Performance of SQL PreparedStatement on large dataset

Qingwen Zeng

December 15, 2022

1 Background

1.1 Parameterized statement

A parameterized query is a query in which placeholders are used for parameters and the parameter values are provided as arguments at execution time. Parameters are defined before a query using a declare statement. Declare statements start with the keyword DECLARE , followed by the name of the parameter (starting with a question mark) followed by the type of the parameter and an optional default value. The default value must be a literal value, either string , numeric , boolean , date , or time[2].

1.2 Query plan

A query execution plan is a sequence of steps used to access data in a SQL relational database management system. Since SQL is declarative, there are multiple ways to execute a given query, with widely varying performance. When a query is submitted to the database, the query optimizer evaluates some of the different, correct possible plans for executing the query and returns what it considers the best option[11]. The query optimizer evaluates multiple query plans for a single query and determines the most efficient plan to run, so the output must be the best query plan.

1.3 Query Process Workflow

Before we look into the PreparedStatement, we need to know parsing and the steps of processing a query. The query process workflow is in the following diagram Figure2. When there is reusable cursor, DBMS will directly execute it, otherwise it will parse the query and find the optimal execution plan.

1.4 Parsing

A SQL statement works with many tables and variables. Thus there might be multiple ways to execute one given query. The Figure1. shows that DBMS must

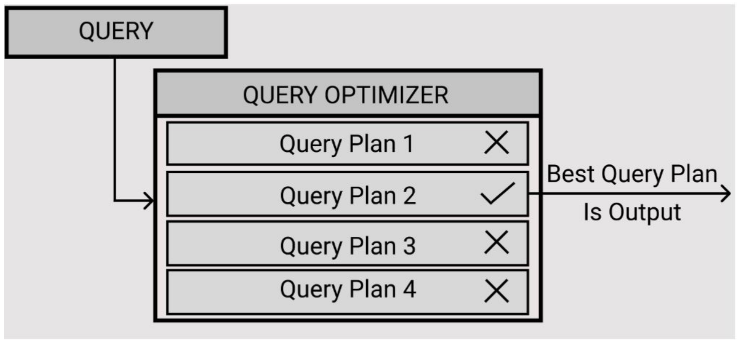


Figure 1: Query Optimizer[4]

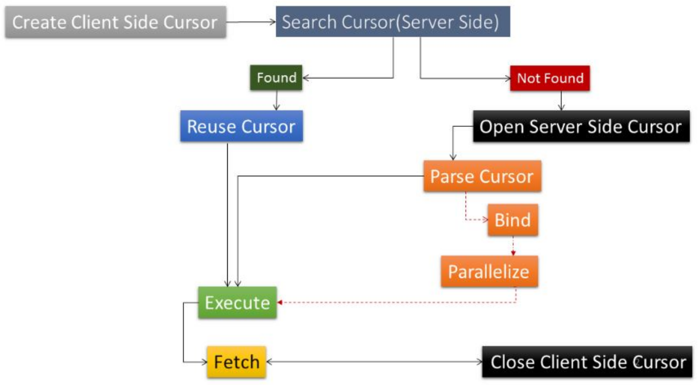


Figure 2: Query Processing Workflow[8]

execute the most optimal query plan in order to execute in the shortest possible time. And the optimal query plan desicion is made by parsing a query[8]. Parsing a query is performed within the DBMS using the optimizer component. The optimizer evaluates the condition and makes the desicion in the following order.

1. Check whether the SQL statement exists in the Postgres Server(Shared Buffer) by performing syntax and semantic checks but avoiding query optimization[3].
2. If the SQL statement exists, reuse the existing PostgresSQL area which already has the execution plan. This is soft parsing.
3. If not, such as the first time loading this query or a statement is aged out of the Postgres Server(because the Postgres Server is limited in size), it results in all the processes in Figure 2, including evaluating many different

attributes of the given query, such as number of tables, having indexes or not, etc. And then find the most optimal query plan. This is hard parsing.

1.5 Prepared Statement Technique

1.5.1 What is PreparedStatement?

Prepared statements are a feature of the programming language used to communicate with the database, and it creates a template for a query that establishes an immutable grammar[9]. It's a performance optimization and a security measure; it protects against SQL injections, where an attacker hijacks unguarded string concatenation to produce malicious queries[10].

