

BACHELORARBEIT

Performance – Optimierung von Datenbanken

vorgelegt am 26. März 2022
Daniel Freire Mendes

Erstprüferin: Prof. Dr. Stefan Sarstedt
Zweitprüfer: Prof. Dr. Olaf Zukunft

**HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG**

Department Informatik
Berliner Tor 7
20099 Hamburg

Zusammenfassung

Der Arbeit beginnt mit einer kurzen Beschreibung ihrer zentralen Inhalte, in der die Thematik und die wesentlichen Resultate skizziert werden. Diese Beschreibung muss sowohl in deutscher als auch in englischer Sprache vorliegen und sollte eine Länge von etwa 150 bis 250 Wörtern haben. Beide Versionen zusammen sollten nicht mehr als eine Seite umfassen. Die Zusammenfassung dient u. a. der inhaltlichen Verortung im Bibliothekskatalog.

Abstract

The thesis begins with a brief summary of its main contents, outlining the subject matter and the essential findings. This summary must be provided in German and in English and should range from 150 to 250 words in length. Both versions combined should not comprise more than one page. Among other things, the abstract is used for library classification.

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Tabellenverzeichnis	III
1 Überblick	1
1.1 Einführung in Benchmarks	1
1.2 Measures	2
1.3 Tools	3
1.3.1 Einführung	3
1.3.2 Einführung in die Tools	4
2 Projektdurchführung	14
2.1 GitHub Action	14
3 Indexierung und Einfluss auf die Performance	15
3.1 Grundlagen der Indexierung	15
3.2 B-Baum-Index	16
3.3 Hash - Index	18
Literatur	23
Anhang	24

Abbildungsverzeichnis

1.1	Pandas - Beispiel	12
1.2	Gnuplot - Beispiel	13
3.1	Binärbaum - Grafik	17
3.2	High-Counts: Reads und Writes	18
3.3	Low-Counts: Reads und Writes	19
3.4	B - Tree - Selects - Ergebnis	19
3.5	Hash-Kollisionen: Reads und Writes	21
3.6	Hash - Selects - Ergebnis	22

Tabellenverzeichnis

1 Überblick

1.1 Einführung in Benchmarks

Benchmarks dienen dazu, praktisch und effektiv zu untersuchen, wie sich ein System unter Last verhält. Die wichtigste Erkenntnis, die man aus Benchmarks gewinnen kann, sind die Probleme und Fehler, die man systematisch dokumentieren und nach Priorität abarbeiten sollte. Das Ziel von Benchmarks ist die Reduzierung und Bewertung von unerwünschtem Verhalten sowie die Analyse, wie sich das System derzeit und unter simulierten, zukünftigen, anspruchsvolleren Bedingungen verhalten könnte.

Es gibt zwei verschiedene Techniken für Benchmarks. Die erste zielt darauf ab, die Applikation als Ganzes zu testen (full-stack). Dabei wird nicht nur die Datenbank getestet, sondern die gesamte Applikation, einschließlich des Webserver, des Netzwerks und des Applikationscodes. Der Ansatz dahinter ist, dass ein Nutzer genauso lange auf eine Abfrage warten muss, wie das gesamte System benötigt. Daher sollte diese Wartezeit so gering wie möglich sein. Es kann dabei vorkommen, dass MySQL nicht immer das Bottleneck ist.¹

Full-Stack-Benchmarks haben jedoch auch Nachteile. Sie sind schwieriger zu erstellen und insbesondere schwieriger korrekt einzurichten. Wenn man lediglich verschiedene Schemas und Abfragen in MySQL auf ihre Performance testen möchte, gibt es sogenannte Single-Component-Benchmarks. Diese analysieren ein spezifisches Problem in der Applikation und sind deutlich einfacher zu erstellen. Ein weiterer Vorteil besteht darin, dass nur ein Teil des gesamten Systems getestet wird, wodurch die Antwortzeiten kürzer sind und man schneller Ergebnisse erhält.

Wenn bei Benchmarks schlechte Designentscheidungen getroffen werden, kann dies zu einer falschen Interpretation des Systems führen, da die Ergebnisse nicht die Realität widerspiegeln. Die Größe des Datensatzes und des Workloads muss realistisch sein. Idealerweise verwendet

¹Gemeint ist ein Engpass beim Transport von Daten oder Waren, der maßgeblichen Einfluss auf die Abarbeitungsgeschwindigkeit hat. Optimierungsversuche an anderer Stelle führen oft nur zu geringen oder gar keinen messbaren Verbesserungen der Gesamtsituation. (Vogel, 2009)

man einen Snapshot² des tatsächlichen produktiven Datensatzes. Gibt es keine Produktionsdaten, sollten die Daten und der Workload simuliert werden, da realistische Benchmarks komplex und zeitaufwendig sein können.

Häufige Fehler beim Durchführen von Benchmarks sind unter anderem, dass nur ein kleiner Teil der tatsächlichen Datensatzgröße verwendet wird und die Datensätze unkorrekt gleichmäßig verteilt sind. In der Realität können Hotspots auftreten. Bei zufällig generierten Werten kommt es hingegen häufig zu unrealistisch gleichmäßig verteilten Datensätzen. Ein weiterer Fehler besteht darin, dass man beim Testen einer Anwendung nicht das tatsächliche Benutzerverhalten nachstellt. Wenn gleiche Abfragen in einer Schleife ausgeführt werden, muss man außerdem auf das Caching achten, da sonst falsche Annahmen über die Performance getroffen werden können. Zudem wird oft die Warmmachphase des Systems vollständig ignoriert. Kurze Benchmarks können schnell zu falschen Annahmen über die Performance des Systems führen.

Um verlässliche Ergebnisse zu erhalten, sollte ein Benchmark ausreichend lange laufen, um den stabilen Zustand des Systems zu beobachten, insbesondere bei Servern mit großen Datenmengen und viel Speicher. Dabei ist es wichtig, so viele Informationen wie möglich zu erfassen und sicherzustellen, dass der Benchmark wiederholbar ist, da unzureichende oder fehlerhafte Tests wertlos sind. Außerdem ist es wichtig, die Ergebnisse in einem Diagramm darzustellen, da auftretende Phänomene sonst anhand einer tabellarischen Darstellung nicht erkannt werden können.

1.2 Measures

- **Durchsatz (Throughput):** Der Durchsatz ist die Anzahl an Transaktionen pro Zeiteinheit. Er ist standardisiert, und Datenbankanbieter versuchen, diesen zu optimieren. Meistens werden Transaktionen pro Sekunde (oder manchmal pro Minute) als Einheit verwendet.
- **Antwortzeiten (Latenz):** Die Antwortzeit misst die gesamte Zeit, die für eine Abfrage benötigt wird. Diese kann, abhängig von der Applikation, in Mikrosekunden (μ s), Millisekunden (ms), Sekunden oder Minuten angegeben werden. Von dieser Zeit können aggregierte Antwortzeiten wie Durchschnitt, Maximum, Minimum und Perzentile abgeleitet werden. Das Maximum ist oft eine weniger sinnvolle Metrik, da es sich nicht gut wiederholen lässt. Daher nutzt man eher Perzentile bei den Antwortzeiten. Wenn beispielsweise das 95. Perzentil der Antwortzeit bei 5 ms liegt, bedeutet dies, dass mit einer Wahrscheinlichkeit von 95 % die Abfrage in weniger als 5 ms abgeschlossen ist.

