

# Performantievergelijking van database-systemen

## Casus onderzoeksproces

Pieter-Jan Rotsaert<sup>1</sup>, Robby Daelman<sup>2</sup>, Jeffrey Waegneer<sup>3</sup>

### Samenvatting

Dit artikel beschrijft en vergelijkt de performantieverschillen tussen twee populaire RDBMS: MySQL en MariaDB. Concreet testen we de DBMS op uitvoeringstijd van query's en op hun CPU- en geheugengebruik. We kiezen specifiek voor deze twee RDBMS omdat MySQL en MariaDB gelijkaardig opgebouwd zijn. MariaDb is immers een 'fork' van MySQL. We voeren enkele testen uit en meten de individuele performantie voor elk van de twee systemen op voorgaande criteria. De testresultaten in acht nemend kunnen we concluderen dat MySQL doorgaans sneller is dan MariaDB en dat het verschil statistisch significant is. We testen op een databank met tabellen tot 100.000 records, bijkomend onderzoek kan verricht worden op tabellen met meer records. Ook het geheugengebruik bij langdurige operatie van de systemen kan nog verder onderzocht worden.

### Sleutelwoorden

Database-beheer – Relationale databases – Performantie – MySQL – MariaDB

**Contact:** <sup>1</sup> pieterjan.rotsaert.y7988@student.hogent.be; <sup>2</sup> jeffrey.waegneer.y7989@student.hogent.be;

<sup>3</sup> robby.daelman.w1593@student.hogent.be

**GitHub-Repository:** <https://github.com/HoGentTIN/ozt-dbperf-a11>

## Inhoudsopgave

<b>1</b>	<b>Inleiding</b>	<b>2</b>
1.1	Voorwoord	2
1.2	Literatuurstudie	2
<b>2</b>	<b>Methodologie</b>	<b>2</b>
2.1	De testomgeving	3
2.2	Meten van de uitvoeringstijd	3
2.3	Meten van het cpu- en geheugengebruik	3
2.4	De uitgevoerde queries	4
<b>3</b>	<b>Experimenten</b>	<b>5</b>
3.1	Testresultaten	5
3.2	Statistische significantie	7
<b>4</b>	<b>Conclusie</b>	<b>8</b>
	<b>Referenties</b>	<b>8</b>

## 1. Inleiding

### 1.1 Voorwoord

In dit onderzoek zullen we de focus leggen op de verschillen in performantie tussen MariaDB en MySQL, twee populaire RDBMS (Relational Database-Management System(s).) We vergelijken onze resultaten thans met die van een eerder onderzoek, Bassil (2012), om zodoende na te gaan of die laatste nog steeds relevant zijn. Om de resultaten zo consistent mogelijk te houden nemen we een deel van de queries over uit Bassil (2012) evenals de structuur van de databank die gebruikt werd. In ons onderzoek wordt zowel als bij Bassil (2012) getest op uitvoeringstijd, geheugengebruik en procentueel cpu-gebruik.

We hebben de keuze gemaakt om MariaDB te vergelijken met MySQL omwille van hun gelijkaardige structuur en werking. MariaDB is een fork van MySQL dat ernaar streeft om volledig API- en protocol-compatibel te blijven met MySQL, met als gevolg dat de enige verschillen tussen de twee zich achter de schermen bevinden, datzijnde de interne werking en opbouw.

De twee kunnen met andere woorden als drop-in replacement gebruikt worden voor elkaar, vandaar dat het nuttig is om te weten of er eventueel sprake is van performantie-verschillen, aangezien het net zo makkelijk is om de switch te maken.

### 1.2 Literatuurstudie

Vooraleer we aan ons onderzoek beginnen bekijken we eerst enkele andere bezienswaardige artikels. In het onderzoek van Liang en Lu (2010) worden 4 database management systems getest: SQLite, PostgreSQL, Firebird en MySQL. Men focust zich hier vooral op gebruikerservaring, namelijk de kostprijs, marktpositie, gebruiksvriendelijkheid enzovoort. Qua performantie is deze studie minder interessant omdat men enkel de uitvoeringstijd meet en niet de andere factoren zoals geheugengebruik en dergelijke meer. Verder werd er niet veel achtergrond informatie gegeven over de geschiedenis en ontstaan van de systemen.

