Project Part 2
CS487/587
Ian Percy + Sean Mitchell

What systems we will be working with and why we chose them?

(Option 1) We have chosen to use PosgreSQL and MySQL, both installed on the server dbclass.pdx.edu on CAT hardware. The choice here was to compare two different DBMS systems operating on the same hardware. Having two systems on the same server is critical for an accurate comparison when attempting to judge overall performance of benchmarks. These systems were also chosen in part because they are open-source and provide ample community support for any questions and content online.

System Research

PostgreSQL and MySQL are two popular open-source relational database management systems (RDBMS). PostgreSQL has been around since 1996 and is heavily focused on both ACID transactional compliance and extensibility. MySQL was released slightly before PostgreSQL in 1995. MySQL is another open-source RDBMS which is easy to setup and offers a large-scale database server. There are few areas important to comparing both PostgreSQL and MySQL.

SQL compliance is important when comparing any RDBMS. It isn't always possible for RDBMS to offer the full SQL capabilities, but the goal is to offer as many as possible. PostgreSQL offers 160 out of 179 of the core SQL standard features thus being "most" SQL compliant. MySQL is partially SQL compliant and also offers some no-SQL features as well.

Important to comparing these two systems are the types of indices, the types of joins, buffer pool size/structure and also mechanisms for measuring query execution time. PostgreSQL offers a several different types of indexes for tables. The main an default index in PostgreSQL is a B-tree, which is inherently ordered. Most important part of the index discussion for PostgreSQL is that for the traditional idea of a "cluster", there is physical ordering of rows in a table. Postgresql does not support this except using the CLUSTER command to physically reorder the table based on the index chosen. Subsequent insertions into the table are not guaranteed to maintain the physical ordering of the table. MySQL offers B+ tree and hash indexes. The B+ tree index remains the default for MySQL as well. However, MySQL offers actual clustering based on the primary key/index.

These two databases differ in the types of joins available to them. PostgreSQL offers three different join strategies: nested loop join, merge join, and hash join. Nested loop join offers the ability to scan with an index from one of the relations. The types of joins available in MySQL are the nested-loop join and the block nested-loop join. This is different than PostgreSQL because MySQL does not offer hash joins. The lack of the hash join could impact performance for certain analytic type queries.

The next mode of comparison between these two databases are the buffer pool sizes and structures. For PostgreSQL, the buffer manager contains the buffer table, buffle descriptors,

and the buffer pool. The buffer pool is structured as an array. Slot sizes for the pool are 8KB and the size of the pool can be configured. PostgreSQL has chosen to use a clock algorithm for the replacement policy of buffer pages. The MySQL InnoDB manages the buffer pool as a list. The replacement policy chosen for buffer pool pages is a least recently used (LRU). The InnoDB buffer pool has a default size of 8MB for MySQL.

Gathering execution plans and time is fairly straightforward for both databases. PostgreSQL offers a command “EXECUTE ANALYZE” to prepend on any query that will display both the query plan chosen by the optimizer as well as the execution time for each node. MySQL is a bit different for determining execution time. The option needed for MySQL is to use the command “set profiling =1;” and then after any query (or set of queries), running the command “show profiles;” will display execution duration for queries that have been run so far.

Overall, there are fundamental differences between the two systems. The biggest ones that stand out are the lack of an actual “clustered” index that updates for PostgreSQL and the lack of hash joins for MySQL. These two different databases, while both RDBMS, serve different purposes. From the research seen it appears that PostgreSQL will perform better on analytic and querying, but MySQL performs stronger real-time with multiple updates due to its InnoDB storage engine.

