A COMPARISON OF COMPUTATIONAL PERFORMANCE BETWEEN SQLITE AND POSTGRESQL DATABASES

Steve Desilets

MSDS 420: Database Systems and Data Preparation

June 1, 2023

1. Abstract

This study aims to examine differences in the computational performance of PostgreSQL and SQLite databases as measured via run times in seconds for SQL queries. The data collected by these SQL queries are representative of data collected by digital measurement companies about panelists and their online activity. SQL query run times were collected from each of the two databases (PostgreSQL and SQLite) for three varying database sizes (small, medium, and large) using SQL queries of three complexity levels (simple, moderate, and complex). The data from these nine experiments were then analyzed via visualizations and Bonferroni-adjusted, two-tailed t-tests for differences in mean run times between the two database types. The visualizations and t-tests display that PostgreSQL outperforms SQLite in returning SQL results more quickly across database sizes and query complexities and that these differences in query run times appear to grow as the database size increases.

2. Introduction

Data-centric organizations aiming to optimize the efficiency of their operating models must carefully assess which databases will maximize organizational performance by executing database queries as quickly as possible. Such organizations may elect to conduct a benchmark study to statistically evaluate the magnitude of differences in the speed with which various database systems complete queries. For this study, we put ourselves in the position of a digital measurement company evaluating whether it should establish a SQLite or PostgreSQL database to store its data regarding panelists' characteristics and online activities. Given that effective data management sits near the core of this company's business model, this company must identify which database will operate most efficiently so that the firm may minimize labor costs associated with employees awaiting query results.

3. Literature Review

Since selection of the optimal database is such an important area of focus for individuals designing a database system, previous research has focused on comparing the computational performance of various databases.

Grzesik and Mrozek conducted experiments to compare the computational speeds of three time-series databases (TimescaleDB, InfluxDB, and Riak TS) and two relational databases (PostgreSQL and

SQLite) within the context of inserting and querying data for edge analytics applications (Krzhizhanovskaya et al. 2020). These researchers found that PostgreSQL outperformed the other four databases in three out of the four conducted experiments, so the researchers concluded that PostgreSQL would most likely be the best database choice (among those tested) for data scientists interested in edge computing.

4. Methods

Given that the goal of this study was to identify whether operationally important differences in computational speeds existed between SQLite and PostgreSQL databases, this study aimed to conduct two-tailed t-tests for differences in mean query run times between the two databases. The first step for executing this experiment was to generate the data that our SQL databases would store. In Jupyter Notebooks, we generated two datasets that would hold data representative of the data needed by this digital measurement company: one table to hold panelist information and one table to hold the panelist activity data. The panelist information dataset contained five columns of data: 1) panelist ID, 2) name, 3) socioeconomic status, 4) age, and 5) sex. The panelist activity dataset contained four columns of data: 1) panelist ID, 2) webpage, 3) number of pages visited, and 4) minutes on webpage. Given that we are curious about the impact of database size on performance, we generated three versions of each of these two datasets that each contained 1,000, 10,000, and 100,000 rows. After conducting exploratory data analysis on these datasets to confirm they were constructed properly, we loaded these 6 tables into our PostgreSQL and SQLite databases.

Next, we calculated the sample size of query run times that we would need to collect from each database by applying the following formula that researchers use to determine sample sizes for experiments that aim to identify statistically significant differences in population means between two groups (Sullivan).

$$n_i = 2 * \left(\frac{t_{1-\frac{\alpha}{2}} + t_{1-\beta}}{\frac{E}{\sigma}}\right)^2$$

where

n_i = the number of query run times that need to be collected from each database for each experiment

σ = the standard deviation of the difference in query run times (between PostgreSQL and SQLite) in a previous study

E = the operationally significant margin of error of interest

$t_{\alpha/2}$ = the appropriate t-test statistic given the desired confidence level for Type I errors

$t_{1-\beta}$ = the appropriate t-test statistic given the desired power for Type II errors

We calculate the values of the $t_{\alpha/2}$ and $t_{1-\beta}$ for the above equation as 12.7 and 1.4, respectively, by assuming that the digital media measurement company wants to detect mean differences in SQL query run times of 0.5 seconds (E) while achieving a Type 1 error rate of 5% and a power of 80%. By extracting data from visualizations included in a prior study of the differences in run times between SQLite and PostgreSQL databases, we can estimate the value for σ as 0.328 seconds (Wodyk and Skublewska-Paszkowska 2020). By plugging these values into the above equation, we learn that we will need to acquire 171 samples from each database for each of the nine experiments.