Vervolgens nemen we de studie van Abubakar (2014) onder de loep. Opnieuw wordt hier enkel de uitvoeringstijd gemeten. Tot nu toe blijkt de studie van Bassil (2012) de meest uitgebreide te zijn, aangezien daar rekening gehouden wordt met drie andere belangrijke factoren: geheugengebruik, procentueel cpu-gebruik en het aantal threads die in beslag genomen worden. Abubakar (2014) vermeldt echter wel de geschiedenis van de betreffende RDBMS alsook een korte beschrijving van de verschillende ontwikkelaarsfilosofieën.

Nog een benoemenswaardig artikel is dat van Weinberger (2015). Hierin worden bovenop RDBMS ook nog zogenaamde 'Document-Stores' zoals MongoDB vermeldt. Deze werken op een geheel andere manier dan relationele systemen. Gerelateerde data wordt namelijk hiërarchisch opgeslaan in plaats van verspreid over verschillende tabellen en verschillende records. Bij deze systemen is het dus mogelijk om met één enkele query zeer veel gegevens op te halen, bijvoorbeeld een gebruiker met naam, adres, enz. maar ook een lijst van aankopen, details van die aankopen enzoverder. Als men deze gegevens wil opvragen in een RDBMS dient men al meerdere JOINS te gebruiken of meerdere queries te maken. Dit type databanken is vooral handig wanneer men zeer vaak alle gegevens nodig heeft tegelijk, het is bijvoorbeeld één van de meest populaire soorten databanken bij online games voor het opslaan van gebruikersprofielen en data.

Hoewel we de performantieverschillen voor simpele select-queries zouden kunnen vergelijken tussen bijvoorbeeld MySQL en MongoDB hebben we besloten dit niet te doen aangezien de resultaten niet representatief zouden zijn tegenover het gebruik van deze systemen in de praktijk.

Voorgaande artikels geven ons een duidelijke rapportering van hun respectievelijke resultaten. Er zijn duidelijke grafieken aanwezig die aantonen welke DBMS beter presteren in verschillende situaties. Telkens werd er een duidelijke en logische conclusie getrokken door de auteur betreffende de performantie van de besproken DBMS.

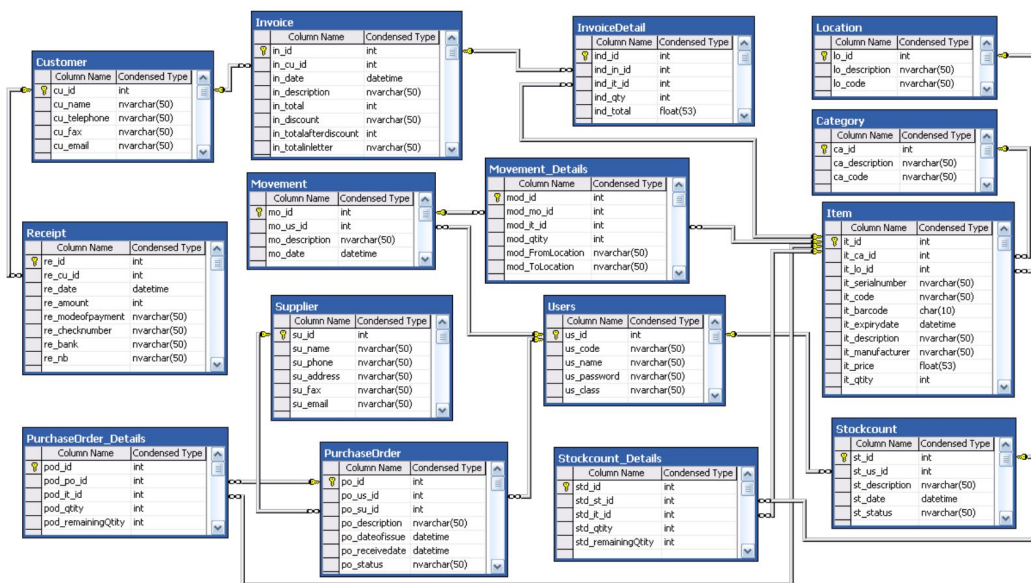
Zoals eerder aangehaald is het begrip performantie niet eenduidig. Wanneer men spreekt over performantie bij een RDBMS wordt vooral rekening gehouden met CPU-gebruik, uitvoeringstijd, geheugengebruik en eventueel het aantal threads. Deze verschillende factoren hebben geen rechtstreeks verband met elkaar, een RDBMS kan performant zijn op vlak van uitvoeringstijd, maar bijvoorbeeld zeer hongerig op vlak van geheugengebruik, zoals wordt aangehaald over het IBM-DB2 systeem in Abubakar (2014).

