Benchmarking databases focusing on small to medium sized personal projects *

Florian Nemling[1] and Renad Quku[1]

Vienna University of Technology, Karlsplatz 1040 Wien, Austria

Abstract. Persistent storages are a very efficient way to save our data. In this paper we want to compare three very well-known Relational Database Management Systems: MySQL, PostgreSQL and OracleDB in three different scenarios. Our focus is database management for small to medium sized personal projects. Firstly, we found an appropriate dataset: "New Passenger Car Registrations by Makes; monthly time series from January 2000 onwards" [1], which includes one main table with more than 17 thousand records and three lookup-tables. We want to benchmark these database management systems based on these three criteria: Response time of a single simple query avoiding caching, handling of concurrent requests and Response Time and efficiency of analytical workloads, commonly found in reporting. After we executed our scripts we found out that for our defined scenarios PostgreSQL outperformed by a large margin Oracle and MySQL.

Keywords: Database Benchmarking · Response time · Aggregation · Handling multi-users simultaneously · MySQL · OracleDB · PostgreSQL

1 Introduction

The scope of this paper is to benchmark three different database management systems: MySQL, PostgreSQL and OracleDB. Therefore we make use of a well known dataset on car registration data and perform three queries with different scopes. Outcomes will be compared and discussed. This paper is structured as follows. First we briefly describe the dataset in use and the queries constructed. Then we summarize the benchmarking outcome. Finally we conclude.

2 Description of Dataset

We make use of the *New Passenger Car Registrations by Makes; monthly time series from January 2000 onwards* dataset which is freely available from the Statistik Austria. Data can be accesed via `http://data.statistik.gv.at/web/meta.jsp?dataset=OGD_fkfzul0759_OD_PkwNZL_1`. This dataset consists

* Supported by Institut für Information Systems Engineering (E194) Compilers and Languages (E194-5) and written for the course "Forschungsmethoden SS2020 (Research methods)"

of one main table containing about 17000 entries and three additional dimension tables. We decided to work with this dataset as it is easy to understand, freely available and we like cars. Additionally we could find out how many London taxis are currently driving in Austria.

3 Database Management Systems in Use

We are using three very well-known Relational Database Management Systems: MySQL, PostgreSQL and OracleDB.

- **OracleDB** Oracle was selected as it's a well renowned contender in the commercial RDBMS market. Due to associated license fees this is the most expensive product to pick.
- **MySQL:** MySQL is maintained by both a community and Oracle who acquired it in 2010 via its purchase of Sun Microsystems.
- **PostgreSQL:** PostgreSQL is software maintained by a open source community. It has its roots in academia and offers a wide range of functionality.

4 Implementation

4.1 Queries in Use

In this section we will post the queries that we are using for our benchmark to test the different criteria.

Q1: Query for individual data records

```
SELECT tf.name, tpm.name, tzm.name
FROM benchmarker.t_neuzulassungen nz
LEFT JOIN benchmarker.t_fahrzeuge tf
    ON nz.fahrzeugen = tf.code
LEFT JOIN benchmarker.t_pkw_marken tpm
    ON tpm.code = nz.pkw_marken
LEFT JOIN benchmarker.t_zeit_monatswerte tzm
    ON tzm.code = nz.zeit_monatswerten
WHERE tzm.name LIKE :quoted;
```

In Q1 is listed the query that is used to test the response times of a single query avoiding caching in all RDBMS covered from this paper. The query selects the name of the vehicle, the month and year where the brand was registered and also the brand of the vehicle. There were used three left joins to join the dimension tables. This query takes a named variable for PostreSQL and MySQL and a positional argument for the OracleDB. The arguments are passed from an array, which include some brands that have at most two records in the registration table. We use exactly this query to also test the concurrency, but the parallelism is taken over from the Powershell script.

Q2: Analytical Query

```
SELECT TZM.NAME, sum(NEUZULASSUNGEN),
    LAG (sum(neuzulassungen)) over (partition
    by SUBSTR(TZM.NAME, 0, INSTR(TZM.NAME, '␣')-1)
    order by tzm.name)
FROM T_NEUZULASSUNGEN NZ
LEFT JOIN T_FAHRZEUGE TF
    ON nz.FAHRZEUGEN = tf.CODE
LEFT JOIN T_PKW_MARKEN TPM
    ON tpm.CODE = nz.PKW_MARKEN
LEFT JOIN T_ZEIT_MONATSWERTE TZM
    ON tzm.CODE = nz.ZEIT_MONATSWERTEN
WHERE tzm.name is not null
GROUP BY TZM.NAME;
```

Q2 lists the query that is used to test the efficiency of the SQL database for different analytical and reporting tasks. Here we use a common requirement of comparing a month's aggregate to the previous year. We are selecting and grouping based on the time period the registration were made and we are interested on the total number of registrations. PostgreSQL does not have the function `INSTR` so instead we are using the built-in function `POSITION`

4.2 Powershell Script

All queries are triggered via a script that fires the repeated requests to the databases. In the test on concurrency performance this script also allows the parallel execution of requests and measurement of response times.