After generating our panelist data, loading the panelist data into our databases, and calculating the necessary sample size, we then needed to collect, compile, and analyze the run time data. We executed three SQL queries of varying complexity (simple, moderate, and complex) on our database tables of three varying sizes (small, medium, and large) for each of our two databases (PostgreSQL and SQLite). (The full text of the nine SQL queries are provided in Appendix B). Accordingly, this experiment design meant that we conducted 18 SQL queries, which were each repeated 171 times per our sample size calculations, so we collected 3,078 run time observations in total.

We then explored the distributions of this run time data via visualizations and conducted two-tailed, Bonferroni-adjusted t-tests for each of our nine experiments to better understand the differences in performance between SQLite and PostgreSQL databases.

5. Results

Analysis of run time distributions disaggregated by our three single categories (database type, database size, and query complexity) yields some interesting findings (as displayed in Figures 1, 2, and 3 in Appendix A). The most striking finding is that database size clearly plays a strong role in query run times with increasing database sizes leading to visibly greater run time distributions. Though the distribution of SQLite and PostgreSQL run times largely overlapped and exhibited similar right skew, the SQLite queries still notably took nearly 1.6 times longer than the PostgreSQL queries on average. The run time distributions for the simple, moderate, and complex queries exhibited a fair amount of overlap and similar right skew. Interestingly, though, the simplest query (a simple select statement from one table) created the distribution with the highest average run times even though the moderate and complex queries contained "join" (and "group by" and "order by") clauses. This finding suggests that, in some cases, queries that appear to be more complex may actually have shorter run times than queries that appear to be more simple.

The nine side-by-side boxplots displayed in Figure 4 in Appendix B allow us to directly compare the performance of SQLite and PostgreSQL for each of our nine experiments. The boxplots clearly display PostgreSQL outperforming SQLite in each of the nine experiments. In fact, in three of the nine experiments, the longest PostgreSQL run time is shorter than the shortest SQLite run time, so the run time distributions don't even overlap. Visually, as we scan from left to right across the rows in this figure, we observe that as the database sizes increase, the differences in the median run times between SQLite and PostgreSQL increases. However, as we scan from top to bottom in the columns of this figure, we do not observe a similar trend of increasing differences in median run times between SQLite and PostgreSQL. Still, these run time differences appear to have potential operational importance, given that one of the experiments (the query of moderate complexity on the large databases) yields a difference in median run times of approximately 0.25 seconds.

The t-tests confirmed the results that were visually apparent in the side-by-side boxplots that had visualized the SQLite and PostgreSQL performance in each of our nine experiments. Eight of the nine two-tailed, Bonferroni-adjusted t-tests yielded statistically significant differences in the means between the

PostgreSQL and SQLite database run times (as measured in seconds). These findings clearly confirm that for a broad array of database sizes and query complexities, PostgreSQL outperforms SQLite in returning SQL query results promptly.

6. Conclusions

This study achieved its goal of identifying differences between the computational performance of SQLite and PostgreSQL databases (as measured via query run times) for queries of varying complexity and databases of varying sizes. This study concludes that PostgreSQL databases tend to outperform SQLite databases across database sizes and query complexities and that as the database size grows, these performance differences across database types tend to grow as well. The findings of this study indicate that digital measurement companies interested in optimizing the query runtime efficiency of their databases (in order to minimize the costs associated with longer query run times) should consider constructing their relational databases using PostgreSQL rather than SQLite. While the findings of this study focused on data collected by digital measurement companies, these findings could be highly beneficial to other types of organizations that leverage relational databases as well. Further research could confirm these findings hold true for other types of data or could compare the performance of SQLite and PostgreSQL databases to other types of databases.

