Terminal 1 and Terminal 2 start a transaction and update the same row and increment c by 1. Terminal 1 adds 1 to 14, commits the transaction. After that, Terminal 2 increments the same value by adding 1 and commits the transaction. Since the second update overwrote the value set by the first update, the changes made by Terminal 1 are lost. Thus, the final value of c is 15 instead of 16.

Solution To Lost Update - Serializable



```
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE a=2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 |   23 |  320 |
+---+------+------+
1 row in set (0.00 sec)

mysql> UPDATE t SET b=b+1 WHERE a=2;
Query OK, 1 row affected (33.31 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

```
mysql>
mysql>
mysql> SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE a=2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 |   23 |  320 |
+---+------+------+
1 row in set (0.00 sec)

mysql> UPDATE t SET b=b+1 WHERE a=2;
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM t WHERE a=2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 |   24 |  320 |
+---+------+------+
1 row in set (0.00 sec)

mysql> UPDATE t SET b=b+1 WHERE a=2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE a=2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 |   25 |  320 |
+---+------+------+
1 row in set (0.00 sec)

mysql>
```