Here is just a relevant snippet from the powershell code we have written to automate the execution of our queries. The whole script can be found in the appendix section. In this snippet we show how we deal with concurrency.

```
1..$no_executions | ForEach-Object -Parallel {

    $ora_results = $using:ora_results


        $ORA_USER           =$using:ORA_USER
        $ORA_PASS           =$using:ORA_PASS
        $ORA_HOST           =$using:ORA_HOST
        $ORA_PORT           =$using:ORA_PORT
        $ORA_DB             =$using:ORA_DB
        $ORA_CLIENT_PATH    =$using:ORA_CLIENT_PATH
        $ORA_CLIENT         =$using:ORA_CLIENT
        $ORA_ARGS           =$using:ORA_ARGS
        $whereclauses       =$using:whereclauses
        $function:ora_run = $using:ora_run_def
        $function:ora_timespan = $using:ora_timespan_def
        $x = get-random -Minimum 0 -Maximum 23
    $ex_time = & $ORA_CLIENT "$ORA_USER`/$ORA_PASS`@$ORA_HOST`/$ORA_DB" "`@ora_c
  Where-Object { $_ -match 'Elapsed'}[0] |
  foreach-object {$_.Split(' ')[1]}

  $ora_results.Add(@{"ora_concurrent.sql"= ora_timespan($ex_time)})

        #$val = ora_run("ora_concurrent.sql", $whereclauses[1])
    #$ora_results.Add($val)

} -ThrottleLimit $throttle
```

5 Benchmarking procedure

We retrieve performance statistics of query structure 1 (Fast and Furious) by running the same query 10 times. We avoid unrealistic cache usage by changing filter criteria for every execution. We include the following summary statistics: minimum execution time, maximum execution time and average execution time.

For the second query (Multi-User Rush Hour) we simulate multiple concurrent users. We are requesting the same queries as in query structure 1 but trigger 200 executions with up to 50 running in parallel.

The third query is a query aggregating data over the entire dataset.

After we let are automated Powershell script run, the results will be written in three CSV files. We used Microsoft Excel, especially the Pivot Table Functionality to analyze the outcome

5.1 Fast and Furious

The first query is a query for and individual data record or a very small data sets. This scenario is relevant for use cases commonly seen in application development where quick response times are essential. The goal is to achieve the best response times.

5.2 Multi-User Rush Hour

Also for the Rush Hour we are using the same query as the one for Fast and Furious but the focus is this time at the handling of simultaneous requests. This scenario is relevant for a database that needs to service concurrent requests. The databases goal should be to service all requests with good response without outliers.

5.3 Tower to Management

For the Tower To Management we are using a typical report task of comparing the outcome of a month with the outcome of the previous year. This scenario focuses on good performance on queries accessing several thousand rows and use of aggregate and windowing functions.

5.4 Devices and programs used

Because the focus of this paper was the right choice of a database management system for small projects the script was executed from a personal laptop. All databases were running on virtual docker containers as provided by the software producers with two cores and 2 GB RAM.

Programs

- `Docker for Windows` to run the containers
- `SQL*PLUS` as Client for Oracle 12
- `mysqlsh` as Client for MySQL 8.0
- `PSQL` as Client for PostgreSQL
- Powershell 7 to use the new feature `ForEach-Object -Parallel`
- Excel 2016 for analyzing the results
- PostgreSQL 12.3 docker image
- Oracle 12c Enterprise Edition docker image
- MySQL 8.0.20 docker image
- `Microsoft Powershell 7`

6 Challenges and Limitations

While preparing the needed test cases, automated scripts and writing of this paper we encountered these challenges

- Oracle reports the execution time of an statement in a number with just two decimal places.
- The import of the needed data records using the default console programs could not be done in a short timed manner, because of the manual changes needed. The manual changes were needed because of the small differences that the database management systems have with each other: Uppercase vs. lowercase; a neccessary use of schema name vs. not using it etc.
- multithreading support in PowerShell turns out to be quite fragile
- The use of the quite new program mysqlsh, which is not still very well documented. mysql could not be used to report the execution time
- The time mysqlsh needs to be invoked is too long (This time does not affect the results, because we are measuring just the execution time)