References

Krzhizhanovskaya, Valeria V, Gábor Závodszky, Michael H Lees, Jack J Dongarra, Peter M. A Sloot, Sérgio Brissos, and João Teixeira. 2020. "Comparative Analysis of Time Series Databases in the Context of Edge Computing for Low Power Sensor Networks." In *Computational Science - ICCS 2020*, 12141:371–83. Switzerland: Springer International Publishing AG. https://doi.org/10.1007/978-3-030-50426-7_28.

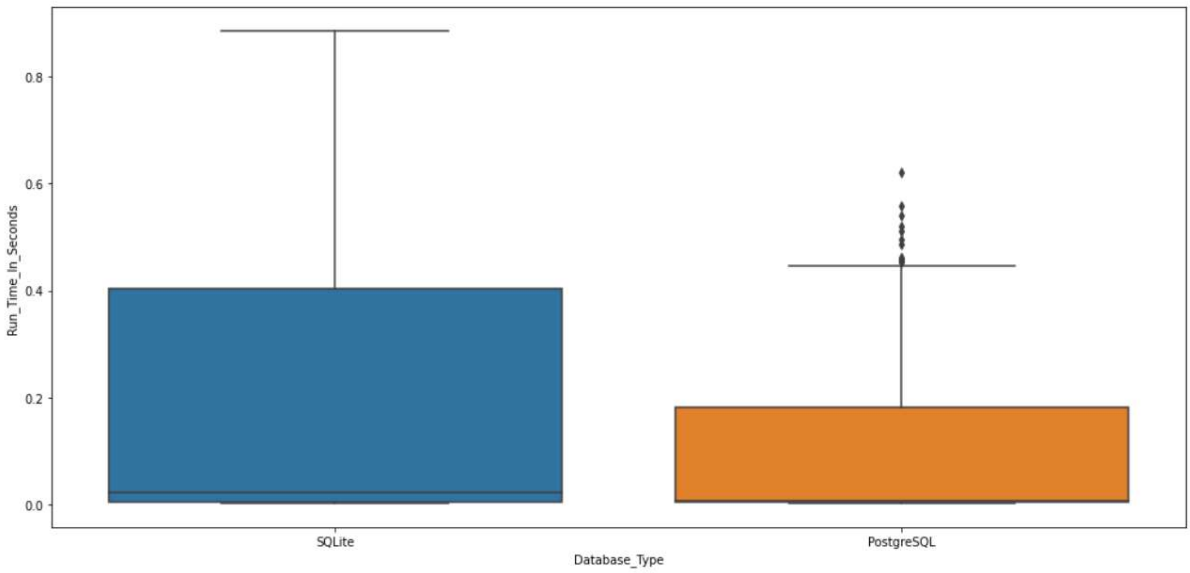
Sullivan, Lisa. "Power and Sample Size Determination." *Boston University School of Public Health.* https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_power/bs704_power_print.html#:~:text=The%20formula%20for%20determining%20sample%20size%20to%20ensure%20that%20the,%3D%200.975%20and%20Z%3D1.960.

Wodyk, Rafał, and Maria Skublewska-Paszkowska. 2020. "Performance Comparison of Relational Databases SQL Server, MySQL and PostgreSQL Using a Web Application and the Laravel Framework." *Journal of Computer Sciences Institute* 17: 358–64. https://doi.org/10.35784/jcsi.2279.