Wij richten, zoals eerder vermeld, ons onderzoek op MariaDB en MySQL. Hiervoor gebruiken we enkele queries evenals de databankstructuur uit Bassil (2012). We verzamelen gegevens over de uitvoeringstijd, het cpu-gebruik en het geheugengebruik.

De rest van dit artikel is als volgt gestructureerd: Sectie 2 beschrijft de gevolgde methodologie, Sectie 3 beschrijft de resultaten van onze experimenten, Sectie 4 beschrijft de conclusie.

## 2. Methodologie

De databank die we gaan gebruiken is een databank voor een winkel die producten verkoopt en verzendt. Dit is dezelfde databank die bij het onderzoek van Bassil (2012) gebruikt werd in zijn onderzoek. Het databank schema kan u hieronder terug vinden:



Figuur 1. Databank-schema Bassil (2012)

## 2.1 De testomgeving

We genereren willekeurige data met een kleine zelf geschreven applicatie ontwikkeld in C++. Deze applicatie maakt deel uit van een PowerShell script dat automatisch een databank (figuur 1) (her-)aanmaakt en deze dan opvult met de random gegenereerde data.

Er kan worden ingegeven hoeveel records er gegenereerd dienen te worden en vervolgens worden deze op automatische en willekeurige manier gegenereerd. Deze gegenereerde data wordt weggeschreven in .csv bestanden die vervolgens door het script ingeladen worden in de databank. De applicatie die de data genereerd evenals het PS script kan teruggevonden worden op onze github. Voor het uitvoeren van de MariaDB en MySQL servers gebruiken we twee gelijk opgestelde Linux virtuele machines in VirtualBox.

De specificaties van deze virtuele machines zijn als volgt:

- Besturingssysteem: CentOS Linux Release 7.4.1708 (Core)
- CPU architectuur: AMD64 (64-bits)
- Aantal CPU's (Cores): 4
- Geheugen: 4GB DDR4 @1329MHz

De specificaties van de gastmachine waarop de virtuele machines draaien zijn als volgt:

- Merk en type: Dell Alienware Aurora
- Besturingssysteem: Windows 10 Home
- CPU architectuur: AMD64 (64-bits)
- Processor: Intel Core i7-8700 CPU @ 3.20GHz
- Geheugen: 16GB DDR4 @1329MHz

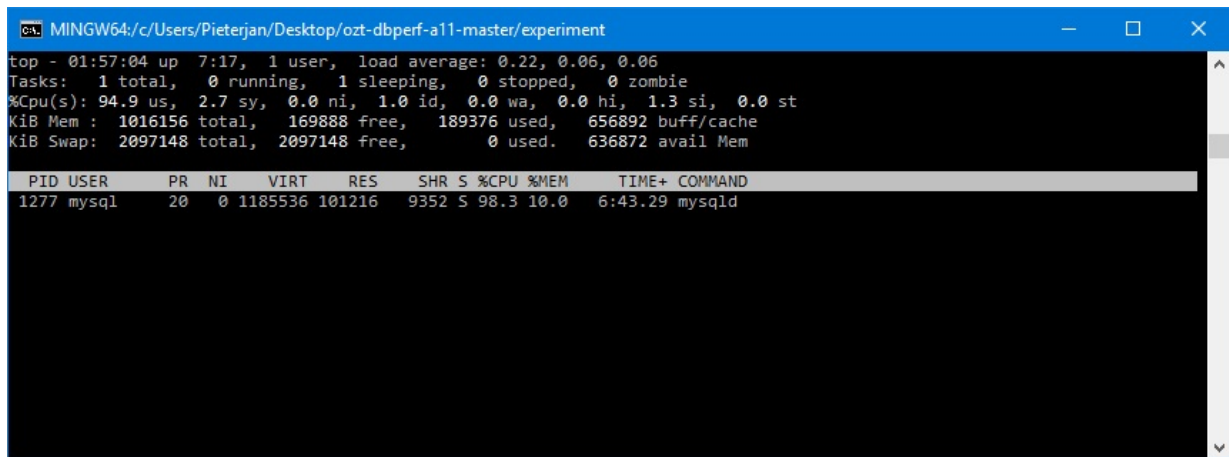
## 2.2 Meten van de uitvoeringstijd

Voor het meten van de uitvoeringstijd gebruiken we een kleine zelfgeschreven Java jdbc console-applicatie. De broncode voor deze applicatie is net zoals de scripts beschikbaar in de github repository.