1.5.2 Why is PreparedStatement popular?

A PreparedStatement is a pre-compiled SQL statement. Instead of hard coding queries, PreparedStatement object provides a feature to execute a parameterized query; So DBMS can just run the query instead of first compiling it. In a simple understanding, PreparedStatement is based on parameterized query and a forced soft parsing. At the time calling the PreparedStatement, it will automatically form the previous parse tree instead of compiling again, and then DBMS will execute the previous steps in Postgres server. In this way, PreparedStatement is faster than standard SQL.

1.5.3 Applicability of PreparedStatement

Some major DBMSs and popular programming laguages provide the PreparedStatement, which proves that PreparedStatement is widely used, so the performance of it is important.

1. Major DBMSs, such as MySQL(after version 4.0)[12], Oracle, and PostgreSQL support PreparedStatement.

2. A number of programming languages, such as Java's JDBC, and Python's DB-API[5] support PreparedStatement in their standard libraries, even if the underlying DBMS does not support it.

1.5.4 Steps of using PreparedStatement[6]

1. Create Connection to Database.

2. Create Prepared Statement object.

3. Set parameter values for type and position.

4. Execute the Query.

2 Motivation

As a method passing queries to DBMS, PreparedStatement can effectively prevent SQL Injection, and save time in some situation. The purpose of this paper is to find on large dataset, under which situation PreparedStatement has a good performance. By reusing the results of parsing, optimization and compilation, PreparedStatement takes less time comparing to standard SQL on executing a query. But this is not always true. When the data amounts or the table schema significantly altering, the PreparedStatement may not maintain the good performance. Even we don't have the PreparedStatement, according to the Figure 2, DBMS itself will decide to soft parse or hard parse every time, which means DBMS can find the optimal execution plan all the time[8]. Even through PreparedStatement saving the time of parsing, optimization and compilation, it may gradually deviates from optimal query plan which represents the shortest execution time, especially when the tables significantly change. Also, PreparedStatement needs two round trips when standard SQL query only have one[7]. And if we prepare, execute once, and then close the statement, it makes three network round-trips to the server, which is a performance reduction[10]. This makes PreparedStatement takes more time as well. If we want to keep the best performance of DBMS, we need to know the reason whether the performance of PreparedStatement decreases in some conditions, in this way we can choose the suitable query method case by case. There is a reference[7] from Tomislav Seser examined PreparedStatement performance on small dataset, and listed some possible reasons which affect the performance, but the experiments and conclusions didn't include the performance on large dataset. In this paper, I will dig more about the performance of PreparedStatement on large dataset.

3 Methods

In Seser's reference shows how the PreparedStatement can have decremental performance on small dataset, and when they should and should not be used if resources and performance is a concern[7]. My project focus on the PreparedStatement performance on large dataset. I will use the PreparedStatement class in JDBC, and comparing its processing time with which cost on PostgreSQL standard SQL. I will use currentTimeMillis() function to calculate the execution time of PreparedStatement queries, and using timing method in PostgreSQL to calculate the execution time of standard SQL query. Here are the steps I want to accomplish this project. All the updates are in this repository:`https://github.com/QingwenZeng123/preparedStatement`

1. Download netflix dataset from Kaggle[1].
2. Create a PostgreSQL database and load the data to it.
3. Create a QueryService and ExecMain class in Java.
4. Implement the PreparedStatement.

(a) Download the PostgreSQL JDBC driver and add to class path.
(b) Start the database by pg_ctl start -D pgnetflix.
(c) Make the connection to database.
(d) Write the query and create the PreparedStatement object.
(e) Add currentTimeMillis() function to the class methods.
(f) Write down the time of forming PreparedStatement and executing query.