²Snapshots bestehen größtenteils aus Metadaten, die den Zustand Ihrer Daten definieren, und sind keine vollständige Duplikation der Daten auf Ihrer Festplatte. Snapshots werden häufig für Test-/Entwicklungsaufgaben verwendet. (Germany, 2024)

- **Nebenläufigkeit (Concurrency):** Die Nebenläufigkeit auf dem Webserver lässt sich nicht zwangsläufig auf den Datenbankserver übertragen. Eine genauere Messung der Gleichzeitigkeit auf dem Webserver besteht darin, zu bestimmen, wie viele gleichzeitige Anfragen zu einem bestimmten Zeitpunkt ausgeführt werden. Es kann auch geprüft werden, ob der Durchsatz sinkt oder die Antwortzeiten steigen, wenn die Gleichzeitigkeit zunimmt. Beispielsweise benötigt eine Website mit „50.000 Benutzern gleichzeitig“ vielleicht nur 10 oder 15 gleichzeitig laufende Abfragen.
- **Skalierbarkeit (Scalability):** Skalierbarkeit ist wichtig für Systeme, die ihre Performance unter unterschiedlich starken Workloads beibehalten müssen. Ein ideales System würde doppelt so viele Abfragen beantworten (Throughput), wenn doppelt so viele „Arbeiter“ versuchen, die Aufgaben zu erfüllen. Die meisten Systeme sind jedoch nicht linear skalierbar und zeigen Leistungseinbußen, wenn die Parameter variieren.

1.3 Tools

1.3.1 Einführung

Als Haupttool, um Benchmarktests durchzuführen, habe ich mich für Sysbench [akopytov, 2024](#) entschieden. Sysbench ist ein Open-Source-Tool, das ein skriptfähiges, multi-threaded Benchmark-Tool ist, das auf LuaJIT basiert. Es wird auch hauptsächlich für Datenbankbenchmarks verwendet, kann jedoch auch dazu eingesetzt werden, beliebig komplexe Arbeitslasten zu erstellen, die keinen Datenbankserver erfordern. Dabei werden Tests auf verschiedenen Systemressourcen, wie CPU, Speicher, I/O und Datenbanken wie MySQL Reimers, [2017](#) verwendet.

Im Zuge der Recherchearbeit habe ich mir auch andere Benchmarking-Tools betrachtet, wie z.B. Benchbase [Difallah et al., 2013](#) oder mybench [Shopify, 2024](#). Die größten Vorteile von Sysbench habe ich in der Skriptfähigkeit und Flexibilität gesehen. D.h. dass ich benutzerdefinierte Benchmarks schneller und unkompliziert erstellen kann. Außerdem hat sich Sysbench als de facto Standard im Bereich der Datenbankbenchmarks etabliert [Shopify, 2022b](#). Dadurch stehen eine breite Nutzerbasis und viele verfügbare Ressourcen zur Verfügung. Im Vergleich zu den anderen Tools bietet allerdings Sysbench eine weniger präzise Steuerung der Ergebnisrate und der Transaktionen. Außerdem haben Tools wie mybench die Möglichkeit, in Echtzeit umfassende Visualisierungen darzustellen. Damit können Metriken live in einem Diagramm angezeigt werden [Shopify, 2022a](#). Dieses Feature ist sicherlich hilfreich, aber in meinem Fall habe ich abgewogen und bin zu dem Entschluss gekommen, dass die einfachere Bedienung für mich der ausschlaggebende Grund, neben dem Fakt, dass Sysbench der de facto Standard ist.

Trotzdem kann man nicht komplett auf Graphen verzichten, da beispielweise Entwicklungen im Laufe einer Zeitmessung in einem Kurvenverlauf deutlich besser zu erkennen sind als in einer CSV-Datei. Anhand der reinen Zahlen aus diesen Tabellen fallen diese wiederkehrende Trends unter anderem nicht direkt auf. Die Kennzahlen, die mithilfe von Sysbench ermittelt werden, werden in einer CSV-Datei gespeichert. Um diese tabellarische Form in eine Grafische umzuwandeln, gibt es unterschiedliche Tools, die wiederum eigene Vor- und Nachteile bieten.

Die erste mögliche Alternative stellt das Tool Gnuplot Williams et al., 2024 dar. Mit diesem lassen sich CSV-Dateien sehr gut darstellen. Wenn man aber beispielweise nur bestimmte Spalten aus der Tabelle anzeigen lassen will, dann kommt man schnell an seine Grenzen. Um besser Anpassungsfähig sein zu können, habe ich mich letztlich dazu entschieden ein eigenes Python-Script zu schreiben, die mithilfe der Libraries pandas (//TODO: find source) matplotlib.pyplot (//TODO: find source) die Graphen erstellt.

1.3.2 Einführung in die Tools

Als allererstes muss der MySQL-Server (oder eine anderes relationales Datenbankverwaltungssystem, das von Sysbench unterstützt wird) lokal auf dem Rechner gestartet sein. Wichtig ist dabei die User -und Passwortdaten zu merken, da diese von den Sysbench - Benchmarks benötigt werden. Nachdem das RDBMS gestartet worden ist, muss zudem eine Datenbank erstellt werden. Dies könnte unter anderem so aussehen:

```
1 CREATE DATABASE sbtest;
```

Nachdem man die Datenbank erstellt hat, muss das Tool Sysbench zunächst installiert werden. Als nächstes machen wir uns mit dem Tool und den verschiedenen Argumenten, die beim Aufruf mitübergeben werden müssen oder können, vertraut. Hier ist eine Auflistung mit den übergebenen Argumenten:

- `--db-driver`: Gibt den Treiber für die Datenbank an, die Sysbench verwenden soll. In diesem Fall `mysql`, um MySQL-Datenbanken zu testen.
- `--mysql-host`: Der Hostname oder die IP-Adresse des MySQL-Servers. Standardmäßig wird `localhost` verwendet, wenn nichts angegeben wird.
- `--mysql-user`: Der Benutzername, mit dem Sysbench auf die MySQL-Datenbank zugreift.
- `--mysql-password`: Das Passwort für den MySQL-Benutzer. Falls der Benutzer kein Passwort hat oder der Zugriff über eine andere Authentifizierungsmethode erfolgt, kann dieses Argument weggelassen werden.

- `--mysql-db`: Der Name der MySQL-Datenbank, auf die zugegriffen wird. In diesem Beispiel `sbtest`.
- `--time`: Gibt die Laufzeit des Benchmarks in Sekunden an und muss immer mit angegeben werden.
- `--report-interval`: Gibt das Intervall in Sekunden an, in dem Zwischenergebnisse während des Tests ausgegeben werden. Sofern `--report-interval` nicht gesetzt wird, werden die Ergebnisse erst am Ende des Tests angezeigt.
- `--tables`: Die Anzahl der Tabellen, die für den Test erstellt werden sollen. Standardmäßig wird nur eine Tabelle erstellt.
- `--table-size`: Die Anzahl der Datensätze (Zeilen) pro Tabelle. Muss auch nicht zwingend angegeben werden.

Neben den sieben aufgelisteten Argumenten gibt es zwei weitere wichtige Optionen:

1. Wie im Abschnitt ?? erwähnt, kann ein Lua-Skript angegeben werden, um eigene Tabellen zu erstellen, Beispieldaten einzufügen und bestimmte Abfragen durchzuführen. Dazu muss am Ende der Sysbench-Befehlszeile lediglich der Pfad zur Lua-Datei hinzugefügt werden. Ein erklärendes Beispiel dazu folgt weiter unten in diesem Abschnitt.
2. Die Methode, den Sysbench ausführen soll, muss ebenfalls spezifiziert werden. Auch dieser wird am Ende der Sysbench-Befehlszeile angehängt.

Zunächst schauen wir ein kurzes Demo-Beispiel, denn es gibt die Möglichkeit die Datenbank auf Performance zu testen, ohne selbst eigene SQL-Befehle zu schreiben. Dafür gibt es vordefinierte Testtypen von Sysbench. Auf diese Weise kann man schnell die Korrektheit der Einrichtung des Tools überprüfen, bevor man Lua-Skripts für die eigenen Bedürfnisse schreibt.

