Proseminar Datenbanksysteme

Universität Innsbruck — Institut für Informatik

Bottesch R., Hupfauf B., Kelter C., Mayerl M., Moosleitner M., Peintner A., Zangerl D.

10.01.2023

Übungsblatt 10 – Proposed Solution

Discussion Part (solve in the PS; no submission needed)

a) ★ Diskutieren Sie, ob ein Datenbanksystem für alle möglicherweise auftretenden Anfragen optimiert werden kann und begründen Sie Ihre Antwort. Falls nein: bezüglich welcher Abfragen sollte ein Datenbanksystem optimiert werden? Wie finden Sie diese Abfragen?

Solution ✔

Ein Datenbanksystem kann nicht auf jede möglicherweise auftretende Anfrage optimiert werden. So können z.B. nicht lesende und schreibende Abfragen meist nicht gleich stark unterstützt werden, da sich die Anforderungen an Indexstrukturen etc. stark unterscheiden. Hier wird also meist auf eine Query-Art optimiert oder versucht, ein passendes Gleichgewicht zu finden. Ein Datenbanksystem sollte prinzipiell immer auf Performance-kritische Queries optimiert werden. Dies sind meist Queries, die oft ausgeführt werden.
Diese Abfragen kann man auf verschiedene Weise aufspüren: einerseits im Gespräch mit den Anwendungsentwicklern und den (direkten) Datenbank-Nutzern und andererseits über Datenbank-Logging. Beim Logging kann die Datenbank so konfiguriert werden, dass alle Queries oder auch nur jene Queries die z.B. länger als eine konfigurierbare Sekundenanzahl zur Berechnung benötigen, geloggt werden. Dieses Log kann dann dazu verwendet werden, die kritischen und häufigsten Abfragen und Abfragearten festzustellen und das System dahingehend zu optimieren.

b) ★★ Man spricht oft davon, dass eine Query direkt “aus dem Index beantwortet werden kann” (“index-only”). Was bedeutet das für die Query-Abarbeitung und für die Performance der Datenbank? Welche Bedingungen müssen dafür erfüllt sein?

Solution ✔

Eine Query kann aus dem Index beantwortet werden, wenn zur Berechnung des Ergebnisses nur Daten direkt aus den Indexstrukturen (die bestenfalls im RAM liegen) benötigt werden. Entsprechend benötigt die Berechnung keinen Zugriff auf Hintergrundspeicher über den Datenpointer und kann somit performanter abgearbeitet werden. Die Bedingung für eine Performancesteigerung ist, dass für die Bearbeitung der Abfrage nur Daten herangezogen werden müssen, die in einer passenden Indexstruktur enthalten sind. Hier gehen wir davon aus, dass die Indexstruktur bereits im RAM liegt.

Homework Part (solve at home; submission required)

Exercise 1 (Tooling) [5 Points]

Ziel dieser Aufgabe ist es, Ihnen zu zeigen, welche Tools Ihnen PostgreSQL zur Verfügung stellt, um Ihnen beim Tunen einer Datebank und beim Lösen von Problemen behilflich zu sein. Die Tools, die wir hier behandeln, gehören allesamt zum sogenannten `Statistics Collector` von PostgreSQL. Beginnen Sie daher, indem Sie sich die Dokumentation[1] dazu durchlesen. Falls nötig, konfigurieren Sie Ihr System so, dass es Statistiken sammelt.

a) *2 Points* Der Leiter der Finanzabteilung kommt zu Ihnen — dem Datenbankadministrator Ihres Arbeitgebers — und informiert Sie darüber, dass das Buchhaltungssystem "hängt". Bei jeder Operation, die Buchungen aus der Datenbank auslesen bzw. Buchungen in diese schreiben soll, reagiert das System nicht mehr. Andere Operationen, wie zum Beispiel das Auslesen von Kundendaten, laufen einwandfrei.