De applicatie voert 500 maal de 9 queries van Bassil (2012) uit op een nieuwgegenereerde databank. Dit gebeurt drie keer, eerst op een dataset met 1.000 records per tabel, vervolgens een met 10.000 records per tabel en tot slot nog een met 100.000 records per tabel.

## 2.3 Meten van het cpu- en geheugengebruik

Voor het meten van het cpu- en geheugengebruik gebruiken we het 'top'-commando (figuur 2) van Linux. Dit commando geeft ons in realtime het percentage van de cpu dat in gebruik is weer, alsook hoeveel geheugen het serverproces in beslag neemt. Het enige nadeel hierbij is dat het update-interval van deze statistieken ongeveer een second bedraagt. Dit maakt het moeilijk om het precieze cpu-gebruik te meten bij lage uitvoeringstijden. Vandaar dat bij de testen met de laagste record-aantallen het cpu-gebruik vaak gelijkloopt voor de verschillende queries.



```
top - 01:57:04 up 7:17, 1 user, load average: 0.22, 0.06, 0.06
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 94.9 us, 2.7 sy, 0.0 ni, 1.0 id, 0.0 wa, 0.0 hi, 1.3 si, 0.0 st
KiB Mem : 1016156 total, 169888 free, 189376 used, 656892 buff/cache
KiB Swap: 2097148 total, 2097148 free, 0 used. 636872 avail Mem

  PID USER      PR  NI   VIRT    RES    SHR S  %CPU  %MEM    TIME+   COMMAND
 1277 mysql     20   0 1185536 101216  9352 S   98.3   10.0   6:43.29   mysqld
```

**Figuur 2.** Het 'top'-commando op onze Linux VM

## 2.4 De uitgevoerde queries

We nemen 8 van de 11 queries over uit het onderzoek van Bassil (2012), alsook een lege query (query 6). De lege query laat ons toe om de overhead te meten bij het sturen van een query naar de DB server en de ontvangst van een antwoord. De queries zien er uit als volgt:

### Query 1

```
SELECT * FROM dbo.customer
```

### Query 2

```
SELECT * FROM dbo.invoice
WHERE invoice.in_id > 50 AND invoice.in_date > '1-1-2006'
AND invoice.in_date < '1-1-2007'
AND invoice.in_description LIKE '%re%'
AND (invoice.in_total < 100 OR NOT invoice.in_cu_id >= 5 )
AND (invoice.in_id BETWEEN 1 AND 10000 OR invoice.in_id > 49+1)
AND invoice.in_total+33 < 5
```

### Query 3

```
SELECT customer.cu_id, customer.cu_name, customer.cu_telephone,
customer.cu_fax, customer.cu_email
FROM customer
ORDER BY customer.cu_id, customer.cu_name DESC, customer.cu_telephone DESC,
customer.cu_fax, customer.cu_email DESC;
```

### Query 4

```
SELECT SUM(invoice.in_total),
AVG(invoice.in_totalafterdiscount),
MAX(invoice.in_total),
COUNT(customer.cu_id),
SUM(invoicedetail.ind_qty)
FROM customer, invoice, invoicedetail
WHERE customer.cu_id = invoice.in_cu_id AND
invoice.in_id = invoicedetail.ind_in_id
GROUP BY invoice.in_id ;
```

**Query 5**

```
SELECT SUM(dbo.invoice.in_total)
FROM dbo.invoice
JOIN dbo.customer ON customer.cu_id = invoice.in_cu_id
GROUP BY dbo.customer.cu_id
HAVING COUNT(invoice.in_id) > 0 AND
SUM(invoice.in_total) < AVG(invoice.in_totalafterdiscount)
ORDER BY SUM(dbo.invoice.in_total);
```

**Query 6**

Lege query.

