CS487/587 Project Part III

Mingjue Wang & Brent Daniels-Soles

DBMS Systems - What

We chose to compare:

- PostgreSQL
- MySQL

DBMS Systems - Why?

- PostgreSQL
  - Not as Popular as MySQL
  - Open Source
  - “Battle Tested”
  - Has a rich feature set

- MySQL
  - More Popular than Postgres
  - (Kinda) Open Source
    - There is an Enterprise version, along with Community version.
  - Has less features, but has all the essentials to build a product

DBMS Systems - Why?

- Commonalities
  - They are both used heavily in industry
  - They both have a similar syntax/semantics

- Given all of the previous points and their similarities, **we wanted to see how different they are when it comes to dealing with larger amounts of data.**
- Our hypothesis was MySQL would do better with smaller amounts of data, and Postgres would perform better with our JOIN tests on large amounts of data.

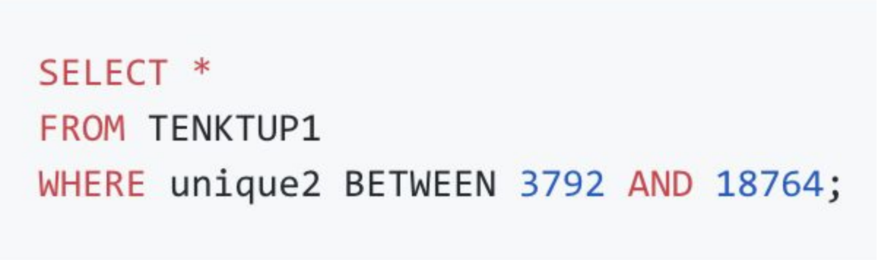
Benchmark Approach

- Goals
  - Keep systems as ‘vanilla’ as possible
    - Reason: most people use basic install and then start hacking away
  - Give each system a large amount of data to sift through
    - Created a 1,000,000 tuple table to mock this
  - See how each system performs with JOIN operations
    - Common, yet expensive
    - Want to determine which may be the best when heavy/many transactions are involved.
  - Test each query at least 5 times
    - In order to get a general idea of average execution time.

Experiments - Warm Up

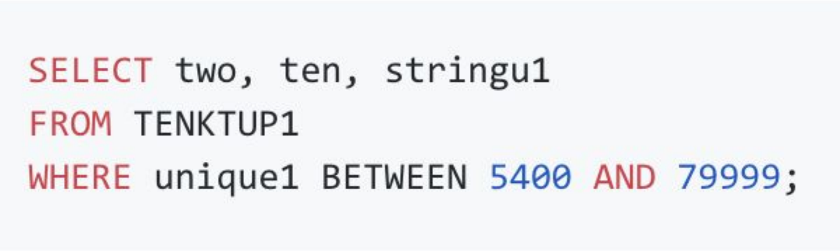
- Queries ran at the start:

Query 1)



```sql
SELECT *
FROM TENKTUP1
WHERE unique2 BETWEEN 3792 AND 18764;
```

Query 2)

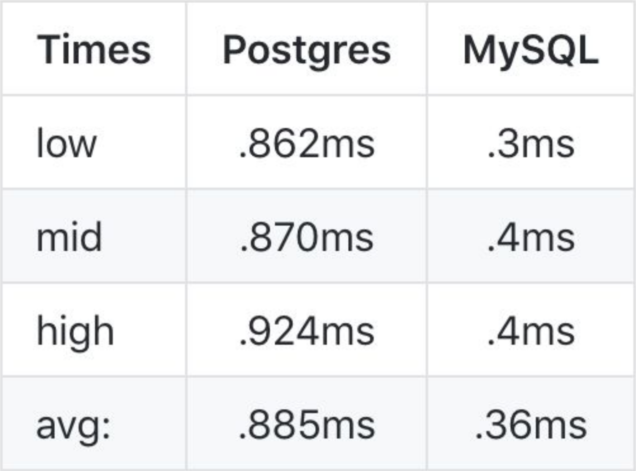


```sql
SELECT two, ten, stringu1
FROM TENKTUP1
WHERE unique1 BETWEEN 5400 AND 79999;
```

Experiments - Warm Up - Why?