Was könnte hier die Ursache sein? Sehen Sie sich die View `pg_stat_activity` an. Welche Informationen können Sie aus der Ausgabe dieser View herauslesen? Können Sie diese View verwenden, um das Problem zu lösen? Wie würden Sie vorgehen?

Abgabe

1a.pdf

Solution

Die wahrscheinliche Ursache hier ist, dass eine sehr langsame Transaktion eine Sperre auf die Buchungstabelle hält und alle späteren Operationen dadurch warten bzw. irgendwann in ein Timeout laufen.
Um das zu überprüfen und die langsame Transaktion zu identifizieren, können Sie die `pg_stat_activity`-View benutzen. Hier sollten, falls die Ursache tatsächlich eine langsame Transaktion ist und im System einige nicht reagierende Transatkionen laufen (die "hängenden" Prozesse der Anwender) einige Zeilen zurückbekommen, die warten (zu erkennen am Inhalt der Spalten `wait_event_type` und `wait_event`).
Auch die langsame Transaktion können Sie mit dieser View identifizieren. Die Spalte `xact_start` sagt Ihnen etwa, wann eine Transaktion begonnen hat.

b) *3 Points* Ihr Chef beauftragt Sie damit, die aktuelle Indizierungsstrategie für eine von Ihrem Arbeitgeber betriebene Datenbank zu evaluieren. Sie sollen also ermitteln, ob die aktuell vorhandenen Indizes ihren Zweck erfüllen, ob einige davon eventuell gelöscht werden könnten und ob Indizes fehlen.

Welche Views, die Ihnen der Statistics Collector zur Verfügung stellt, können Sie hier verwenden? Wie würden Sie vorgehen?

Abgabe

1b.pdf

[1] https://www.postgresql.org/docs/current/monitoring-stats.html

Solution ✔

Hier sind drei Views besonders nützlich: `pg_stat_user_indexes`, `pg_statio_user_indexes` und `pg_stat_user_tables`.
Die ersten beiden Views sagen Ihnen, inwiefern vorhandene Indizes wirklich genutzt werden. Mit `pg_stat_user_indexes` können Sie herausfinden, wie oft ein Index insgesamt benützt wurde (`idx_scan`) und wie viele Zeilen dadurch ausgelesen wurden (`idx_tup_read` bzw. `idx_tup_fetch`). Mithilfe von `pg_statio_user_indexes` können Sie sich ein Bild davon verschaffen, was das für Zugriffe auf den Hintergrundspeicher bedeutet.
Kandidaten für Indizes, die eventuell gelöscht werden können, sind dann solche, die kaum benutzt werden.
Um festzustellen, ob Indizes fehlen, können Sie sich die View `pg_stat_user_tables` ansehen. Diese enthält eine Zeile pro Tabelle in der Datenbank. Die Spalten `seq_scan` und `idx_scan` sagen Ihnen, wie oft auf diese Tabelle über sequentielle Scans bzw. über Indizes zugegriffen wurden. Ist die Anzahl der sequentiellen Zugriffe hier hoch und die Anzahl der Index-Zugriffe niedrig (oder gar 0), ist das ein Hinweis dafür, dass Indizes auf diese Tabelle fehlen könnten.

Exercise 2 (Query-Tuning) [5 Points]

In dieser Aufgabe werden Sie verschiedene Möglichkeiten anwenden, eine gegebene Abfrage zu optimieren. Wir werden uns dabei auf zwei Möglichkeiten konzentrieren:

- Optimieren einer Abfrage, indem wir sie anders formulieren (Stichwort Rewriting).
- Optimieren einer Abfrage, indem wir die physische Datenstruktur der Datenbank so ändern bzw. erweitern, dass die Abfrage schneller beantwortet werden kann (Stichwort Indizes).