Man kann unter anderen zwischen diesen Testtypen wählen:

- **oltp_insert**: Prüft die Fähigkeit der Datenbank, Daten schnell und effizient einzufügen und simuliert eine Umgebung, in der viele Schreiboperationen ausgeführt werden.
- **oltp_read_only**: Fokussiert sich auf die Performance bei Leseoperationen und eignet sich, um die Leistung bei einer rein lesenden Arbeitslast zu testen.
- **oltp_read_write**: Simuliert eine realistische Arbeitslast, bei der sowohl Lese- als auch Schreiboperationen gleichzeitig durchgeführt werden.

Des Weiteren gibt es auch unterschiedliche Methoden, die mit den Testtypen kombiniert werden können.

•

- **prepare:** Bereitet die Datenbank für den Test vor, u.a. das Einfügen von benötigten Datensätze.
- **run:** Ist die Ausführungsphase des Tests. Je nach Testtyp führt diese Methode die spezifizierten Operationen aus, wie etwa das Einfügen von Daten (oltp_insert), das Abfragen von Daten (oltp_read_only) oder beides (oltp_read_write). Dabei wird die Performance der Datenbank unter der angegebenen Arbeitslast gemessen.
- **cleanup:** Diese Methode sorgt dafür, dass nach Abschluss des Tests alle Testdaten entfernt werden. Sie stellt die Datenbank in ihren ursprünglichen Zustand zurück und stellt sicher, dass keine Testdaten zurückbleiben, die eine mögliche produktive Umgebung beeinträchtigen könnten.

Für das Demo-Beispiel wählen wir den Testtypen **oltp_read_write** und allen Methoden aus. Für die Methode run würde unsere Query so aussehen, wobei YOUR_USER und YOUR_PASSWORD entsprechend ersetzt werden müssten:

```
1 sysbench oltp_read_write \
2   --db-driver=mysql \
3   --mysql-user=YOUR_USER \
4   --mysql-password=YOUR_PASSWORD \
5   --mysql-db="sbtest" \
6   --time=10 \
7   --report-interval=1 \
8   run
```

Wenn man nur diese Query ausführt, fällt er auf, dass die Query scheitert. Deshalb bietet es sich an ein Shell-Script zu schreiben, indem zuerst prepare aufgerufen wird und als Nächstes erst run. Die Ergebnisse der Log-Datei speichert man sich dann in einer Datei und aus dieser Datei erstellt man eine CSV-Datei, mit der man später die Graphen erstellen lässt. Und als letzten Schritt ruft man die cleanup-Methode auf, damit bei erneuter Ausführung keine Fehler entstehen, bzw. die Produktivumgebung nicht gestört wird, wenn diese sonst beeinflusst werden würde.

Dies ist das Shell-Script, dass zuständig ist für den kompletten Ablauf:

Codeblock 1.1: Sysbench Script

```
1 #!/bin/bash
2
3 # File Paths
4 GENERATE_PLOT_SCRIPT="/Users/danielmendes/Desktop/Bachelorarbeit/Ausarbeitung/Tools/
  Pandas/generateplot.py"
```

```

5 OUTPUT_DIR="output"
6 OUTPUT_FILE="output/sysbench_output.csv"
7 RAW_RESULTS_FILE="output/sysbench.log"
8 GNUPLOT_SCRIPT="plot_sysbench.gp"
9
10 # Connection parameters
11 DB_USER="root"
12 DB_PASS="password"
13 DB_NAME="sbtest"
14 TABLES=10
15 TABLE_SIZE=10000
16 DURATION=10
17
18 # Ensure output directories exist
19 rm -rf "$OUTPUT_DIR"
20 mkdir -p "$OUTPUT_DIR"
21
22 # Function to run sysbench with parameters
23 run_sysbench() {
24     local MODE="$1"
25     local EXTRA_ARGS="$2"
26     local LOG_FILE="$3"
27
28     sysbench oltp_read_write \
29         --db-driver=mysql \
30         --mysql-user="$DB_USER" \
31         --mysql-password="$DB_PASS" \
32         --mysql-db="$DB_NAME" \
33         --tables="$TABLES" \
34         --table-size="$TABLE_SIZE" \
35         $EXTRA_ARGS \
36         $MODE >> "$LOG_FILE" 2>&1
37
38     return $?
39 }
40
41 echo "Preparing the database..."
42 run_sysbench "prepare" "" "$RAW_RESULTS_FILE"
43 echo "Database prepared."
44
45 echo "Running benchmark..."
46 run_sysbench "run" "--time=$DURATION --threads=1 --report-interval=1" "

```

```

    $RAW_RESULTS_FILE"
47 echo "Benchmark complete. Results saved to $OUTPUT_FILE."
48
49 # Format the results into CSV
50 echo "Time (s),Threads,TPS,QPS,Reads,Writes,Other,Latency (ms;95%),ErrPs,ReconnPs" >
    "$OUTPUT_FILE"
51 grep '^\[ ' $RAW_RESULTS_FILE | while read -r line; do
52     time=$(echo $line | awk '{print $2}' | sed 's/s//')
53     threads=$(echo $line | awk -F 'thds: ' '{print $2}' | awk '{print $1}')
54     tps=$(echo $line | awk -F 'tps: ' '{print $2}' | awk '{print $1}')
55     qps=$(echo $line | awk -F 'qps: ' '{print $2}' | awk '{print $1}')
56     read_write_other=$(echo $line | sed -E 's/.*\(r\w\o: ([0-9.]+)\|([0-9.]+)
        \|([0-9.]+)\).*\/\1,\2,\3/')
57     reads=$(echo $read_write_other | cut -d',' -f1)
58     writes=$(echo $read_write_other | cut -d',' -f2)
59     other=$(echo $read_write_other | cut -d',' -f3)
60     latency=$(echo $line | awk -F 'lat \\\(ms,95%\\\): ' '{print $2}' | awk '{print $1
        }')
61     err_per_sec=$(echo $line | awk -F 'err/s: ' '{print $2}' | awk '{print $1}')
62     reconn_per_sec=$(echo $line | awk -F 'reconn/s: ' '{print $2}' | awk '{print $1
        }')
63
64     echo "$time,$threads,$tps,$qps,$reads,$writes,$other,$latency,$err_per_sec,
        $reconn_per_sec" >> "$OUTPUT_FILE"
65 done
66
67 echo "Cleaning up..."
68 run_sysbench "cleanup" "" "$RAW_RESULTS_FILE"
69 echo "Database cleanup complete."
70
71 # Generate plot with gnuplot
72 rm -rf "$OUTPUT_DIR/gnuplot"
73 mkdir -p "$OUTPUT_DIR/gnuplot"
74 echo "Generating plot with gnuplot..."
75 gnuplot $GNUPLOT_SCRIPT
76
77 # Generate plot with pandas and move objects
78 echo "Generating plots with pandas..."
79 python3 "$GENERATE_PLOT_SCRIPT" "$OUTPUT_FILE"
80 SOURCE_DIR="output/detailed_pngs"
81 DEST_DIR="output/pandas"
82 FILE_TO_MOVE="output/output_final.png"

```

```

83 mkdir -p "$DEST_DIR"
84 mv "$SOURCE_DIR"/* "$DEST_DIR"
85 mv "$FILE_TO_MOVE" "$DEST_DIR/Summary.png"
86 rm -rf "$SOURCE_DIR"
87
88 echo "Plots generated."