Appendix A – Visualizations

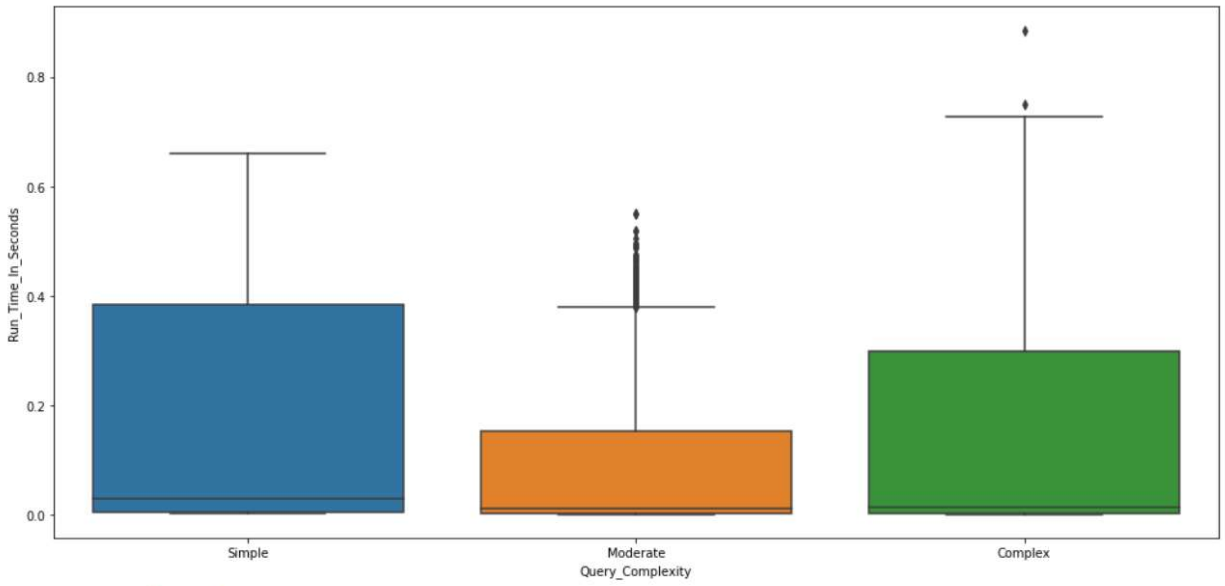
Figures 1, 2, 3, and 4 display the distribution of SQL query run times (in seconds) when disaggregated by database type, query complexity, database size, or all three categories, respectively. Figure 5 displays the results of the Bonferroni-adjusted t-tests that compare the mean run time performance of the PostgreSQL and SQLite databases for each of our nine experiments focused on three database sizes (small, medium, and large) and three levels of query complexity (simple, moderate, and complex).

Figure 1. Distribution of Query Run Times Disaggregated by Database Type



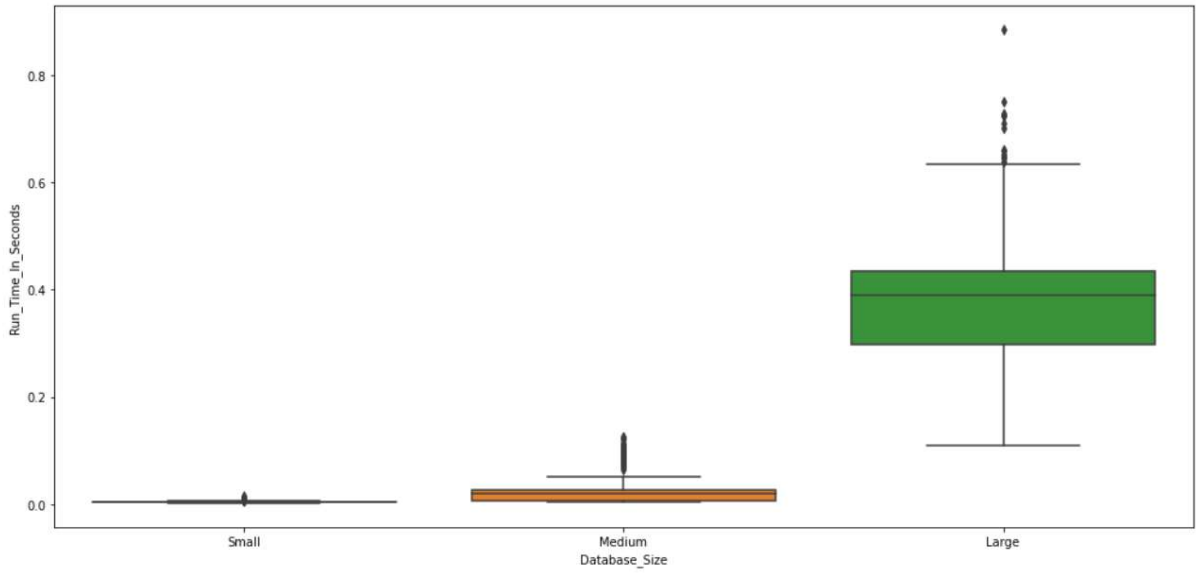
	Database_Type	count	mean	std	min	25%	50%	75%	max
0	PostgreSQL	1539.0	0.100824	0.142583	0.001493	0.004046	0.00717	0.181677	0.622197
1	SQLite	1539.0	0.158834	0.205716	0.002513	0.005163	0.02182	0.404369	0.885628