Wir werden bei den folgenden Aufgaben zum Teil mit der `Pagila` Datenbank arbeiten. Da in dieser Datenbank allerdings schon recht viele Indizes definiert sind, müssen Sie diese zuerst löschen, damit Sie die folgenden Aufgaben richtig bearbeiten können. Führen Sie dazu in Ihrer `Pagila` Datenbank folgende Query aus:

```
1   DROP INDEX
2       film_fulltext_idx,
3       idx_actor_last_name,
4       idx_fk_address_id,
5       idx_fk_city_id,
6       idx_fk_country_id,
7       idx_fk_customer_id,
8       idx_fk_film_id,
9       idx_fk_inventory_id,
10      idx_fk_language_id,
11      idx_fk_original_language_id,
12      idx_fk_payment_p2017_01_customer_id,
13      idx_fk_payment_p2017_01_staff_id,
14      idx_fk_payment_p2017_02_customer_id,
15      idx_fk_payment_p2017_02_staff_id,
```

```
16      idx_fk_payment_p2017_03_customer_id,
17      idx_fk_payment_p2017_03_staff_id,
18      idx_fk_payment_p2017_04_customer_id,
19      idx_fk_payment_p2017_04_staff_id,
20      idx_fk_payment_p2017_05_customer_id,
21      idx_fk_payment_p2017_05_staff_id,
22      idx_fk_payment_p2017_06_customer_id,
23      idx_fk_payment_p2017_06_staff_id,
24      idx_fk_staff_id,
25      idx_fk_store_id,
26      idx_last_name,
27      idx_store_id_film_id,
28      idx_title,
29      idx_unq_manager_staff_id,
30      idx_unq_rental_rental_date_inventory_id_customer_id
```

Bearbeiten Sie dann die folgenden Aufgaben.

a) *2 Points* Gegeben sei die folgende Abfrage (auf der bekannten `Pagila`-Datenbank):

```
1   SELECT first_name, last_name, count(film_id)
2   FROM film_actor
3   INNER JOIN actor
4   ON actor.actor_id = film_actor.actor_id
5   WHERE film_id != ALL(SELECT film_id FROM film WHERE length < 120)
6   GROUP BY first_name, last_name
7   ORDER BY first_name, last_name
```

Diese Abfrage wird von PostgreSQL relativ ineffizient abgearbeitet. Finden Sie eine äquivalente Abfrage, welche das gleiche Ergebnis liefert und effizienter ausgeführt wird. Messen Sie mithilfe Ihres Client-Tools (etwa pgAdmin), wie lange beide Varianten für die Ausführung benötigen. Sehen Sie sich weiters die Ausführungspläne der beiden Varianten an und versuchen Sie, zu erklären, warum eine Variante schneller ist als die andere. Schreiben Sie Ihre Überlegungen und alle zugehörigen Materialien (z.B. die Ausführungspläne) in einem PDF-Dokument zusammen.

Abgabe

2a_faster.sql

2a_explanations.pdf

Solution

```
1   SELECT first_name, last_name, count(film_id)
2   FROM film_actor
3   INNER JOIN actor
```

```
4    ON actor.actor_id = film_actor.actor_id
5    WHERE film_id NOT IN (SELECT film_id FROM film WHERE length < 120)
6    GROUP BY first_name, last_name
7    ORDER BY first_name
```

PostgreSQL bereitet im Optimierungsvorgang einen gehashten Subplan vor, der die Ausführungszeit verkürzt. Dies gilt aber nur in diesem speziellen Fall, da die Subquery nur eine geringe Anzahl an Zeilen zurück gibt. Angenommen die Subquery returniert mehrere 1000 Zeilen, dann wäre dieser Ausführungsplan nicht mehr performant, da das Ergebnis des Subplans nun materialisiert wird. Daraus folgt, dass die Verwendung des IN Statements nur Vorteile bringt, wenn der Subplan weniger Zeilen auswählt.
Folgende Queries liefern (auf unserem Testsystem) ähnliche Performances wie die ursprüngliche Lösung:

```
1    SELECT first_name, last_name, count(film_id)
2    FROM film_actor
3    INNER JOIN actor
4    ON actor.actor_id = film_actor.actor_id
5    WHERE NOT EXISTS
6     (SELECT film_id FROM film WHERE length < 120
7        AND film.film_id = film_actor.film_id)
8    GROUP BY first_name, last_name
9    ORDER BY first_name, last_name
```

