Benchmarking Common SQL Implementations

PostgreSQL, MySQL, and SQLite

https://github.com/apaulled/sql-benchmarking

Introduction

I conducted a series of time analyses on 3 different SQL implementations, across a variety of datatypes. The analysis program was written in Python.

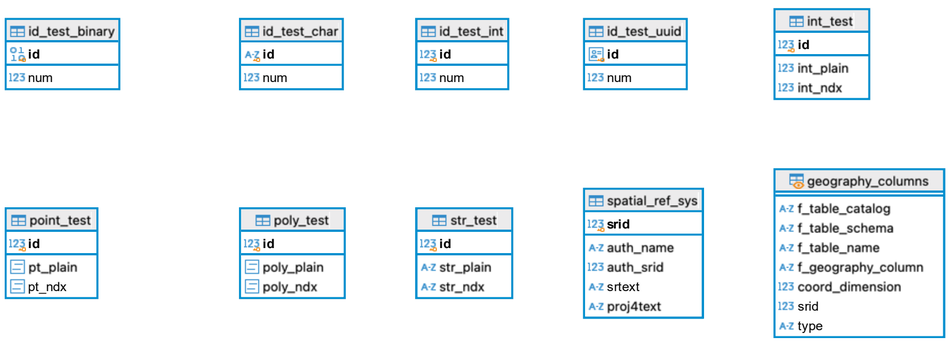
Datatypes:
- Integer
- String (text/char)
- Point (geometric)
- Polygon (geometric)

Each query was run on an indexed and non-indexed version of the column.

I varied the iterations of each query and the number of entries in the tables for the sake of running the analysis in a reasonable time, but these numbers were kept consistent across implementations so that the run times could be fairly compared.

Schema

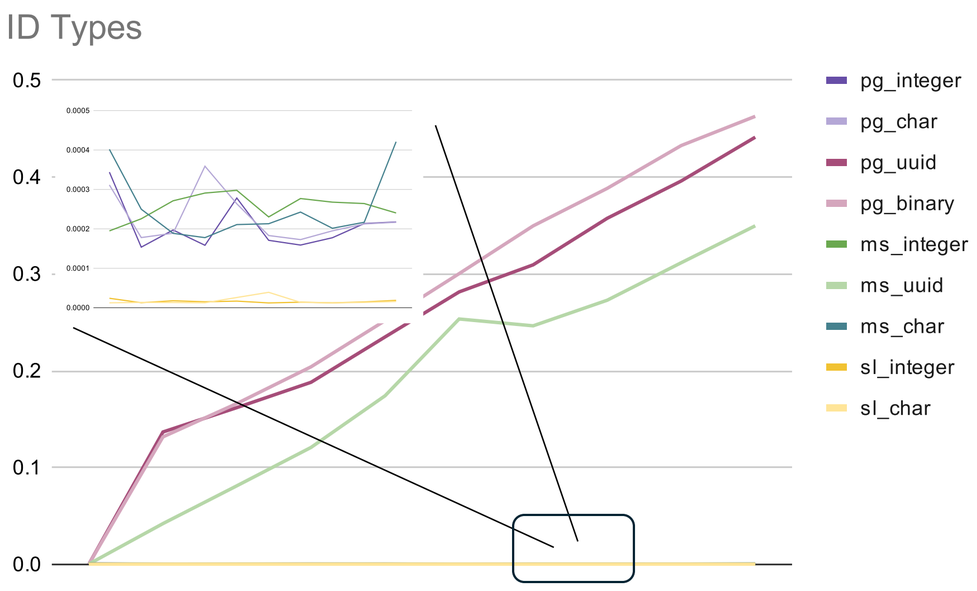
The schema for this experiment is relatively simple, to keep the time analyses mostly free of outside influence.



ID/Primary Key Analysis

I also ran an analysis of query times over various common ID datatypes in the three implementations:
- Integer
- Character
- UUID
- Binary

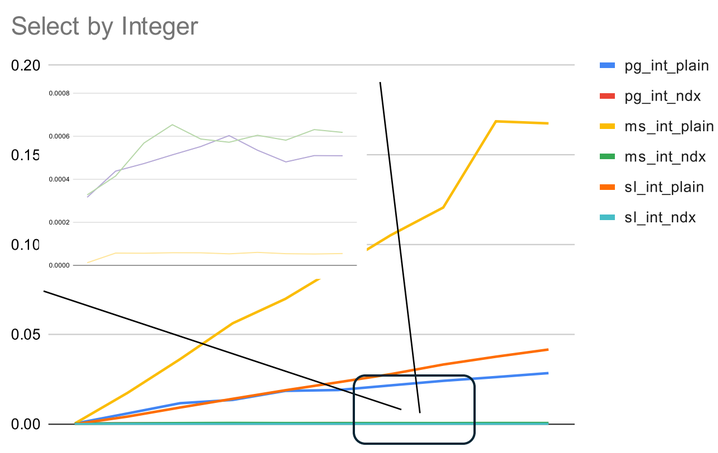
Query times in this section were averaged across 100 iterations for all data types.