		Min	Max	Avg
Fast and Furious	Oracle	0,0000	0,0100	0,0090
	MySQL	0,0104	0,0168	0,0138
	PostgreSQL	0,0027	0,0057	0,0035
Multi-User – Rush Hour	Oracle	0,0600	1,9700	0,6007
	MySQL	0,0101	1,5326	0,2937
	PostgreSQL	0,0034	0,0579	0,0115
Tower to Management	Oracle	0,0400	0,0500	0,0420
	MySQL	0,0447	0,0779	0,0572
	PostgreSQL	0,0209	0,0349	0,0259

Table 1. Summary of the results

7 Outcome

Table 1 gives a summary of our Benchmarks.

7.1 Analysis of Fast and Furious

As we can see in the results PostgreSQL is the fastest when it comes to filtering just a very small subset of the whole data. It takes just 3.5 ms to do the executions on average and at most 5.7 ms. The results of Oracle are for this test case not really reliable because of the limitation that the time will be returned as a number with just two decimal places. But as we can see the average time way nearer 10 ms than 5ms and therefore the execution takes more than by Postgres. In this testcase is MySQL clearly the slowest by taking at average around 14 ms.

7.2 Analysis of Rush Hour

As we can see in the results PostgreSQL is the fastest when it comes to reaction time when a considered number of users are accessing the system at the same time. If we compare the least needed time Postgres outperforms Oracle by a lot: 3ms vs. 60 ms. But the difference is even huger when we compare the maximum needed time: Postgres needs at most less than Oracle needs at its best scenario. Unexpectedly Oracle perform in this concurrent scenario as the worst Database Management System by needing at some cases up to 2 seconds. Also MySQL is in this task not particularly strong taking in some cases 1.5 seconds to finish returning the results, but nevertheless stronger than Oracle.

7.3 Analysis of Tower of Management

As we can see in the results PostgreSQL is the fastest also when it comes to using grouped aggregation functions. In average case Postgres needs around 16 ms less to be executed Oracle needs and around 32 ms than MySQL needs.

References

1. T. Conrad: Postgresql vs. mysql vs. commercial databases: It's all about what you need, 2006, - Technical report, Devx
2. Database Engines, `https://db-engines.com/de/`. Last accessed 21.05.2020

8 Appendix

The database scripts that were used can be found in this public Github Repository `https://github.com/renquk10/ForschungsmethodenSS20/tree/master/DB_Skripte`

The implemented benchmark written in Powershell can be found in this public Github Repository `https://github.com/renquk10/ForschungsmethodenSS20/tree/master/Powershell`

The Docker File used to create the containers for each DBMS can be found in this public Github Repository `https://github.com/renquk10/ForschungsmethodenSS20/tree/master/Docker`

Benchmarking databases on small to medium size projects

Florian Nemling[a], Renad Quku[a]

[a] **Institute for Information Systems Engineering (E194)**
Compilers and Languages (E194-5)

Forschungsmethoden SS2020 – Gruppe 2

Intro

The scope of this project is to compare three well-known relational database management systems - MySQL, PostgreSQL and OracleDB – for three different scenarios.
We make use of the “New Passenger Car Registrations by Makes; monthly time series from January 2000 onwards” [1], which includes one main table with more than seventeen thousand records and three lookup-tables.
We benchmark these database management systems on the following three criteria:

1. Response time of a single simple query avoiding caching,
2. Handling of simultaneous request (starting from 50 requests)
3. Response time and efficiency of grouped functions.

Methods and Material

QUERY	DESCRIPTION	SCOPE
1	Query for individual data records	Fast and Furious
2	Many concurrent data requests	MultiUser – Rush Hour
3	Analytical query aggregating several thousand records	Tower to Management

Why use theses 3 databases

Oracle is undoubtedly one of the leading product in the **enterprise world.**

MySQL is one of the most popular database systems driven by a substantial **community** powering **websites and applications**.

PostgreSQL has an academic heritage and still has strong roots in the **academic community**. Moreover, being bundled with numerous *NIX distributions it has become quite **ubiquitous**.

The database systems are being benchmarked in three different scenarios.

Scenario 1: Query for individual data records / very small sets
This scenario is relevant for use cases commonly seen in application development where quick response times are essential. The goal here is to achieve the best response times.

Scenario 2: Concurrent data requests
This scenario is relevant for a database that needs to service concurrent requests. The databases goal should be to service all requests with good response without outliers.