**Query 7**

```
SELECT *
FROM dbo.category, dbo.item
WHERE dbo.item.it_ca_id = dbo.category.ca_id;
```

**Query 8**

```
UPDATE item
SET item.it_price = (item.it_price * 0.1),
    item.it_qty = 10,
    item.it_description = 'TV'
WHERE item.it_id > 10
AND item.it_expirydate > '2007/1/1'
AND item.it_expirydate < '2008/1/1';
```

**Query 9**

```
DELETE FROM dbo.invoicedetail
WHERE dbo.invoicedetail.ind_id
IN (SELECT in_id FROM dbo.invoice where invoice.in_description LIKE '%t%' )
AND dbo.invoicedetail.ind_id
IN (SELECT in_id FROM dbo.invoice where invoice.in_date > '2006/1/1' )
AND dbo.invoicedetail.ind_id
IN (SELECT in_id FROM dbo.invoice where invoice.in_date < '2007/1/1' );
```

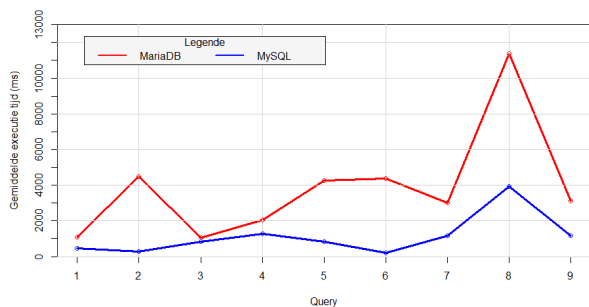
## 3. Experimenten

### 3.1 Testresultaten

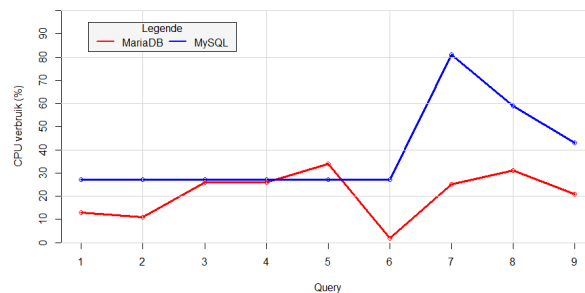
De databank die we gebruiken om te testen (zie figuur 1) bevat 15 tabellen. De tabellen worden elk opgevuld met 1.000 records, vervolgens met 10.000 records en tot slot met 100.000 records. Alle queries worden 500x uitgevoerd. Het totaal aantal milliseconden (ms) wordt weergegeven per query in de onderstaande grafieken voor zowel MariaDB als MySQL. Het gemiddelde CPU-gebruik tijdens de uitvoering van de respectievelijke queries wordt in de grafieken aan de rechterkant weergegeven. We bekeken ook het geheugen verbruik maar deze bleef onveranderd en heeft dus geen impact bij op de testresultaten.

### 1.000 records per tabel

Testresultaten bij het 500x uitvoeren van de 9 queries op een databank met 1.000 records per tabel:



(a) Uitvoeringstijd bij 1.000 records per tabel

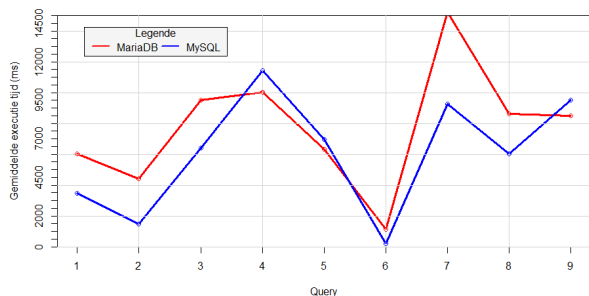


(b) CPU-gebruik bij 1.000 records per tabel

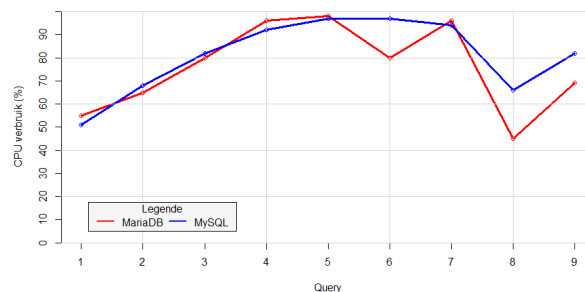
We zien zeer duidelijk dat bij 1000 records MySQL het snelst is maar dan ook wel het meeste cpu verbruikt. MySQL is bij de meeste queries meer dan dubbel zo snel in vergelijking met MariaDb. Dit is vreemd want als we kijken naar het cpu verbruik dan verbruikt MySQL inderdaad meer cpu dan MariaDb, maar dit verschil is niet groot in vergelijking met het tijds verschil. Bij de eerste vijf queries gebruiken beide ongeveer evenveel cpu maar vanaf query 6 verbruikt begint MySQL meer cpu te verbruiken, de tijdswinst is dan ook tijdens deze queries het grootst. Het cpu verschil is het grootste bij query 7, MySQL verbruikt hier namelijk 4 keer meer cpu als MariaDb terwijl MySQL slechts 2 keer sneller was. Bij deze test resultaten kunnen we stellen dat MySQL cpu opoffert ten gunste van uitvoeringstijd, terwijl MariaDb het iets trager deed maar dan ook wel met minder resources.