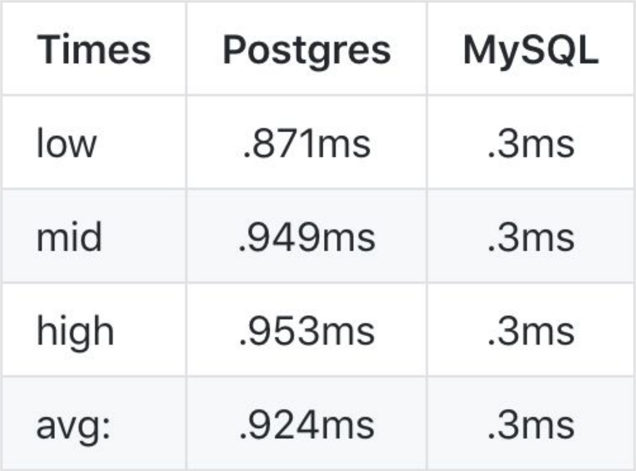
- Wanted to get a “baseline” as to how each of these systems perform with scanning tables.
- Results:

Query 1)



Times	Postgres	MySQL
low	.862ms	.3ms
mid	.870ms	.4ms
high	.924ms	.4ms
avg:	.885ms	.36ms

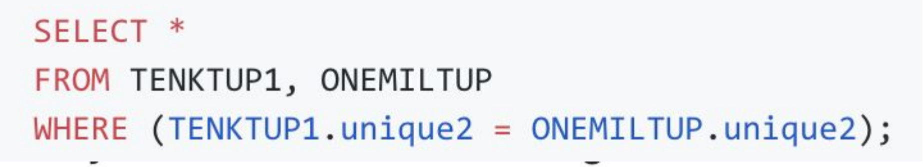
Query 2)



Times	Postgres	MySQL
low	.871ms	.3ms
mid	.949ms	.3ms
high	.953ms	.3ms
avg:	.924ms	.3ms

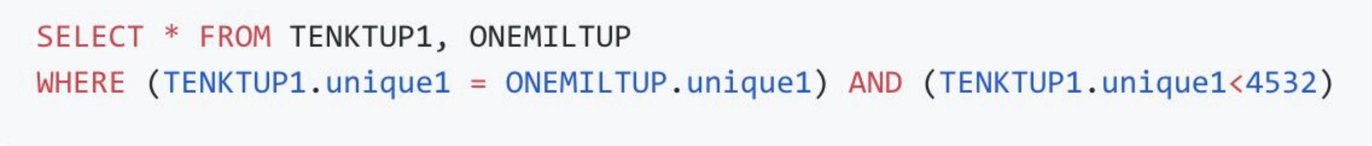
Experiments - JOIN’s

Query 3)



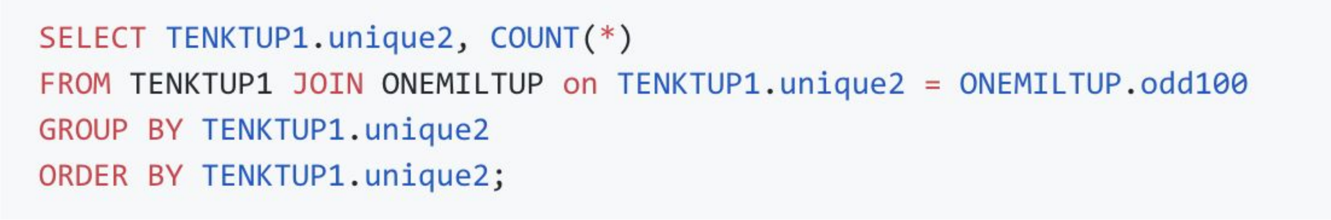
```
SELECT *
FROM TENKTUP1, ONEMILTUP
WHERE (TENKTUP1.unique2 = ONEMILTUP.unique2);
```

Query 4)



```
SELECT * FROM TENKTUP1, ONEMILTUP
WHERE (TENKTUP1.unique1 = ONEMILTUP.unique1) AND (TENKTUP1.unique1<4532)
```

Query 5)

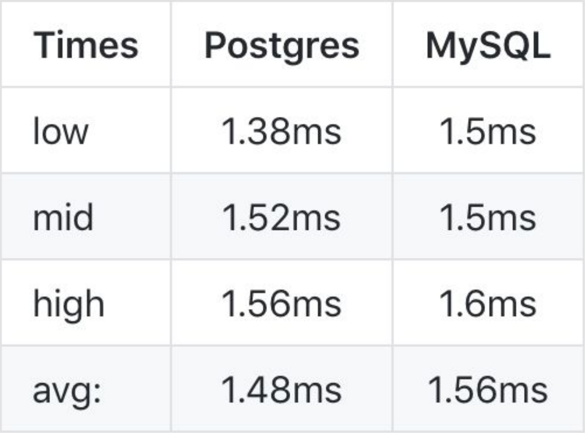


```
SELECT TENKTUP1.unique2, COUNT(*)
FROM TENKTUP1 JOIN ONEMILTUP on TENKTUP1.unique2 = ONEMILTUP.odd100
GROUP BY TENKTUP1.unique2
ORDER BY TENKTUP1.unique2;
```

Experiments - JOIN’s - Why?

