FOM München

Fachbereich Wirtschaftsinformatik

Seminararbeit

Optimierung von MySQL Anfragen unter Zuhilfenahme von Explain

Eingereicht von:

Oliver Kurmis

Perfallstraße 8, 81675 München

Email: oliver@kurmis.com

Matrikel-Nr: 328091

Abgegeben am:

1. Juli 2014

Erarbeitet im:

3. Semester

Inhaltsverzeichnis

Abkürzungsverzeichnis

CPU Central Processing Unit (deutsch: Hauptprozessor)

DB Datenbank

HDD Hard Disk Drive (deutsch: Festplattenlaufwerk)

QEP Query Execution Plan (deutsch: Anfrage-Ausführungsplan)

RAM Random Access Memory (deutsch: Hauptspeicher oder Arbeitsspeicher)

RDBMS Relationales Datenbank-Managementsystem

SQL Structured Query Language, standardisierte Datenbank-Abfragesprache

SSD Solid State Drive (deutsch: Halbleiterlaufwerk, i.d.R. mit Flash-Speicher)

1 Einleitung

1.1 Problemstellung

Datenbank-Systeme finden heute in nahezu allen IT-Systemen Verwendung. Der Optimierung von Datenbank-Anfragen kommt daher eine große Bedeutung zu. Hierfür gibt es eine Vielzahl von Möglichkeiten, z.B. Latenz und Bandbreite der Anbindung der Datenbank, Leistungsfähigkeit des Datenbank-Servers, Anzahl der Datenbank-Anfragen im Programmcode, Cachingmeachanismen.

Hat man andere Flaschenhälse ausgeschlossen oder bereits optimiert, gilt es die für die Performance relevanten SQL-Abfragen des Systems zu identifizieren und gezielt zu optimieren. Dies können lange laufende Abfragen sein, die z.B. bei MySQL mit der Logdatei für langsame Anfragen gezielt ermittelt werden können. Oftmals sind es aber auch viele einfache, kurze Abfragen, die jedoch zu Hunderten oder Tausenden pro Sekunde auftreten und so die Anwendung viel Zeit kosten und den Datenbank-Server belasten.

Viele RDBMS stellen mit dem SQL-Kommando EXPLAIN eine Möglichkeit zur Verfügung, mehr über die innere Arbeitsweise der Datenbank bei einer bestimmten SQL-Abfrage zu erfahren. Durch gezielte Veränderung der SQL-Abfrage oder des Datenschemas kann somit die Bearbeitung der Abfrage optimiert werden.

1.2 Zielsetzung der vorliegenden wissenschaftlichen Auseinandersetzung

Die folgende Arbeit bezieht sich speziell auf die Optimierung von SQL-Anfragen mittels EXPLAIN bei dem RDBMS MySQL. Es soll untersucht werden

1.3 Vorgehensbeschreibung

-Literaturrecherche betrieben, -wesentliche Punkte zusammengefasst -an Beispieldatenbank experimentell nachvollzogen

2 Theoretische Grundlagen

2.1 Der physische Zugriff auf die Daten

Daten einer DB werden in der Regel auf einer Festplatte (HDD) oder einem Flash-Laufwerk (SSD) gespeichert. Das RDBMS nutzt dazu Funktionen des Betriebssystems auf verschiedenen Ebenen. Dateisystem-Treiber, nimmt Lese und Schreibanforderungen für Datensätze an und gibt rechnet diese in nie durchnumerierten Blöcke des Blockgerätes um. Der Blockgeräte-Treiber liest dann die entsprenden Blöcke von der Platte oder schreibt sie dort hin.

DBMS <—> Dateisystemtreiber <—> Blockgerätetreiber <—> HDD/SSD

Auf den verschiedenen Ebenen findet hierbei Caching statt, um die relativ langsamen Zugriffe auf den Massenspeicher (HDD oder SSD) zu vermeiden oder zumindest zu bündeln. Die Reduzierung von Massenspeicher-Zugriffen ist daher auch eine effektive Methode der DB-Anfrage-Optimierung.