Scenario 3: Analytical query
This scenario focuses on good performance on queries accessing several thousand rows and use of grouped aggregate functions. .

Benchmarking Procedure:
The benchmark numbers for scenarios 1 & 3 are based on samples of 10 queries.
The concurrency performance in scenario 2 was collected with 200 executions and 50 parallel requests.
All databases were running on virtual docker containers as provided by the software producers with two cores and 2 GB RAM.
All Numbers are execution times in seconds, without result transfer or logon/logoff overhead.

SQL & Powershell

```sql
SELECT TF.NAME, TPM.NAME, TZM.NAME
FROM T_NEUZULASSUNGEN NZ
LEFT JOIN T_FAHRZEUGE TF ON nz.FAHRZEUGEN = tf.CODE
LEFT JOIN T_PKW_MARKEN TPM on tpm.CODE = nz.PKW_MARKEN
LEFT JOIN T_ZEIT_MONATSWERTE TZM on tzm.CODE = nz.ZEIT_MONATSWERTEN
where TPM.NAME LIKE :'quoted';
```

```sql
SELECT TZM.NAME, sum(NEUZULASSUNGEN), LAG (sum(neuzulassungen)) over
(partition by SUBSTR(TZM.NAME, 0, INSTR(TZM.NAME, ' ')-1) order by
tzm.name)
FROM T_NEUZULASSUNGEN NZ
LEFT JOIN T_FAHRZEUGE TF ON nz.FAHRZEUGEN = tf.CODE
LEFT JOIN T_PKW_MARKEN TPM on tpm.CODE = nz.PKW_MARKEN
LEFT JOIN T_ZEIT_MONATSWERTE TZM on tzm.CODE = nz.ZEIT_MONATSWERTEN
where tzm.name is not null
GROUP BY TZM.NAME;
```

```powershell
1..$no_executions | ForEach-Object -Parallel {

  $PG_results = $using:PG_results

  ########################################
  ###  Define here the needed variables  ###
  ########################################

  $x = get-random -Minimum 0 -Maximum 23

  $ex_time = & "$PG_CLIENT" "--username=$PG_USER" "--dbname=$PG_DB" "-vquoted=""$whereclauses[$x]""" "--file=pg_simple.sql" |
        Where-Object { $_ -match 'Time:'}[0] |
        foreach-object {$_.Split(' ')[1]}

  $ex_time = [System.Convert]::ToDecimal($ex_time,[cultureinfo]::GetCultureInfo('de_DE'))
  $val = @{"pg_concurrent.sql"= $ex_time/1000}
  $pg_results.Add($val)

} -ThrottleLimit $throttle
```

Outcome and Discussion

seconds		Min	Max	Avg
Fast and Furious	Oracle	0,0000	0,0100	0,0090
	MySQL	0,0104	0,0168	0,0138
	PostgreSQL	0,0027	0,0057	0,0035
Multi-User – Rush Hour	Oracle	0,0600	1,9700	0,6007
	MySQL	0,0101	1,5326	0,2937
	PostgreSQL	0,0034	0,0579	0,0115
Tower to Management	Oracle	0,0400	0,0500	0,0420
	MySQL	0,0447	0,0779	0,0572
	PostgreSQL	0,0209	0,0349	0,0259

Fast and Furious
- PostgreSQL is the clearly the fastest
- Oracle results for this testcase not really reliable, because of the limitation of the return value with just 2 decimal places
- Still slower than Postgres
- MySQL is the slowest with an average of 14 ms

Multi-User – Rush Hour
- PostgreSQL is the clearly the fastest when a lot of users are accessing the system at the same time
- Least needed time: Postgres (3 ms) vs. Oracle (60 ms)
- Postgres as its worst scenario is still faster than Oracle
- Oracle the worst in this scenario: it takes at cases up to 2 seconds
- MySQL is also weak: In some cases it takes up to 1.5 seconds

Tower to Management
- PostgreSQL is the clearly the fastest when using aggregate functions in the queries
- Postgres needs around 16 ms less than Oracle and around 32 ms less than MySQL
- MySQL is clearly the slowest for reporting tasks
- MySQL needs at its worst case around 30 ms more than Oracle and around 40 ms more than PostgreSQL

Contact

Florian Nemling, Renad Quku
florian@nemling.eu , renadquk@gmail.com
Technische Universität Wien
Institute for Information Systems Engineering (E194)
Compilers and Languages (E194-5)
Argentinierstraße 8,1040 Wien - AT