```

Codeblock 1.2: Gnuplot Script

```

1 set datafile separator ","
2 set title "Benchmark Results: TPS, Latency, Queries, and More"
3 set xlabel "Time (s)"
4 set ylabel "Values"
5 set grid
6 set key outside
7 set terminal pngcairo enhanced font 'Arial,10'
8 set output "/Users/danielmendes/Desktop/Bachelorarbeit/Ausarbeitung/Tools/Output/
  sysbench_output_plot.png"
9 set yrange [0:~]
10
11 # Plot each attribute on its own line
12 plot "/Users/danielmendes/Desktop/Bachelorarbeit/Ausarbeitung/Tools/Output/
  sysbench_output.csv" using 1:2 title "Threads" lt 1 lc rgb "black" with lines, \
13   "" using 1:3 title "TPS" lt 2 lc rgb "green" with lines, \
14   "" using 1:4 title "QPS" lt 3 lc rgb "blue" with lines, \
15   "" using 1:5 title "Reads" lt 4 lc rgb "red" with lines, \
16   "" using 1:6 title "Writes" lt 5 lc rgb "orange" with lines, \
17   "" using 1:7 title "Other" lt 6 lc rgb "purple" with lines, \
18   "" using 1:8 title "Latency (ms)" lt 7 lc rgb "cyan" with lines, \
19   "" using 1:9 title "Err/s" lt 8 lc rgb "magenta" with lines, \
20   "" using 1:10 title "Reconn/s" lt 9 lc rgb "brown" with lines

```

Das Python-Skript, das zuständig ist für die Graphgenerierung muss als Argument zum einen die CSV-Datei übermittelt bekommen und zum anderen kann es nur eine bestimmte Auswahl an Messwerten übergeben, damit nur für diese die Graphen erzeugt werden. Dies ist das zuständige Python-Skript:

Codeblock 1.3: Pandas Graph Generator

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import os

```

```

4 import argparse
5
6 def parse_arguments():
7     parser = argparse.ArgumentParser(description='Generate plots from CSV data.')
8     parser.add_argument('datafile', type=str, help='Path to the input CSV data file'
9     )
10    parser.add_argument('metrics', type=str, nargs='*', help='List of metrics to
11    plot (e.g., QPS Reads Writes). If empty, all metrics will be used.')
12    return parser.parse_args()
13
14 def plot_metrics(data, measures, output_dir, detailed_pngs_dir):
15     has_script_column = 'Script' in data.columns
16
17     if has_script_column:
18         scripts = data['Script'].unique()
19     else:
20         scripts = [None]
21
22     plt.figure(figsize=(10, 6))
23
24     for measure in measures:
25         if has_script_column:
26             # Plot each script as a separate line for each measure if 'Script'
27             column exists
28             for script in scripts:
29                 script_data = data[data['Script'] == script]
30                 plt.plot(script_data['Time (s)'], script_data[measure], label=f"{
31                 script} - {measure}")
32         else:
33             # Plot only the measure if no 'Script' column exists
34             plt.plot(data['Time (s)'], data[measure], label=measure)
35
36     plt.title('Metrics over Time' + (' by Script' if has_script_column else ''))
37     plt.xlabel('Time (s)')
38     plt.ylabel('Values')
39     plt.legend(title="Script and Measure" if has_script_column else "Measure")
40     plt.grid(True)
41
42     # Save the combined plot
43     output_final_path = os.path.join(output_dir, 'output_final.png')
44     plt.savefig(output_final_path)
45     plt.close()

```

```

42
43 for measure in measures:
44     plt.figure(figsize=(10, 6))
45     if has_script_column:
46         # Plot each script as a separate line for each measure if 'Script'
column exists
47         for script in scripts:
48             script_data = data[data['Script'] == script]
49             plt.plot(script_data['Time (s)'], script_data[measure], label=f"{
script} - {measure}")
50     else:
51         # Plot only the measure if no 'Script' column exists
52         plt.plot(data['Time (s)'], data[measure], label=measure)
53
54     # Plot settings for individual figures
55     plt.title(f'{measure} over Time by Script')
56     plt.xlabel('Time (s)')
57     plt.ylabel(measure)
58     plt.legend(title="Script")
59     plt.grid(True)
60
61     # Save the detailed plot to a PNG file in the specified output directory
62     detailed_output_file_path = os.path.join(detailed_pngs_dir, f"{measure}.png"
)
63     plt.savefig(detailed_output_file_path)
64     plt.close()
65
66 def main():
67     args = parse_arguments()
68
69     # Load CSV data
70     datafile = args.datafile
71     if not os.path.isfile(datafile):
72         raise FileNotFoundError(f"The file {datafile} does not exist.")
73
74     data = pd.read_csv(datafile)
75
76     # Determine metrics to plot
77     if args.metrics:
78         measures = args.metrics
79     else:
80         # Use all columns except 'Time (s)' and 'Script' as metrics

```