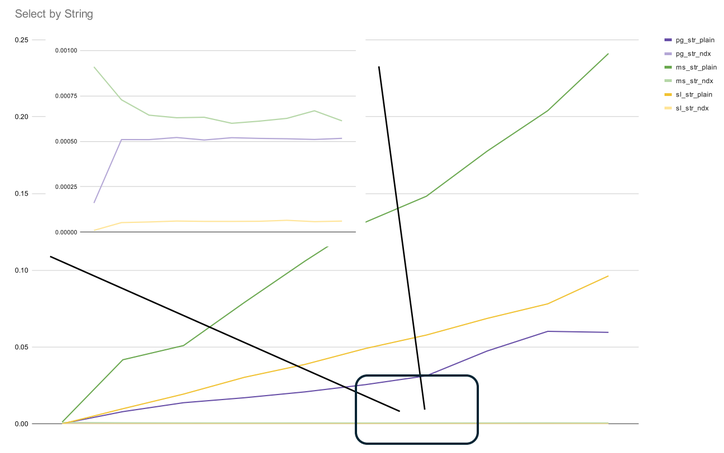


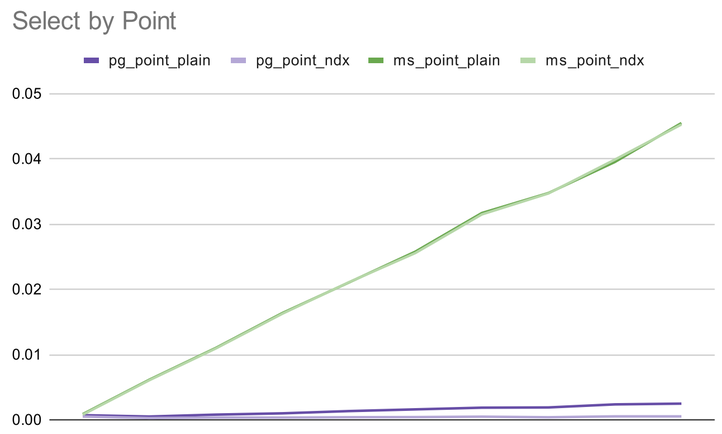
It appears that the most expensive ID types, according to this analysis, are PostgreSQL UUID/Binary types, and the MySQL UUID type. The thing that all of these types have in common is that they are stored as binary data. PostgreSQL stores UUIDs as 16-bit binary data, and I implemented the MySQL UUID type as a 16-bit binary column because MySQL does not have a native UUID type. This suggests that read-only queries are more expensive on binary data than integer or character data.

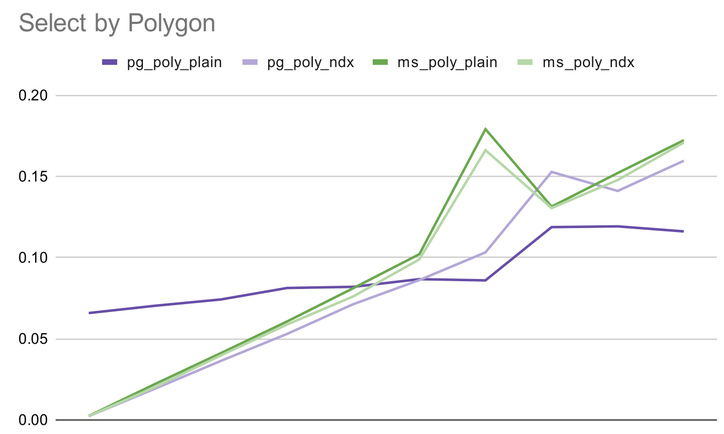
Simple Query Analysis

Query times in this section were averaged across 100 iterations for non-geometric datatypes and 10 iterations for geometric datatypes.