5. Open the timing method in standard SQL on command line.

6. Write down the time of first time executing and repeatedly executing the query.

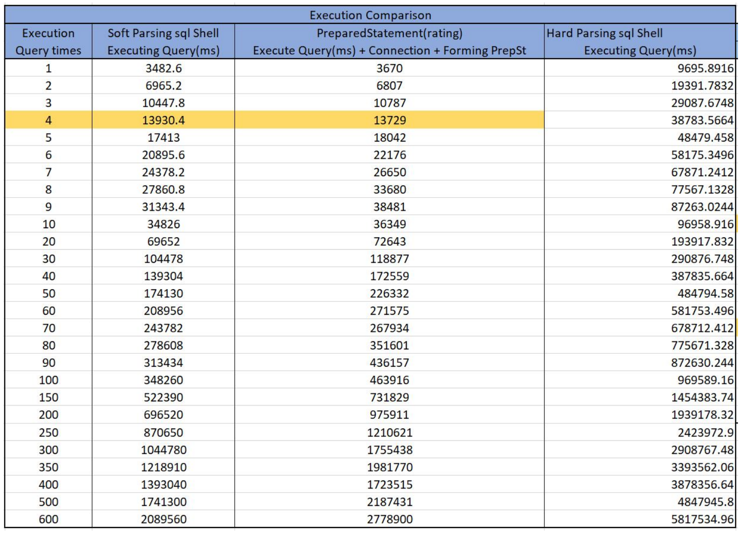
7. Drawing two scatterplots. X-axis is the times of execution, Y-axis is the execution time consuming.

8. Make the conclusions from references, tables and scatterplots.

4 Results Analysis

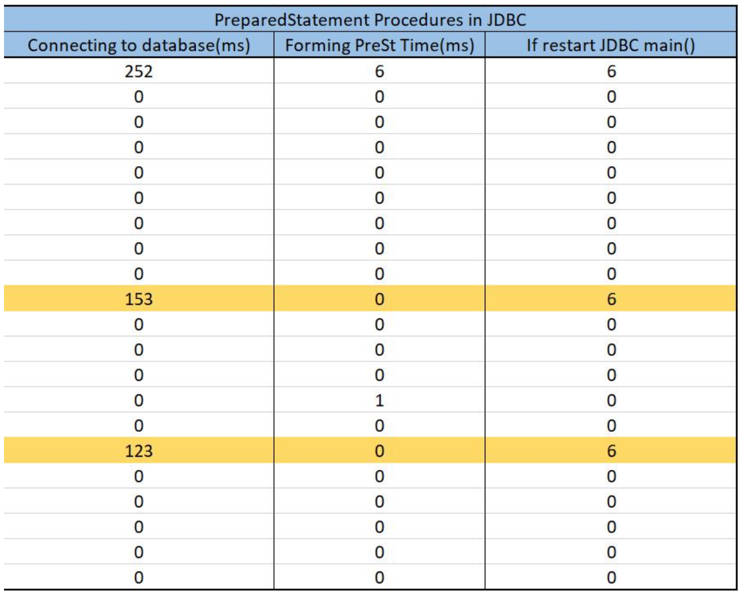
I will analyse the tables and charts in PerfCom.xlsx file in the repository.

1. From Figure 3. the PreparedStatement Procedures in JDBC table, we can find that when we create a PreparedStatement object in one program, only the first time costs 6ms, which means if we repeatedly create the same query, JDBC will recognize it. So, the PreparedStatement object is stored in the java program memory instead of the memory space in database.



Execution Comparison			
Execution Query times	Soft Parsing sql Shell Executing Query(ms)	PreparedStatement(rating) Execute Query(ms) + Connection + Forming PrepSt	Hard Parsing sql Shell Executing Query(ms)
1	3482.6	3670	9695.8916
2	6965.2	6807	19391.7832
3	10447.8	10787	29087.6748
4	13930.4	13729	38783.5664
5	17413	18042	48479.458
6	20895.6	22176	58175.3496
7	24378.2	26650	67871.2412
8	27860.8	33680	77567.1328
9	31343.4	38481	87263.0244
10	34826	36349	96958.916
20	69652	72643	193917.832
30	104478	118877	290876.748
40	139304	172559	387835.664
50	174130	226332	484794.58
60	208956	271575	581753.496
70	243782	267934	678712.412
80	278608	351601	775671.328
90	313434	436157	872630.244
100	348260	463916	969589.16
150	522390	731829	1454383.74
200	696520	975911	1939178.32
250	870650	1210621	2423972.9
300	1044780	1755438	2908767.48
350	1218910	1981770	3393562.06
400	1393040	1723515	3878356.64
500	1741300	2187431	4847945.8
600	2089560	2778900	5817534.96

Figure 4: Execution Comparison



PreparedStatement Procedures in JDBC		
Connecting to database(ms)	Forming PreSt Time(ms)	If restart JDBC main()
252	6	6
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
153	0	6
0	0	0
0	0	0
0	0	0
0	1	0
0	0	0
123	0	6
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0

Figure 3: PreparedStatement Procedures in JDBC

2. From Figure 4. the Execution Comparison table, the first time executing a standard SQL query takes more time than the second execution. This difference may be from the parsing process. PostgreSQL database hard parses the query first time, and store it in buffer, so it can soft parse the query on second one.