Figure 2. Distribution of Query Run Times Disaggregated by Query Complexity



	Query_Complexity	count	mean	std	min	25%	50%	75%	max
0	Complex	1026.0	0.134285	0.187367	0.001983	0.004102	0.014614	0.298905	0.885628
1	Moderate	1026.0	0.099694	0.150173	0.001493	0.004051	0.013270	0.154563	0.550036
2	Simple	1026.0	0.155508	0.193088	0.002516	0.005524	0.029966	0.384179	0.661045

Figure 3. Distribution of Query Run Times Disaggregated by Database Size



	Database_Size	count	mean	std	min	25%	50%	75%	max
0	Large	1026.0	0.363169	0.119406	0.110172	0.298716	0.389302	0.434585	0.885628
1	Medium	1026.0	0.022216	0.018851	0.004049	0.007139	0.020118	0.026760	0.125762
2	Small	1026.0	0.004102	0.001340	0.001493	0.003105	0.004001	0.004725	0.014552

Figure 4. Distribution of Query Run Times Disaggregated by Database Type, Query Complexity, and Database Size

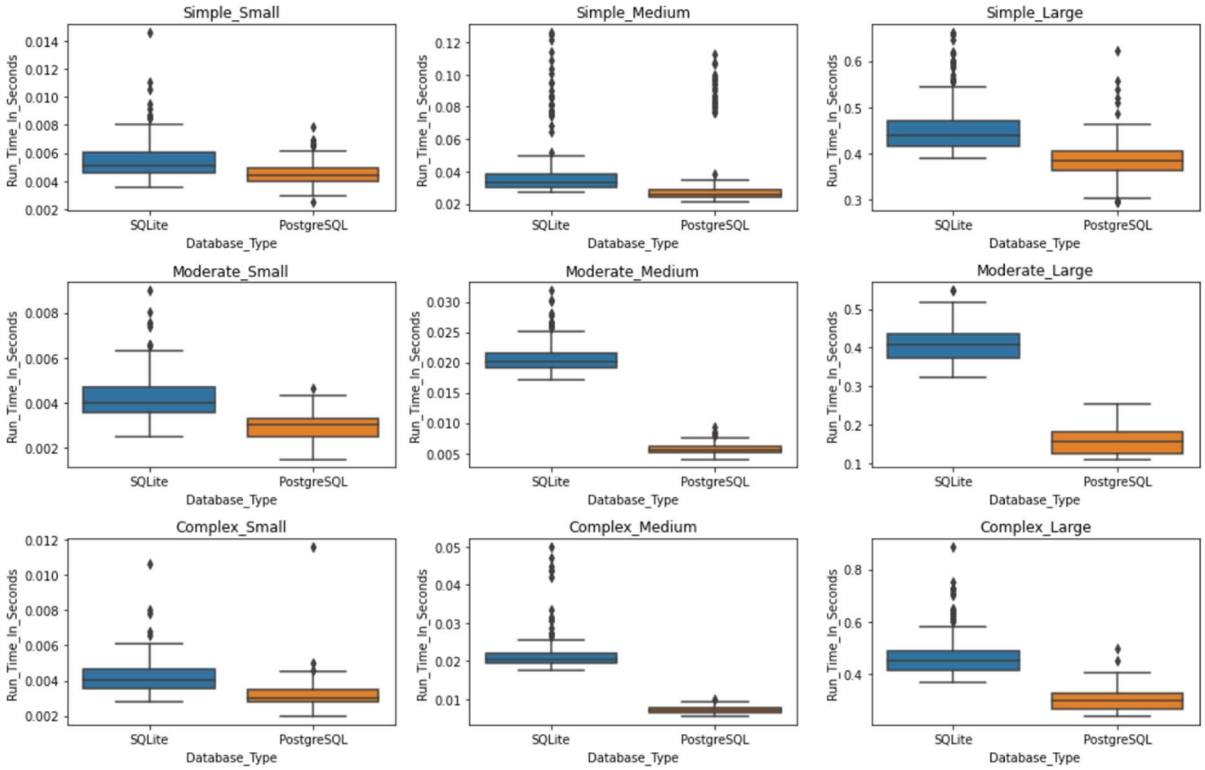


Figure 5. T-Test Results

	Test_Type	T_Test_Statistics	Unadjusted_P_Values	Statistically_Significant_Difference
0	Simple_Small	8.605826	5.272750e-16	True
1	Simple_Medium	2.685789	7.592814e-03	False
2	Simple_Large	12.043038	1.178293e-27	True
3	Moderate_Small	15.111171	4.097822e-38	True
4	Moderate_Medium	72.070307	1.421596e-148	True
5	Moderate_Large	57.380393	1.728995e-166	True
6	Complex_Small	10.297455	9.059194e-22	True
7	Complex_Medium	36.541488	5.118693e-85	True
8	Complex_Large	23.205105	5.678418e-65	True

Appendix B – SQL Queries

The SQL code for each of the nine experiments is provided below. The first three queries execute a simple select statement on one table (of three varying sizes). The second three queries execute a moderately complex SQL statement that includes a join between two tables (of three varying sizes). The last three queries execute a complex SQL statement that includes a join between two tables (of three varying sizes) as well as "group by" and "order by" clauses.

Experiment 1: Small Database and Simple Query

```
SELECT *
FROM small_panelist_df
```

Experiment 2: Medium Database and Simple Query

```
SELECT *
FROM medium_panelist_df
```

Experiment 3: Large Database and Simple Query

```
SELECT *
FROM large_panelist_df
```

Experiment 4: Small Database and Moderate Complexity Query

```
SELECT p.*,
        a.website,
        a.number_of_pages_visited,
        a.minutes_on_webpage