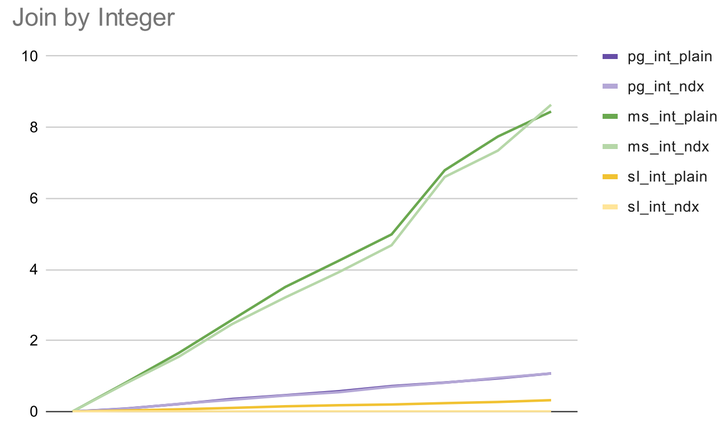


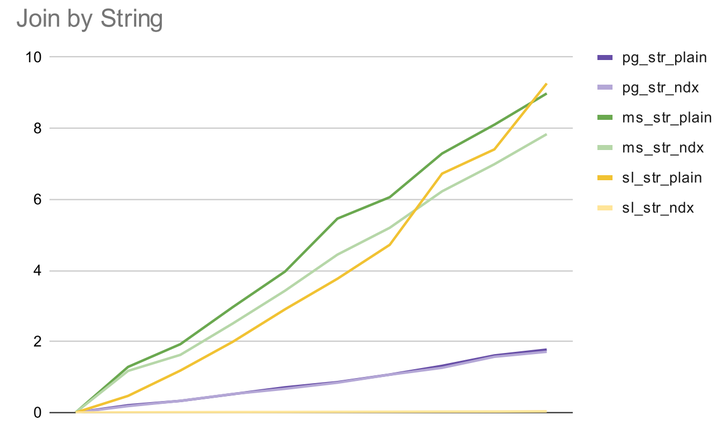


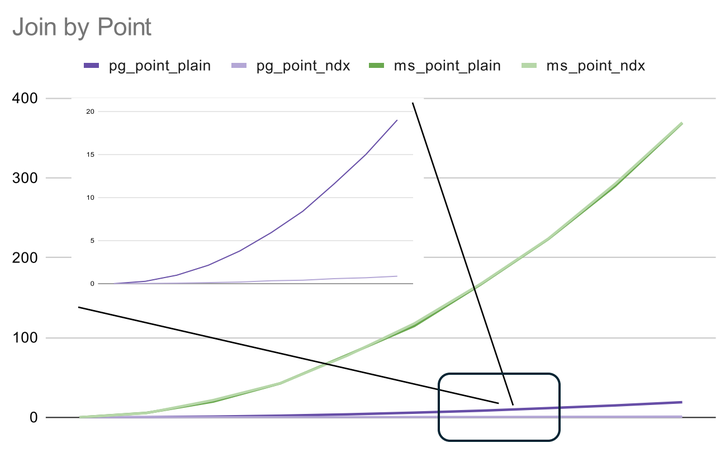


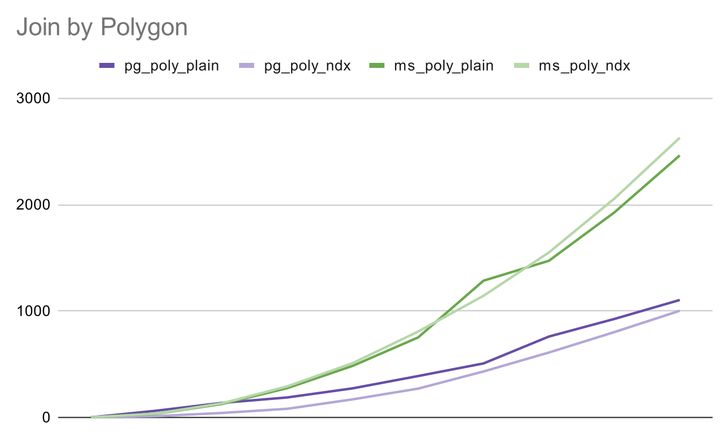
Join Query Analysis

Query times in this section were averaged across 10 iterations for non-geometric datatypes and 2 iterations for geometric datatypes.









Conclusions

In the simple query analysis, the results were largely as-expected. Non-indexed columns appeared to behave with an O(n) time complexity, and indexed columns appeared to behave with an O(1) time complexity. The only major exception appeared in the Polygon datatype, where the index does not appear to have helped. When the query is run with explain, the index is shown as being used, but it does not appear to help much in context.

In the join query analysis, the non-geometric datatypes appeared to behave with O(n) complexity when not indexed. This is unexpected, but it could be the case that there was not enough data to reveal the $O(n^2)$ nature. It is, however, clear that the indexes are helpful in reducing query time. Join queries on the point datatype appear to be $O(n^2)$ when not indexed and O(nlogn) when indexed, as expected. Once again, the Polygon datatype appears not to be helped very much by the index.

Overall Conclusions:
- Indexes help greatly in improving efficiency, especially on queries involving a join
- PostgreSQL with PostGIS is clearly superior to MySQL when handling geometric types
- Integer and character types are the most efficient to use as IDs/primary keys
- Querying geometric types is much more expensive than querying non-geometric types
- SQLite is the quickest of these 3 implementations for read-only data

by: Paul Gagliano