### 10.000 records per tabel

Testresultaten bij het 500x uitvoeren van de 9 queries op een databank met 10.000 records per tabel:



(a) Uitvoeringstijd bij 10.000 records per tabel

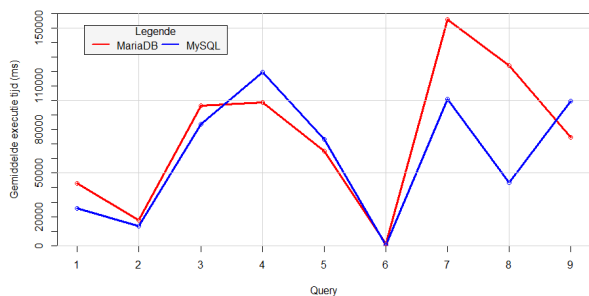


(b) CPU-gebruik bij 10.000 records per tabel

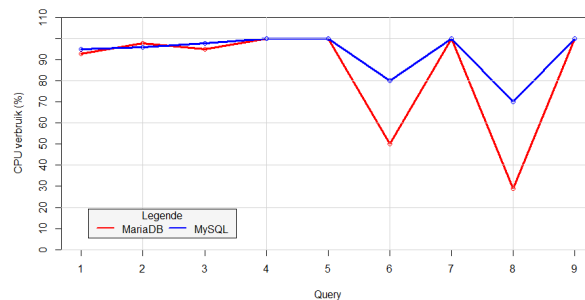
Bij de testresultaten van 10000 records per tabel liggen de resultaten allemaal iets dicht bij elkaar. Maar toch stellen we nog steeds vast dat MySQL bij de meeste queries sneller was dan MariaDb en dat MySQL ook hier weer meer cpu gebruikte. De uitvoeringstijden liggen dicht bij elkaar dan bij de vorige proef. Toch is MySQL duidelijk sneller ookal was MariaDb sneller bij Query 4 en 5, de resultaten liggen bij deze queries heel erg dicht tegen elkaar terwijl dat bij de andere queries niet altijd het geval is. Het Cpu verbruik ligt heel dicht tegen elkaar uitgezonderd van query 8 en 9. Bij deze queries verbruikte MySQL weer meer dan MariaDb maar de resultaten liggen nooit verder dan 10 procent uit elkaar. We vermoeden dat het verschil in cpu verbruik bij query 6 te wijten is aan het update interval van het top commando. Doordat Query 6 een lege query is wordt deze heel erg snel uitgevoerd en heeft het top commando geen tijd om eht cpu-gebruik up te daten.

### 100.000 records per tabel

Testresultaten bij het 500x uitvoeren van de 9 queries op een databank met 100.000 records per tabel:



(a) Uitvoeringstijd bij 100.000 records per tabel



(b) CPU-gebruik bij 100.000 records per tabel

De resultaten van 100000 records wijzen opnieuw uit dat MySQL sneller is dan MariaDb. Het cpu verbruik ligt bij deze testresultaten nog dicht bij elkaar. Het tijdsverschil ligt slechts bij 2 queries ver uit elkaar, dit zijn namelijk query 7 en 8. Een interessante opmerking is dat ondanks het tijdsverschil bij query 7 beide systemen bijna het maximum cpu verbruik gebruiken. Dit wil zeggen dat mysql sneller was met bijna exact hetzelfde cpu verbruik. Bij query 8 was MySQL ook aanzienlijk veel sneller maar in dit geval gebruikte MariaDb ook maar half zoveel cpu. Ook bij deze test resultaten vermoeden we dat het cpu verschil bij query zes ligt aan de onnauwkeurigheid van het top commando bij een lege query.

### 3.2 Statistische significantie

Voor we aan ons onderzoek begonnen zijn hebben we eerst even nagedacht over wat we verwachtte van de resultaten, na een uitgebreid overleg kwamen we tot de constatacie dat het verschil in uitvoeringstijd niet statisch significant zou zijn omdat MariaDb een fork is van MySQL en dat beide dbms dus erg op elkaar gelijken.