2.2 Speicherstrukturen

Binärbaum:

B-Baum, Hashing, Heap

2.3 Bearbeitung von SQL-Statements

Umsetzung in relationale Algebra

2.4 Optimierungen: frühzeitige Restriktionen, JOINs

3 MySQL-EXPLAIN

In vielen RDBMS steht mit dem SQL-Kommando EXPLAIN ein Werkzeug zur Werfügung, um mehr darüber zu erfahren, wie die Datenbank eine bestimmte Anfrage ausführt, wie also der Query Execution Plan (QEP) ist. Das EXPLAIN-Kommando gehört jedoch nicht zum SQL-Standard und wird bei den verschiedenen RDBMS unterschiedliche Ausgaben erzeugen. Bei MySQL ist EXPLAIN sehr mächtig und gibt umfassend und datailliert Auskunft über den QEP. Hierbei muss jedoch beachtet werden, dass der QEP nicht fix ist, sondern bei jeder Anfrage vom Optimierer erneut erstellt wird (sofern die Anfrage nicht bereits aus dem Query-Cache bedient werden kann). Es gibt daher keine Garantie, dass die Anfrage immer mit dem vorher von EXPLAIN gezeigten QEP ausgeführt wird. Es empfielt sich daher, die untersuchten SQL-Anfragen von Zeit zu Zeit erneut mit EXPLAIN zu prüfen - mit Real-World-Daten.

3.1 Einfache SELECT-Anfragen mit einer Tabelle

3.2 Umschreiben von Nicht-SELECT-Anfragen

Vor MySQL 5.6.3 konnte EXPLAIN nur auf SELECT-Anfragen angewendet werden. [4] Einige Nicht-SELECT-Anfragen, wie DELETE, INSERT, REPLACE und UPDATE können jedoch in ein entsprechendes SELECT-Kommando umgeformt werden, um dieses dann mit EXPLAIN zu untersuchen. Zu beachten ist jedoch, daß schreibende Anweisungen generell aufwendiger sind als das entsprechende SELECT-Kommando, da zusätzlich zum Auffinden der Daten noch die Schreiboperation ausgeführt werden muss, ggf. kommen auch noch Aktualisierungen von Indizes hinzu. Beispiel:

```
DELETE user WHERE last_login < '2012-01-01'
```

kann zu folgendem SELECT umgeschrieben werden:

```
SELECT id FROM user WHERE last_login < '2012-01-01'
```

Ab MySQL 5.6.3 kann EXPLAIN auf SELECT, DELETE, INSERT, REPLACE und UPDATE angewendet werden.

3.3 Die Spalten der EXPLAIN-Ausgabe

Als Ergebnis einer EXPLAIN-Anfrage liefert MySQL eine Tabelle mit festen Spalten und einer oder mehrerer Zeilen, je nach Komplexität der Anfrage. Die einzelnen Spalten haben folgende Bedeutung (vgl. [5], [1] S. 665-676, [3] Pos. 2696-2906):

id
Diese Zahl identifiziert das SELECT, zu dem die Zeile gehört. Bei einer einfachen SELECT-Abfrage steht in diesem Feld demnach immer nur die Zahl 1.

select_type
Die Spalte gibt an, ob es sich um ein einfaches oder komplexes SELECT handelt. Folgende Werte können hierbei auftreten:

SIMPLE einfaches SELECT, keine Unterabfragen oder UNIONS
PRIMARY äußeres SELECT eines komplexen SELECT
SUBQUERY SELECT in einer Unterabfrage
DERIVED SELECT in einer Unterabfrage in der FROM-Klausel
UNION zweites bzw. nachfolgende SELECT einer UNION
UNION RESULT Ergebnis des UNION, wird aus temporärer Tabelle geholt

table
Die Spalte Table gibt an, auf welche Tabelle zugegriffen wird. Dies kann der tatsächliche Tabellenname sein oder der Alias. Die Spalte ist von oben nach unten zu lesen, um die Join-Reihenfolge zu sehen, die der Optimierer für die Anfrage gewählt hat. Bei komlexeren Anfragen mit abgeleiteten Tabellen und Vereinigungen können hier noch weitere Werte auftreten: <derivedN> wenn es in der FROM-Klausel eine Unterabfrage gibt, wobei N die ID der Unterabfrage ist und in den nachfolgenden Zeilen der Ausgabe zu finden ist. Bei einer Vereinigung mit UNION enthält die UNION RESULT-Zeile in der table-Spalte die IDs der Abfragen, welche vereinigt werden, und beziehen sich daher immer auf vorhergehende Zeilen der Ausgabe, z.B. <union2,4>.

type

Wie ist der Zugriffstyp, wie wird MySQL die Zeilen in der Tabelle auffinden? Folgende Werte können hierbei auftreten (in geordneter Reihenfolge vom langsamsten zum schnellsten Zugriffstyp):