```

81     measures = [col for col in data.columns if col not in ['Time (s)', 'Script'
82 ]]
83
84     # Define output directory for plots
85     output_dir = os.path.dirname(datafile)
86     detailed_pngs_dir = os.path.join(output_dir, 'detailed_pngs')
87
88     # Create output directories if they don't exist
89     os.makedirs(detailed_pngs_dir, exist_ok=True)
90     plot_metrics(data, measures, output_dir, detailed_pngs_dir)
91
92 if __name__ == '__main__':
93     main()

```

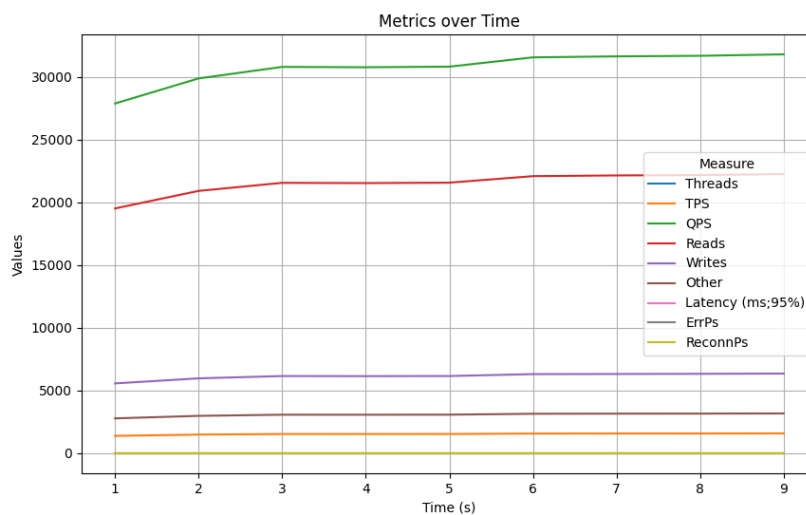


Abbildung 1.1: Grafik generiert mithilfe des Pythontools Pandas

- **Threads:** Die Anzahl der gleichzeitig verwendeten Threads. Mehr Threads können die Parallelität erhöhen, jedoch kann eine zu hohe Anzahl die Leistung beeinträchtigen, wenn das System überlastet wird.
- **TPS (Transactions Per Second):** Die Anzahl der Transaktionen pro Sekunde. Ein höherer Wert deutet auf eine bessere Datenbankleistung hin.
- **QPS (Queries Per Second):** Die Anzahl der Abfragen pro Sekunde. Ein höherer Wert ist besser und zeigt die Effizienz bei der Verarbeitung von Abfragen.
- **Reads:** Die Anzahl der Leseoperationen. Mehr Leseoperationen sind im Allgemeinen besser, da sie eine höhere Datenauslastung anzeigen, was jedoch auch vom spezifischen

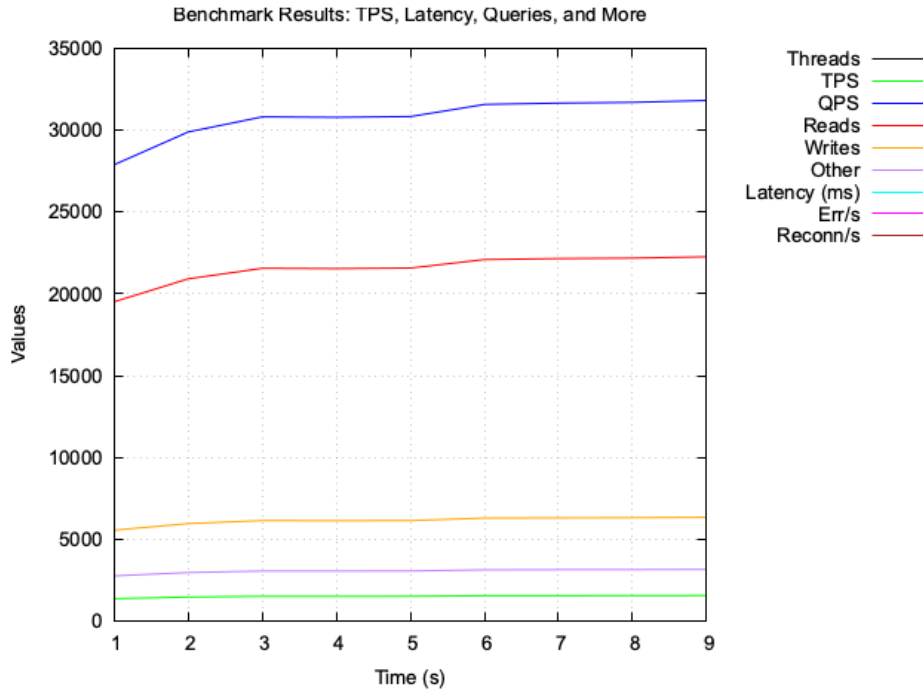


Abbildung 1.2: Grafik generiert mithilfe von Gnuplot

Anwendungsfall abhängt.

- **Writes:** Die Anzahl der Schreiboperationen. Ähnlich wie bei den Leseoperationen: Mehr Schreibvorgänge sind besser, solange die Performance erhalten bleibt.
- **Other:** Bezieht sich auf andere Arten von Operationen, die weder als Reads noch als Writes kategorisiert werden. Ein höherer Wert ist gut, solange er nicht zu einer Überlastung führt.
- **Latency (ms; 95%):** Die durchschnittliche Zeit in Millisekunden, die benötigt wird, um Anfragen zu bearbeiten, wobei der Wert im 95. Perzentil betrachtet wird. Niedrigere Werte sind besser, da sie auf schnellere Reaktionszeiten hinweisen.
- **ErrPs (Errors Per Second):** Die Anzahl der Fehler pro Sekunde. Ein niedriger Wert ist wünschenswert, da er auf eine höhere Stabilität und Zuverlässigkeit des Systems hinweist.
- **ReconnPs (Reconnects Per Second):** Die Anzahl der Wiederverbindungen pro Sekunde. Ein niedrigerer Wert ist ebenfalls besser, da häufige Wiederverbindungen auf Stabilitätsprobleme hindeuten können.

2 Projektdurchführung

2.1 GitHub Action

Tried these here: - GitHub Artifacts - GitHub Cache => deprecated after 7 days and only works - (GitHub Repo) or dedicated feature branch => give github action write permission so push is allowed - (Google Cloud Storage (GSC), AWS S3 or Azure Storage)

Im Laufe des Projekts hat sich herausgestellt, dass viele Bash_Befehle ausgeführt werden müssen. Da das Ausführen des Scripts und damit das Durchführen der Benchmarks führen zu hohen Lasten auf dem lokalen Rechner, weshalb ich nicht jedes Mal alle Skripte ausführen konnte, ohne erhebliche Wärmeveränderungen an meinem Rechner zu spüren. Außerdem hat sich herausgestellt, dass jede Erweiterung an meinen Hauptskripten `generateCombinedCSV.py`, `generatePlot.py` und `sysbench_script.sh` Auswirkungen auf alle auszuführenden Skripten haben kann. Mit zunehmender Skriptanzahl, war es nicht immer auf dem ersten Blick erkennbar, welches Outputdateien der Skripte fehlerbehaftet geworden sind im Vergleich zu dem Lauf davor. Deshalb hat es für mich Sinn ergeben, das Ganze auszulagern. Als sinnvolle Alternative zu dem lokalen Ausführen, haben sich GitHub Actions für mich angeboten. (TODO: Erklärung GitHub Action) Vereinfacht gesagt soll die GitHub Action alle Skripte ausführen und am Ende alle Outputdateien in einen Ordner zusammen als GitHub Artifact hochladen. Anschließend kann ich die Zip-Datei einfach herunterladen, entzippen und anschließend überprüfen, ob alle Dateien noch stimmen.

Man hätte das Ganze noch weiterführen können, in dem man bestimmte Tests durchführt die bestimmte Werte beispielsweise in den CSV - Dateien erwarten.

3 Indexierung und Einfluss auf die Performance

3.1 Grundlagen der Indexierung

Das folgende Thema befasst sich mit der Indexierung und den damit verbundenen Performance-Optimierungen, die näher erläutert werden. Zunächst betrachten wir die Grundlagen der Indexierung, anschließend die verschiedenen Arten von Indizes und schließlich deren Auswirkungen auf die Performance.

Indizes (oder auch Indexes) sind Datenstrukturen, die von Speicher-Engines (engl. storage engines) verwendet werden, um unter anderem Zeilen schneller zu finden. Sie haben einen großen Einfluss auf die Performance der Datenbank und werden umso wichtiger, je größer die Datenbank wird. Weniger ausgelastete Datenbanken können ohne ordnungsgemäße Indizes gut funktionieren, aber die Leistung kann rapide sinken, wenn die Datenmenge wächst. Wenn ein solches Problem auftritt, ist die Index-Optimierung oft der effektivste Weg, die Abfrageleistung zu verbessern. Um wirklich optimale Indizes zu erstellen, ist es häufig notwendig, Abfragen umzuschreiben. Wie genau Indizes erstellt werden müssen, wird im weiteren Verlauf der Arbeit betrachtet.

Um die Funktionsweise eines Indexes zu verdeutlichen, betrachten wir ein Beispiel aus einem wissenschaftlichen Fachbuch. Am Ende solcher Bücher gibt es meist ein Stichwortverzeichnis oder Register. Dieses Register besteht aus einer alphabetisch geordneten Liste von Begriffen, Themen und Stichworten. Möchte man einen Begriff nachschlagen, sucht man ihn in der Liste und erhält die Seitenzahlen, auf denen er vorkommt. In MySQL verwendet die Storage-Engine Indizes auf ähnliche Weise. Sie durchsucht die Datenstruktur des Indexes nach einem Wert. Wird ein Treffer gefunden, kann die Engine die Zeile ermitteln, die den Treffer enthält. Betrachten wir dazu folgendes Beispiel:

Codeblock 3.1: Variationen