Nu we de resultaten hebben kunnen we nagaan of onze hypothese ook echt klopt. Is het verschil in uitvoeringstijden statisch significant? Om deze vraag te beantwoorden gaan we gebruik maken van de z toets. We nemen eerst het gemiddelde van alle standaard afwijkingen en de gemiddelde uitvoeringstijden van beide dbms om te gebruiken in onze formules. We nemen een significantie niveau van 5 procent.

Stel de nulhypothese gelijk aan: 'De gemiddelde tijd van MySQL is sneller dan die van MariaDb.' en de alternatievehypothese gelijk aan: 'De gemiddelde uitvoeringstijd van mysql is ongeveer dezelfde als die van MariaDb.' Om de null hypothese te verwerpen moeten we berekenen of de gemiddelde uitvoeringstijd van MariaDb tussen de toegelaten grenswaarden ligt. De formule om die grenswaarden te berekenen is:

$$g = \mu + z \frac{\sigma}{\sqrt{n}}$$

De grenswaarde die we bekomen is 124 ms dat wil dus zeggen als de gemiddelde uitvoeringstijd van MariaDb lager ligt als 124 ms we de nulhypothese zullen kunnen verwerpen en de alternatieve hypothese aantonen. De gemiddelde uitvoeringstijd van MariaDb is echter 145 ms. Dit wil dus zeggen dat de uitvoeringstijd van MariaDb niet binnen de grenswaarden ligt en we de nulhypothese dus niet kunnen verwerpen. Het verschil in uitvoeringstijd is dus statistisch significant.



## 4. Conclusie

In sectie 3.2 hebben we aangetoond dat onze alternatieve hypothese incorrect was. We dachten dat het verschil tussen de uitvoeringstijden van de queries tussen beide dbms niet statisch significant zouden zijn. We hebben deze hypothese moeten verwerpen en concludeerden dat MySQL sneller is dan MariaDb. We vermoeden dat een reden hiervoor zou zijn dat MySQL sneller alle resources gebruikt. In de testresultaten van 1000 records is duidelijk te zien dat MariaDb slechts alle resources gaat gebruiken wanneer ze dit nodig vinden. MySQL daarentegen gebruikt bijna meteen de volle capaciteit. Er zijn veel richtingen waar dit onderzoek naartoe zou kunnen gaan. Een eerste optie zou zijn om meer resources te gebruiken om te bekijken hoe veel meer resources MySQL zou willen gebruiken, indien het hiervoor de mogelijkheid zou hebben. Een tweede optie zou zijn om onderzoek te gaan doen naar de reden waarom MySQL nu net sneller is. Dit zou echter onderzoek vragen naar de sourcecode van beide projecten. Als laatste zou het mogelijk zijn om met nog meer records te gaan testen. Iets dat we merkten bij de testresultaten is dat hoe meer records er gebruikt werden hoe dichter de test resultaten bij elkaar kwamen te liggen, MariaDb begon zelf hier en daar sneller te worden dan MySQL. Misschien is MariaDb wel beter in queries op grotere databanken in vergelijking met MySQL.

## Referenties

- Abubakar, Y. (2014). BENCHMARKING POPULAR OPEN SOURCE RDBMS: A PERFORMANCE EVALUATION FOR IT PROFESSIONALS. 3, 39. Verkregen 2 mei 2018, van [https://www.researchgate.net/publication/316132763\\_BENCHMARKING\\_POPULAR\\_OPEN\\_SOURCE\\_RDBMS\\_A\\_PERFORMANCE\\_EVALUATION\\_FOR\\_IT\\_PROFESSIONALS](https://www.researchgate.net/publication/316132763_BENCHMARKING_POPULAR_OPEN_SOURCE_RDBMS_A_PERFORMANCE_EVALUATION_FOR_IT_PROFESSIONALS)
- Bassil, Y. (2012). A Comparative Study on the Performance of the Top DBMS Systems. *Journal of Computer Science and Research*, 1(1), 20–31.
- Liang, X. & Lu, Y. (2010). Evaluation of database management systems, 37. Verkregen van <http://www.diva-portal.org/smash/get/diva2:367006/fulltext01.pdf>
- Weinberger, C. (2015, oktober 13). Benchmark: PostgreSQL, MongoDB, Neo4j, OrientDB and ArangoDB. Verkregen van <https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/>