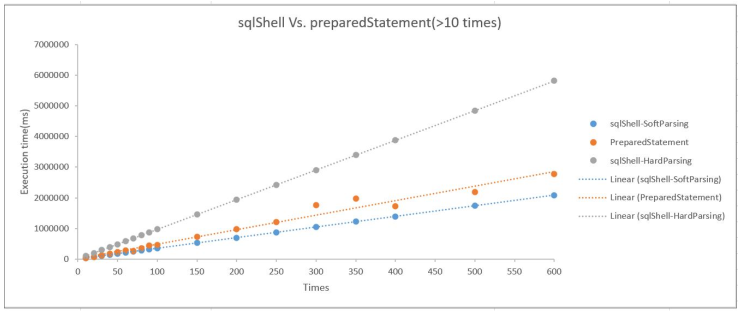


Figure 5: sqlShell Vs. preparedStatement

3. From Figure 5. the sqlShell Vs. preparedStatement chart, we can make the conclusion that the performance of PreparedStatement is better than hard parsing sqlShell, but worse than soft parsing sqlShell. This fact shows that if we want to execute same query multiple times on a busy database, it is good to use PreparedStatement. Because if the database is busy, even

if the buffer can store the previous query, the buffer size is limited[3], it will clear the out-dated memory, which might include the query we want to reuse. If we want to execute different queries or on a not busy database, we can rely on the soft parse in database itself.

5 Future work

There are three parts that I haven't done in this project.

1. From the sqlShell Vs. preparedStatement chart, we can find the both the PreparedStatement and sqlShell-SoftParsing are linear and former one are above latter all the time. Which means that there are some consistent reasons affect PreparedStatement performance. In order to get the answers, it will be a good way to check the source code of PreparedStatement class implementation in JDBC.

2. Finding the performance of PreparedStatement when the data or schema changes in database. Here is my raw plan.

   (a) Find the query which can reduce the data in rating table, such as 75%, 50% and 25%.

   (b) Add those queries to ExecMain class, and execute the same query before and after the data changes in database.

   (c) Compare the execution query time.

3. In order to understand the difference between soft parse and PreparedStatement, it is useful to find the parse tree of PreparedStatement object. This can be implemented by adding antlr4 plug-in to the QueryService class.

References

[1] Netflix prize data. `https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data`.

[2] Parameterized query. `https://docs.data.world/documentation/sql/concepts/dw_specific/parameterized_queries.html`.

[3] NIJAMUTHEEN J. PostgreSQL query flow. `https://www.tutorialdba.com/p/postgresql-query-flow.html`, 2017.

[4] Matthew Layne. What is a query plan. `https://dataschool.com/sql-optimization/what-is-a-query-plan/`, August 2021.

[5] Marc-André Lemburg. PEP 249 – Python Database API Specification v2.0. `https://peps.python.org/pep-0249/`.

[6] prateekc231. How to use PreparedStatement in java. `https://www.geeksforgeeks.org/how-to-use-preparedstatement-in-java/`, September 2022.

[7] Tomislav Seser, Vladimir Pleština, and Frane Marjanica. Performance analysis of SQL PreparedStatement in CRUD operations. In *2022 7th International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–5, 2022.

[8] Aman Sharma. Understanding SQL query parsing – part 1. `https://www.red-gate.com/simple-talk/databases/oracle-databases/understanding-sql-query-parsing-part-1/`, September 2016.

[9] Mike Shema. Chapter 4 - SQL injection & data store manipulation, 2012.

[10] SolarWinds. Analyzing PreparedStatement Performance. `https://orangematter.solarwinds.com/2014/11/19/analyzing-prepared-statement-performance/`, 2014.

[11] Wikipedia contributors. Query plan — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Queryplan&oldid=1047222641",note="[Online;accessed26-October-2022]`, 2021.

[12] Wikipedia contributors. Prepared statement — Wikipedia, the free encyclopedia, 2022. [Online; accessed 15-December-2022].