(1) https://hackr.io/blog/postgresql-vs-mysql
(2) https://pgdash.io/blog/postgres-btree-index.html
(3) https://blog.panoply.io/postgresql-vs.-mysql

Performance Experiment Design

1) Test bulk updates
    i) This test includes observing the impact of performing updates on over 50% of the tuples in a table
    ii) Use the 100,000 tuple relation
    iii) No wisconsin benchmark queries from Wisconsin benchmark exist:
    Perform a update to a table to update the at 50%,75%,100% selectivity:

50% selectivity:

```
UPDATE onehundredktup
SET two = 1
WHERE four = 0 or four = 1
```

75% selectivity:

```
UPDATE onehundredktup
SET two=1
WHERE four <= 2
```

100% selectivity:

```
UPDATE onehundredktup
SET two=1
WHERE four >= 0
```

Perform bulk update after a join (two tables of same size):

```
UPDATE onehundredktup1
```

SET a.two = 1
FROM onehundredktup1 a
JOIN onehundredktup2 b
ON a.four = b.four
WHERE b.four eq 0
Join should match on 25% of the rows from each relation

Perform bulk update on an index:
UPDATE onehundredktup1
Set unique2 = unique1

Report the plans used the by the optimizer and also the execution time

iv) n/a

v) These tests will test how the two databases handle updates to certain portions of each table. The attribute “four” was chosen to give distinct difference between the selectivity of choosing one or more value. There is an expected difference between the two systems since PosgreSQL will store in a heap structure and MySQL uses a B+ tree. These tests allow us to determine the performance of updates on tables in different manners:
Involving different portions of the table
Updates after a join
Updates to an index

2) Join algorithm performance (borrowed from the part2 specification)
   i) Compare join algorithm performance on two different systems
   ii) Using the 100,000 tuple relation
   iii) Use the Wisconsin Benchmark queries 13 and 14. Run queries on PosgreSQL and MySQL and compare the performance. Also, report the query execution plans with the types of joins used. We have chosen to use the queries utilizing an index
   iv) n/a
   v) Interest in this test is to see how the joins perform since PosgreSQL has the option for hash joins and MySQL does not
3) Test the use of partial indices (PosgreSQL offers and MySQL doesn’t)
   i) Compare the performance impact of partial indices (PosgreSQL allows for partial indexes and MySQL does not)
   ii) Using the 100,000 tuple relation
   iii) In PosgreSQL create a partial index on “two” to select 50% of the relation:
CREATE INDEX idx_two_value_0
ON onehundredktup(two)
WHERE two = 0
Perform the following queries on both systems:
SELECT *
FROM onehundredktup

```
WHERE two = 0

SELECT *
FROM onehundredktup
WHERE two = 1
```

Report the plans used by the optimizer as well as the performance information

iv) n/a

v) These tests will show the impact that a partial index has on performance. Partial indexes are enticing because they have smaller space requirements.

4) Scaleup on a 3-way join

i) Compare the impact of scaleup on a 3-way join of the same tuple

ii) Using the 1k, 10k, 100k, and 1mil tuple sizes

iii) Test the following query for each of the systems and tuple sizes:

```
SELECT
    A.unique1, b.unique1, c.unique1
FROM onektup a
    JOIN onektup b
    ON a.unique2 = b.unique2
    JOIN onektup c
    ON a.unique2 = c.unique2
```

This test will be performed on the tables without any indices. Report the plans chosen by the optimizer as well as the joins and the execution times

iv) n/a

v) Analyze the impact that scaleup in table size has on the join performance for both PostgreSQL and MySQL

Lessons Learned

This part of the project taught us several lessons. First, attempting to get everything to move smoothly on every environment does not always go as planned. At first we were running into permission issues which was resolved from a VPN to the PDX domain. Next, attempting to get all queries to run and load smoothly for PostgreSQL had it's challenges when we are not super users (ex. The copy command doesn't work from a local host to the PDX network). Also, connecting to the MySQL database has had some setbacks as well.

Next, finding out implementations details of a DBMS is not always straightforward. For example, when researching the MySQL implementation specifications - not all of it was listed in a clear manner. When attempting to determine the join types available to the MySQL optimizer, it was only through articles comparing PostgreSQL to MySQL implementations that the lack of a hash join was brought forth. It might be that MySQL offers a paid solution, so releasing all information might not be as profitable.

Lastly, there are a multitude of benchmark tests and examples for determining the performance of a database. One can easily find ways to test specific portions of a database or the entire system as a whole. However, without fully knowing how the database structures tables in the file system, what the optimal buffer size is for the average join or other information, determining which benchmarks to run can be confusing. These DBMS's allow for configuration information to be changed, such as working memory size and shared buffer space, so tests can be tweaked to any level of detail.