```
1 SELECT name FROM customer WHERE cust_id = 7;
```

Es gibt einen Index auf der Spalte `cust_id`, sodass MySQL diesen Index nutzt, um Zeilen zu finden, deren `cust_id` gleich 7 ist. Mit anderen Worten wird eine Suche innerhalb der Indexwerte durchgeführt, und alle entsprechenden Zeilen werden zurückgegeben.

Ein Index kann Werte aus einer oder mehreren Spalten einer Tabelle enthalten. Bei mehreren Spalten ist die Reihenfolge der Spalten im Index entscheidend, da MySQL nur effizient auf ein linkes Präfix des Indexes zugreifen kann. Ein Index über zwei Spalten ist nicht gleichbedeutend mit zwei separaten einspaltigen Indizes. Es gibt verschiedene Typen von Indizes, die jeweils für unterschiedliche Zwecke optimiert sind und die im nächsten Abschnitt behandelt werden.

3.2 B-Baum-Index

Indizes werden auf der Ebene der Storage-Engine und nicht auf der Serverebene implementiert. Daher sind sie nicht standardisiert und unterscheiden sich je nach Engine. Zudem unterstützen nicht alle Engines alle Index-Typen. Eine Storage-Engine ist eine Kernkomponente eines Datenbankmanagementsystems (DBMS), die für die Speicherung und Verwaltung der Daten zuständig ist. Sie entscheidet, wie Daten physisch organisiert, gespeichert und abgerufen werden. Verschiedene Storage-Engines unterscheiden sich in ihrer Indexfunktionalität sowie in der Unterstützung von Transaktionen und Sperrmechanismen.

Der erste zu betrachtende Indextyp ist der B-Baum-Index (engl. B-Tree Index), der auf einer speziellen Baum-Datenstruktur basiert. Diese Struktur wird von den meisten MySQL-Storage-Engines unterstützt. Die Implementierung und Nutzung des B-Baum-Indexes kann jedoch je nach verwendeter Storage-Engine variieren.

Das Grundprinzip eines B-Baums ist, dass alle Werte in einer bestimmten Reihenfolge gespeichert werden und jede Blattseite den gleichen Abstand zum Wurzelknoten hat. Ein B-Baum-Index beschleunigt den Datenzugriff, da die Storage-Engine nicht die gesamte Tabelle durchsuchen muss, um die gewünschten Daten zu finden. Stattdessen beginnt die Suche beim Wurzelknoten.

Die Slots im Wurzelknoten enthalten Zeiger auf Kindknoten, und die Storage-Engine folgt diesen Zeigern. Der richtige Zeiger wird durch Vergleich der Werte in den Knoten-Seiten (engl. node pages) ermittelt, die die oberen und unteren Grenzen der Werte in den Kindknoten definieren. Letztlich stellt die Storage-Engine fest, ob der gewünschte Wert existiert, oder sie erreicht erfolgreich eine Blattseite (engl. leaf page).

Blattseiten sind besonders, da sie Zeiger auf die indexierten Daten enthalten, anstatt auf andere Seiten zu verweisen. Zwischen dem Wurzelknoten und den Blattseiten können viele Ebenen von Knoten-Seiten existieren. Die Tiefe des Baumes hängt von der Größe der Tabelle ab.

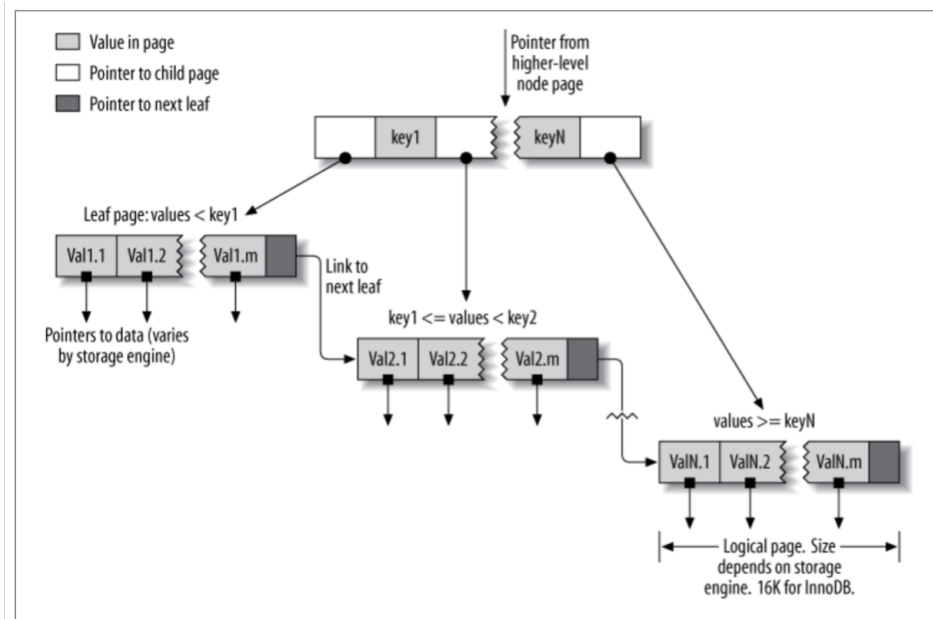


Abbildung 3.1: Darstellung des binären Baums mit Knoten und Blättern

Außerdem speichern B-Bäume die indexierten Spalten in einer festgelegten Reihenfolge, was sie besonders nützlich für die Suche nach Datenbereichen macht. Beispielsweise kann ein Index auf einem Textfeld (z.B. vom Typ VARCHAR) effizient alle Namen finden, die mit „K“ beginnen, da die Werte in alphabetischer Reihenfolge gespeichert sind.

Der Index sortiert die Werte entsprechend der Reihenfolge der in der CREATE TABLE-Anweisung angegebenen Spalten, beispielsweise des Primärschlüssels (`last_name`, `first_name`, `b_day`). B-Baum-Indizes eignen sich gut für Suchen mit dem vollständigen Schlüsselwert (engl. full key value), einem Schlüsselbereich (engl. key range) oder einem Schlüsselpräfix (engl. full key prefix). Beim Schlüsselpräfix ist dies jedoch nur der Fall, wenn die Suche das linkeste Präfix des Indexes verwendet.

Als nächstes betrachten wir die möglichen Abfragen, bei denen B-Baum-Indizes besonders hilfreich sind, um ein besseres Verständnis für ihre optimale Nutzung zu erlangen. Eine Übereinstimmung mit dem vollständigen Schlüsselwert liefert Werte für alle Spalten im Index. Eine beispielhafte Abfrage wäre die Suche nach allen Einträgen für Max Mustermann, geboren am 2000-01-01, wenn der Schlüssel aus Nachname, Vorname und Geburtsdatum besteht. Für diesen Index sind auch Abfragen nützlich, die nur mit dem linken Präfix übereinstimmen, beispielsweise die Suche nach „Mustermann“. Eine weitere Möglichkeit ist die Übereinstimmung mit einem Spaltenpräfix, also dem ersten Teil eines Spaltenwerts, etwa alle Nachnamen, die mit „M“ beginnen. Ebenso effizient ist der Index bei der Übereinstimmung mit einem Wertebereich, z.B. Nachnamen zwischen „Mustermann“ und „Müller“.