ALL full Tablescan, d.h. Tabelle muss in der Regel von Anfang bis Ende durchlaufen werden, ist ein starker Indiz für weiteren Optimierungsbedarf

index wie Tablescan, aber Scannen der Tabelle erfolgt in Indexreihenfolge, eine extra Sortierung wird hierbei vermieden

range Bereichsscan, d.h. eingeschränkter Indexscan, z.B. bei BETWEEN oder > in der WHERE-Klausel

ref Indexzugriff findet statt, auch Index-Lookup genannt, Zeilen entsprechen einem Wert, nur bei einem nichteindeutigen Index, Index wird hierbei mit einem Referenzwert verglichen. Variante: ref_or_null

eq_ref Index-Lookup mit eindeutigem Treffer, bei Primärschlüssel oder eindeutigem Index

const,system der Datenzugriff konnte von MySQL wegoptimiert oder in eine Konstante umgewandelt werden.

NULL Abfrage kann von MySQL bei der Optimierung aufgelöst werden, kein Zugriff auf Tabelle oder Index, z.B. Minimum einer indizierten Spalte

possible_keys

Diese Spalte gibt an, welche Indizes für die Bearbeitung der Anfrage prinzipiell zur Verfügung stehen. Die hier stehenden Werte werden bereits in einer frühen Phase der Optimierung ermittelt, letzendlich wird in der Regel nur ein Index genutzt. Sind hier viele Indizes aufgeführt deutet das auf ein Problem hin.

key

Die Spalte key gibt an, welcher Index der Optimierer für die Anfrage gewählt hat. Dies kann auch ein abdeckender Index sein, aus dem die Ergebniswerte gelesen werden können ohne dass die eigentliche Tabelle gelesen werden muss. Ein Wert von NULL in der Spalte key bedeutet, dass kein Index genutzt wird und ist ein starkes Indiz für einen Optimierungsbedarf.

key_len

wie viel Byte (Spaltenbreite) eines Index werden benutzt welche Spalten des Index werden genutzt, von links beginnend

ref

-welche Spalten aus früheren Tabellen werden benutzt, um in dem key-Index nachzuschlagen

rows
Die Zahl in der Spalte rows ist eine Schätzung für die Anzahl der Zeilen, die gelesen werden müssen. Bei Abfragen mit mehreren Tabellen bezieht sich diese Angabe pro Schleife im Nested-Loop-Join-Plan. Die Schätzung beruht auf Statistiken kann ungenau sein. Im besten Fall steht hier eine 1.

filtered
Die Spalte ist neu seit MySQL 5.1 und erescheint nur bei EXPLAIN EXTENDED (s.u.). Es ist eine pessimistische Schätzung des Prozentsatzes der Zeilen, die eine Bedingung erfüllen, wie z.B. eine WHERE-Klausel oder ein JOIN mit einer anderen Tabelle.

Extra
Die Extra-Spalte enthält weitere Angaben, die nicht in die anderen Spalten passen:

Using Index abdeckender Index wird genutzt, d.h. die angefragten Daten müssen nicht aus der Tabelle gelesen werden

Using where die Zeilen werden nachträglich gefiltert, d.h. für die WHERE-Bedingung wird nicht der Index genutzt

Using temporary Erstellung einer temporäre Tabelle für Sortierung

Using filesort externe Sortierung, im RAM oder auf den Datenträger

ranke checked for each record (index map: N) kein geeigneter Index vorhanden, N ist ein Bitmap auf die Spalten in possible_keys

3.3.1 EXPLAIN EXTENDED

Wird EXPLAIN EXTENDED anstatt von EXPLAIN verwendet, dann erscheint die zusätzliche Spalte **filtered**. Außerdem werden weitere Informationen generiert, die mit dem nachfolgenden SQL-Kommando SHOW WARNINGS angezeigt werden können. Eine ausführliche Beschreibung findet sich unter [6].

3.3.2 EXPLAIN PARTITIONS

Wird EXPLAIN PARTITIONS verwendet, dann zeigt zusätzlichen Spalte **partitions** die Partitionen, auf welche die Anfrage zugreift, sofern welche verfügbar sind. Diese Option ist erst seit MySQL 5.1 verfügbar. Das Schlüsselwort EXTENDED kann nicht zusammen mit EXPLAIN PARTITIONS verwendet werden.