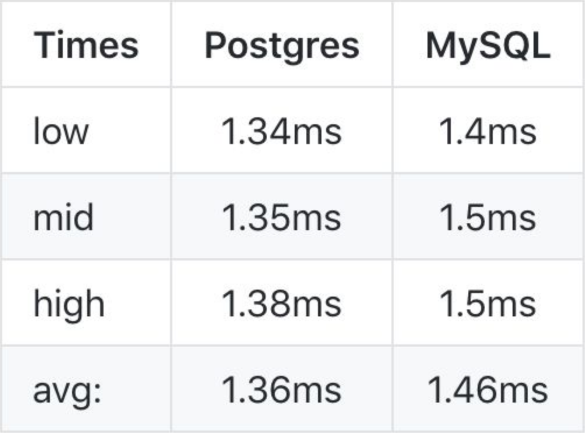
- Since RDBMS’s in production utilize JOIN’s heavily, it would be good to see how each system performs with “production-esque” data.
- Results:

Query 3)



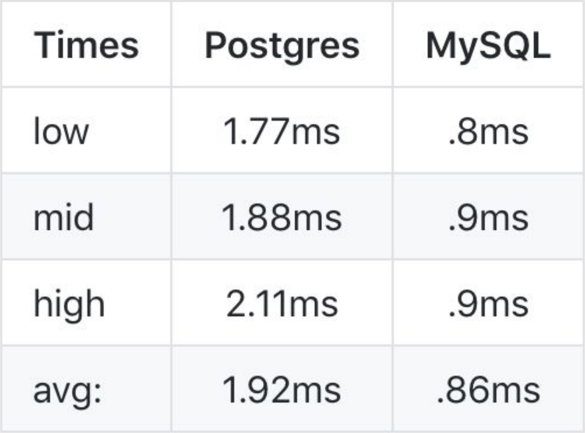
Times	Postgres	MySQL
low	1.38ms	1.5ms
mid	1.52ms	1.5ms
high	1.56ms	1.6ms
avg:	1.48ms	1.56ms

Query 4)



Times	Postgres	MySQL
low	1.34ms	1.4ms
mid	1.35ms	1.5ms
high	1.38ms	1.5ms
avg:	1.36ms	1.46ms

Query 5)



Times	Postgres	MySQL
low	1.77ms	.8ms
mid	1.88ms	.9ms
high	2.11ms	.9ms
avg:	1.92ms	.86ms

Results

- Were these expected...?
    - Kind of…
- Expected MySQL to be more performant with simpler queries, and Postgres to perform better with JOIN’s
    - Reason: MySQL implements only the Nested Loop Join algorithm for JOINS, which is historically slow (there’s a twist)
- We were wrong in estimating MySQL would do poorly with JOIN’s
    - It kept up with Postgres quite well.
    - Even beat Postgres on the aggregate test.
        - This is due to MySQL having a “caching layer” called the join_buffer, which stashes away seen rows as the system works through a join, so there is no need to make direct calls to the buffer pool.

Conclusion

- From a raw performance standpoint, MySQL keeps up with Postgres
- For smaller sets of data, Postgres is relatively slow compared to MySQL
- MySQL does have some performance caveautes
    - i.e. performs slower if large amounts of data aren’t in the join_buffer
    - For large data sets, the first run of a query can be magnitudes slower than the rest

Lessons Learned

- At the end of day, it is about analyzing if a particular DBMS (along with some of the tradeoffs it has) is a good fit for your project.
- Both systems have trade-offs
  - MySQL = query speed inconsistency w/ most being highly performant
  - Postgres = fairly consistent query speeds, but overall less performant for smaller sets of data
- Other systems do as well (e.g. Mongo having no formal JOIN mechanism, yet very easy to get started with; etc....)
- Do the due diligence of making sure chosen DBMS is right tool for the job.

Thank you!