Ein B-Baum-Index kann auch genutzt werden, um Abfragen effizient zu unterstützen, bei denen eine Spalte exakt und eine andere innerhalb eines Wertebereichs abgefragt wird. Beispielsweise könnte dies eine exakte Übereinstimmung mit dem Nachnamen „Mustermann“ und eine Bereichsabfrage für Vornamen, die mit „Ma“ beginnen, umfassen. Der letzte Anwendungsfall sind Abfragen, die nur den Index verwenden und nicht die gespeicherten Zeilen, etwa wenn alle benötigten Daten im Index enthalten sind.

Ein weiterer Vorteil von B-Baum-Indizes ist, dass sie aufgrund der sortierten Baumstruktur nicht nur Abfragen, sondern auch ORDER BY-Bedingungen effizient unterstützen können. Wenn ein B-Baum für die Suche genutzt werden kann, kann er auch für die Sortierung der Ergebnisse verwendet werden.

Es gibt jedoch Einschränkungen von B-Baum-Indizes, die dazu führen, dass andere Indextypen für bestimmte Szenarien besser geeignet sind. Eine Einschränkung ist, dass die Suche nicht am linken Ende des Indexes beginnen kann. Beispielsweise ist ein Index, der aus Nachname, Vorname und Geburtsdatum besteht, nicht geeignet, um alle Personen zu finden, die vor dem Jahr 2000 geboren wurden, ohne dass der Nachname und Vorname ebenfalls spezifiziert werden.

Für optimale Leistung sollten Indizes mit den gleichen Spalten, jedoch in unterschiedlicher Reihenfolge erstellt werden, um die häufigsten Abfragen zu optimieren. Eine Analyse der am häufigsten verwendeten Abfragen kann dabei helfen zu entscheiden, ob zusätzliche Indizes erforderlich sind.

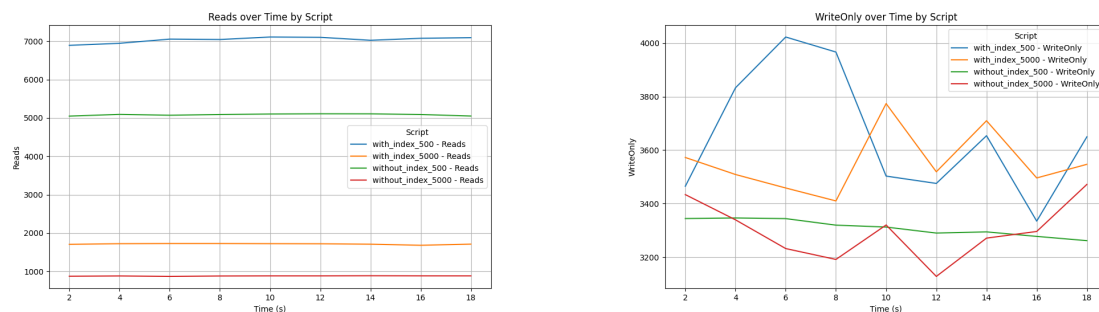


Abbildung 3.2: Grafik zeigt die verschiedenen Zeiten (in ms) für Readsabfragen (links) und Schreibbefehle (rechts) mit 500 bzw. 5000 Zeilen

3.3 Hash - Index

Ein weiterer Indextyp, den wir betrachten, ist der Hash-Index. Dieser basiert auf einer Hash-Tabelle und ist daher nur für exakte Suchanfragen geeignet, die alle Spalten im Index

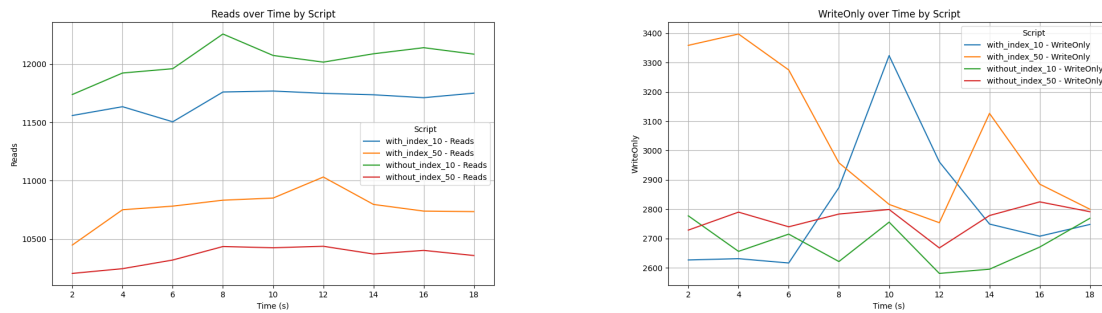


Abbildung 3.3: Grafik zeigt die verschiedenen Zeiten (in ms) für Readsabfragen (links) und Schreibbefehle (rechts) mit 10 bzw. 100 Zeilen dar

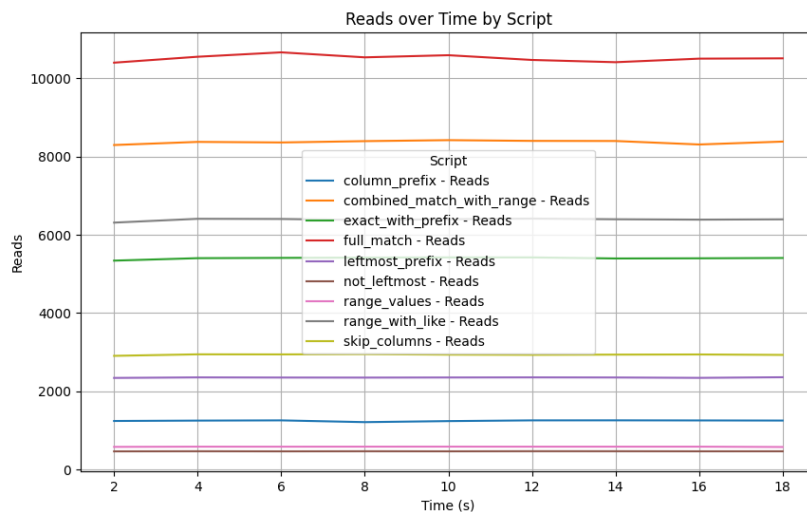


Abbildung 3.4: Grafik visualisiert die Unterschiede von verschiedenen Select - Queries auf dieselben Daten. Je nach Geschwindigkeit greift der Index besser oder nicht

verwenden. Die Funktionsweise der Storage-Engine lässt sich wie folgt beschreiben: Für jede Zeile wird mithilfe einer Hash-Funktion ein Hash-Wert der indexierten Spalte berechnet. Der Hash-Wert (engl. *hash code*) ist eine kleine Zahl, die sich in der Regel von den Hash-Werten anderer Zeilen mit unterschiedlichen Schlüsselwerten unterscheidet.

In MySQL unterstützt nur die Memory-Storage-Engine explizite Hash-Indizes. Der bereits besprochene Standard-Indextyp für Memory-Tabellen, der B-Baum-Index, ist jedoch ebenfalls möglich. Außerdem unterstützt die Memory-Engine keine eindeutigen Hash-Indizes. Das bedeutet, wenn mehrere Werte denselben Hash-Wert besitzen, speichert der Index die Zeiger auf die Zeilen (engl. *row pointers*) in demselben Hash-Tabelleneintrag, typischerweise mithilfe

einer verketteten Liste (z.B. einer *Linked List*). Im Gegensatz dazu stellen eindeutige Hash-Indizes sicher, dass für jeden Hash-Wert nur ein einziger Eintrag existiert. Bei Konflikten wird ein Mechanismus wie die *Open Addressing*-Strategie (z.B. *Linear Probing* oder *Quadratic Probing*) eingesetzt, um Konflikte zu lösen und den Speicherplatz effizient zu verwalten. Hierbei wird versucht, Konflikte direkt innerhalb der Hash-Tabelle zu bewältigen, anstatt auf zusätzliche Datenstrukturen wie verkettete Listen zurückzugreifen.

Um die Berechnung der Hash-Funktion genauer zu erläutern, folgt ein Beispiel:

Codeblock 3.2: Variationen