FROM small_panelist_df p
INNER JOIN small_activity_df a
ON p.panelist_id = a.panelist_id
```

Experiment 5: Medium Database and Moderate Complexity Query

```
SELECT p.*,
        a.website,
        a.number_of_pages_visited,
        a.minutes_on_webpage
FROM medium_panelist_df p
INNER JOIN medium_activity_df a
ON p.panelist_id = a.panelist_id
```

Experiment 6: Large Database and Moderate Complexity Query

```
SELECT p.*,
        a.website,
        a.number_of_pages_visited,
        a.minutes_on_webpage
FROM large_panelist_df p
INNER JOIN large_activity_df a
ON p.panelist_id = a.panelist_id
```

Experiment 7: Small Database and Complex Query

```
SELECT p.panelist_id,
        p.name,
        p.socioeconomic_status,
        p.age,
        p.primary_user_sex,
        a.website,
        SUM(a.number_of_pages_visited) AS total_pages_visited,
        SUM(a.minutes_on_webpage) AS total_minutes_on_webpage
FROM small_panelist_df p
INNER JOIN small_activity_df a
        ON p.panelist_id = a.panelist_id
GROUP BY p.panelist_id,
         p.name,
         p.socioeconomic_status,
         p.age,
         p.primary_user_sex,
         a.website
ORDER BY p.name
```

Experiment 8: Medium Database and Complex Query

```
SELECT p.panelist_id,
        p.name,
        p.socioeconomic_status,
        p.age,
        p.primary_user_sex,
        a.website,
        SUM(a.number_of_pages_visited) AS total_pages_visited,
        SUM(a.minutes_on_webpage) AS total_minutes_on_webpage
FROM medium_panelist_df p
INNER JOIN medium_activity_df a
        ON p.panelist_id = a.panelist_id
GROUP BY p.panelist_id,
        p.name,
        p.socioeconomic_status,
        p.age,
        p.primary_user_sex,
        a.website
ORDER BY p.name
```

Experiment 9: Large Database and Complex Query

```
SELECT p.panelist_id,
        p.name,
        p.socioeconomic_status,
        p.age,
        p.primary_user_sex,
        a.website,
        SUM(a.number_of_pages_visited) AS total_pages_visited,
        SUM(a.minutes_on_webpage) AS total_minutes_on_webpage
FROM large_panelist_df p
INNER JOIN large_activity_df a
        ON p.panelist_id = a.panelist_id
GROUP BY p.panelist_id,
         p.name,
         p.socioeconomic_status,
         p.age,
         p.primary_user_sex,
         a.website
ORDER BY p.name
```