3.4 Abfragen mit mehreren Tabellen

3.5 Optimierungsmöglichkeiten und Benchmarking

Wichtigsten Spalten key (benutzer Index), rows (Anzahl Zeilen bearbeitet), type

QEP ist nicht fix, wird bei jeder Abfrage neu erstellt. Daher EXPLAIN einer Abfrage mit verschiedenen Beispielwerten. Nicht mit Test-Daten testen, sondern RealWord-Daten, z.B. Backups.

ALTER TABLE users ADD INDEX

Vorsicht beim Anlegen von INDIZES: Kann bei grossen Tabellen sehr lange Dauern und blockiert in dieser Zeit die Tabelle. Vorher Informationen über die Grösse der Tabellen holen, am besten einen Probedurchlauf auf einer Kopie machen.

Wird ein vorhandener Index bei der Anfrage nicht genutzt, obwohl er augenscheinlich genutzt werden sollte, dann kann das an veralteten Tabellen-Statistiken oder einer ungenügenden Stichprobe für die Statistiken liegen. Die Statistiken für eine Tabelle lassen sich dann mit ANALYZE TABLE Tabellenname aktualisieren.

3.6 Visuelles EXPLAIN (graphische Werkzeuge)

4 Fazit und Ausblick

Beschränkungen! Optimierung wichtig Mit Explain möglich nicht immer exakte Angaben Kontrolle der Optimierung mit Benchmarks nötig möglichst bereits in den Entwicklungsprozess integrieren, und nicht erst wenn es brennt

5 Anhang

Beispieldaten und -Scripte
Beispieldaten und Scripte können auf GitHub heruntergeladen werden (ca. 18 MB):
`https://github.com/oliworx/MySQL-EXPLAIN/archive/master.zip`

Setup des Testsystems
Sämtliche Tests wurden auf einem Notebook mit folgender Konfiguration durchgeführt:

Model Lenovo Thinkpad Edge 15 0319-A18
CPU Intel® Pentium® P6200 Prozessor, 2x 2,13 GHz , 3 MB Cache
RAM 4 GB, DDR3 SDRAM , PC3 8500 (1066 MHz)
HDD 320 GB, 2,5“, 7200rpm
OS Ubuntu 14.04 LTS, 64 Bit
DB MySQL 5.6.19,query_cache_type=OFF, Workbench 6.0.8.11354,

Literatur

[1] Schwartz, B., Zaitsev, P., Tkachenko, V., Zawodny, J.D.,Lentz, A., Balling, D.J. (2009) *High Performance MySQL. Optimierung, Datensicherung, Replikation & Lastverteilung*, 2. Auflage, O'Reilly Verlag, 2009

[2] Sauer, H. (1998) *Relationale Datenbanken, Theorie und Praxis*, 4. Auflage, Addison Wesley Longman Verlag, 1998

[3] Bradford, R. (2011) *Effective MySQL: Optimizing SQL Statements*, Oracle Press/McGraw-Hill Osborne Media, 2011

[4] ORACLE MySQL Documentation (2014):*Optimizing Queries with EXPLAIN.* URL: http://dev.mysql.com/doc/refman/5.6/en/using-explain.html , Abruf am 28.6.2014

[5] ORACLE MySQL Documentation (2014):*EXPLAIN Output Format.* URL: http://dev.mysql.com/doc/refman/5.6/en/explain-output.html , Abruf am 28.6.2014

[6] ORACLE MySQL Documentation (2014):*EXPLAIN EXTENDED Output Format.* URL: http://dev.mysql.com/doc/refman/5.6/en/explain-extended.html , Abruf am 28.6.2014

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit von mir selbstständig und ohne unerlaubte Hilfe angefertigt worden ist, insbesondere dass ich alle Stellen, die wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen sind, durch Zitate als solche gekennzeichnet habe. Ich versichere auch, dass die von mir eingereichte schriftliche Version mit der digitalen Version übereinstimmt. Weiterhin erkläre ich, dass die Arbeit in gleicher oder ähnlicher Form noch kei- ner anderen Prüfungsbehörde vorgelegen hat. Ich erkläre mich damit einver- standen/nicht einverstanden, dass die Arbeit der Öffentlichkeit zugänglich ge- macht wird. Ich erkläre mich damit einverstanden, dass die Digitalversion dieser Arbeit zwecks Plagiatsprüfung auf die Server externer Anbieter hoch geladen werden darf. Die Plagiatsprüfung stellt keine Zurverfügungstellung für die Öf- fentlichkeit dar.

München, 30. Juni 2014

Oliver Kurmis