```
1 SELECT lname FROM testhash WHERE fname = 'Peter';
```

Zunächst berechnet MySQL den Hash-Wert für 'Peter' und verwendet diesen, um den entsprechenden Zeiger im Index zu finden. Angenommen, die Hash-Funktion liefert für 'Peter' den Wert **7654**. MySQL sucht nun im Index an der Position 7654 und findet einen Zeiger auf Zeile 3. Im letzten Schritt wird der Wert in Zeile 3 mit 'Peter' verglichen, um sicherzustellen, dass es sich um die richtige Zeile handelt. Da die Indizes nur kompakte Hash-Werte speichern, sind Hash-Indizes äußerst platzsparend, und Suchvorgänge erfolgen in hoher Geschwindigkeit.

Ähnlich wie der B-Baum-Index hat auch der Hash-Index einige Einschränkungen, auf die wir nun eingehen:

Da der Index nur Hash-Werte und Zeiger auf Zeilen (engl. *row pointers*) enthält, jedoch nicht die Werte selbst, kann MySQL den Index nicht verwenden, um das Einlesen der Zeilen zu vermeiden. Da der Zugriff auf die in den Speicher geladenen Zeilen jedoch sehr schnell ist, wird die Leistung dadurch nicht wesentlich beeinträchtigt.

Ein wesentlicher Nachteil von Hash-Indizes ist, dass sie nicht für Sortierungen verwendet werden können, da die Werte nicht in einer geordneten Reihenfolge gespeichert sind. Im Gegensatz dazu können B-Baum-Indizes Sortierungen unterstützen, wenn sie entsprechend erstellt und genutzt werden.

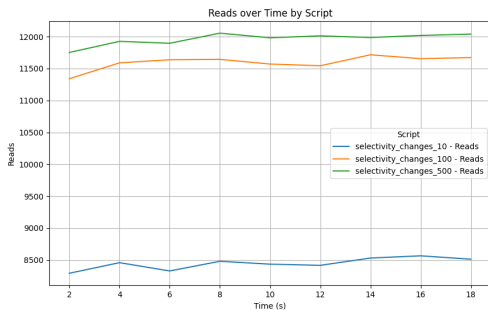
Darüber hinaus ermöglichen Hash-Indizes keine partiellen Schlüsselübereinstimmungen (engl. *partial key matching*). Da der Hash-Wert aus dem gesamten indexierten Wert berechnet wird, hilft ein Hash-Index beispielsweise nicht, wenn ein Index aus den Spalten (A, B) besteht und die WHERE-Klausel nur auf A verweist.

Ein weiterer Nachteil besteht darin, dass Hash-Indizes keine Bereichsabfragen (engl. *range queries*) unterstützen. Sie eignen sich lediglich für Gleichheitsvergleiche, wie die Operatoren = (gleich), <=> (null-sicher gleich) und IN().

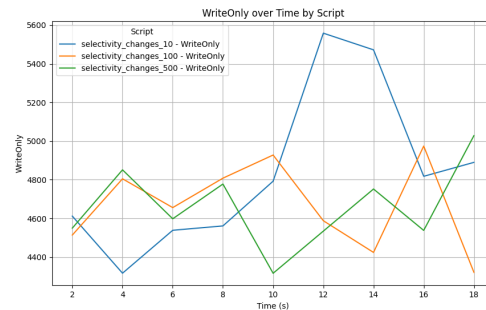
Obwohl Hash-Indizes sehr performant sind, können Hash-Kollisionen ihre Leistung beeinträchtigen. Wenn viele Werte denselben Hash-Wert aufweisen, muss die Storage-Engine jeden Zeiger in der verketteten Liste durchlaufen und die entsprechenden Werte mit dem Suchwert vergleichen, um die richtige(n) Zeile(n) zu finden.

Auch Index-Wartungsoperationen können bei vielen Kollisionen langsamer werden. Wenn beispielsweise ein Index auf einer Spalte mit sehr geringer Selektivität erstellt wird und eine Zeile gelöscht werden soll, kann das Finden des entsprechenden Zeigers im Index sehr aufwendig sein, was auch das Löschen der Zeile verzögert.

Einige Speicher-Engines, wie beispielsweise InnoDB, können zudem erkennen, wenn bestimmte Index-Werte besonders häufig verwendet werden, und automatisch einen Hash-Index für diese Werte im Speicher (engl. *memory*) erstellen, der zusätzlich zu den vorhandenen B-Baum-Indizes genutzt wird.



(a) Unterschiede von Readsabfragen



(b) Unterschiede von Schreibbefehlen

Abbildung 3.5: Vergleich der Auswirkungen von Hashkollisionen auf Lese- und Schreibvorgänge in Abhängigkeit von der Anzahl an Hashkollisionen.

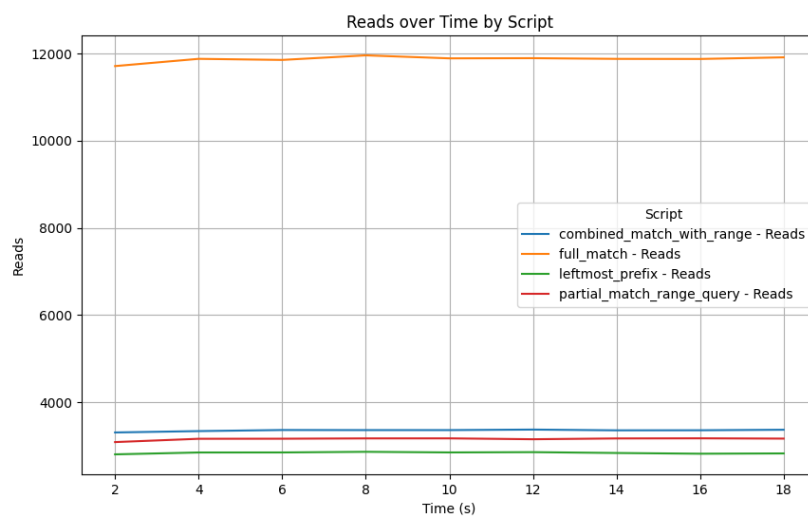


Abbildung 3.6: Grafik visualisiert die Unterschiede von verschiedenen Select - Queries auf dieselben Daten. Je nach Geschwindigkeit greift der Index besser oder nicht

Literatur

- akopytov. (2024). *Sysbench Github Repository*. Verfügbar 28. Oktober 2024 unter <https://github.com/akopytov/sysbench>
- Difallah, D. E., Pavlo, A., Curino, C., & Cudré-Mauroux, P. (2013). OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB*, 7(4), 277–288. <http://www.vldb.org/pvldb/vol7/p277-difallah.pdf>
- Germany, R. (2024). *Was ist ein Snapshot Backup?* Verfügbar 28. Oktober 2024 unter <https://www.rubrik.com/de/insights/what-is-a-snapshot-backup>
- Reimers, N. (2017). *Virtuelle, dezidierte und Cloud-Server: MySQL-Benchmark mittels sysbench*. Verfügbar 28. Oktober 2024 unter <https://www.webhosterwissen.de/know-how/server/mysql-benchmark-mittels-sysbench/>
- Shopify. (2022a). *Detailed design documentation*. Verfügbar 28. Oktober 2024 unter <https://shopify.github.io/mybench/detailed-design-doc.html#live-monitoring-user-interface>
- Shopify. (2022b). *What is mybench?* Verfügbar 28. Oktober 2024 unter <https://shopify.github.io/mybench/introduction.html>
- Shopify. (2024). *Mybench Github Repository*. Verfügbar 28. Oktober 2024 unter <https://github.com/Shopify/mybench>
- Vogel, M. (2009). *EDV-Lexikon: Bottleneck*. Verfügbar 28. Oktober 2024 unter <https://martinvogel.de/lexikon/bottleneck.html>
- Williams, T., Kelley, C., & many others. (2024). *Gnuplot Repository*. Verfügbar 28. Oktober 2024 unter <https://github.com/gnuplot/gnuplot>

Anhang

Hier beginnt der Anhang. Siehe die Anmerkungen zur Sinnhaftigkeit eines Anhangs in Abschnitt ?? auf Seite ??.

Der Anhang kann wie das eigentliche Dokument in Kapitel und Abschnitte unterteilt werden. Der Befehl `\appendix` sorgt im Wesentlichen nur für eine andere Nummerierung.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

Performance - Optimierung von Datenbanken

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, 21. Dezember 1940