oder

```
1    SELECT first_name, last_name, count(film_actor.film_id)
2    FROM film_actor
3    INNER JOIN actor
4    ON actor.actor_id = film_actor.actor_id
5    LEFT JOIN film ON film_actor.film_id = film.film_id
6        AND length < 120
7    WHERE film.film_id IS NULL
8    GROUP BY first_name, last_name
9    ORDER BY first_name, last_name
```

b) *3 Points* Gegeben sei die folgende Abfrage. Diese müssen Sie vorerst nicht ausführen — wir machen erst eine theoretische Betrachtung.

```
1    SELECT        customer_id,
2                  last_name,
3                  first_name
4    FROM          customer
5    ORDER BY      last_name ASC
```

Nehmen Sie an, die `customer`-Tabelle enthält 1.000.000 Zeilen. Nehmen Sie weiters an, dass

auf der Tabelle keine Indizes definiert sind. Welche(n) Index/Indizes können Sie definieren, damit die Ausführung dieser Query eventuell beschleunigt wird?

Führen Sie nun die Abfrage auf der `Pagila`-Datenbank aus, messen Sie, wie lang die Ausführung dauert, und sehen Sie sich den Ausführungsplan an (Speichern nicht vergessen — der Ausführungsplan muss in der Abgabe enthalten sein!). Erstellen Sie anschließend in der `Pagila`-Datenbank den Index bzw. die Indizes, welche(n) Sie vorhin entworfen haben. Starten Sie PostgreSQL neu (um eventuelle Einflüsse durch Caching zu vermeiden) und führen Sie die Abfrage erneut aus. Messen Sie die benötigte Zeit und speichern Sie den Ausführungsplan. Gibt es einen Unterschied? Wurde(n) der Index/die Indizes verwendet? Wenn ja, wie? Wenn nein, welchen Grund könnte das Ihrer Meinung nach haben? Recherchieren Sie und schreiben Sie Ihre Überlegungen und alle zugehörigen Materialien in einem PDF-Dokument zusammen.

Abgabe

- `2b_index.sql`
- `2b_explanations.pdf`

Solution

Für die gegebene Abfrage bietet sich ein sogenannter *covering index* an. Diesen erstellen wir zum Beispiel wie folgt:

```
1   CREATE INDEX   idx_customer_lastname_firstname_id
2   ON             customer(
3       last_name,
4       first_name,
5       customer_id
6   )
```

Wir indizieren hier `last_name` als erstes, weil damit eventuell die Sortierungsoperation eingespart werden kann (das hängt aber davon ab, wie genau das Datenbanksystem die Operation umsetzt).
Wenn wir all das jetzt auf der `Pagila`-Datenbank ausführen, werden wir wahrscheinlich feststellen, dass PostgreSQL diesen Index ignoriert. Dies liegt daran, dass der Optimierer von PostgreSQL zu dem Schluss kommt, dass sich die Verwendung des Indexes nicht lohnt. In diese Entscheidung fließen mehrere Faktoren ein:

- Die Datenmengen in der `Pagila`-Datenbank sind recht klein. Dadurch sind *sequential scans* vergleichsweise günstig, da nicht viele Pages gelesen werden müssen und die Datenbank dadurch vergleichsweise einfach im RAM Platz hat.
- Für sogenannte *index only scans* ist aufgrund der Art und Weise, wie die physische Speicherorganisation von PostgreSQL funktioniert, ein gewisser Overhead erforderlich. *Index only scans* können “schief gehen”, wodurch doch wieder die Daten der eigentlichen Tabelle gelesen werden müssen. Bevor PostgreSQL weiß, ob dies notwendig ist, muss es in der sogenannten *visibility map* nachsehen. Diese Operation verursacht einen Overhead.

Important: Submit your solution to OLAT and mark your solved exercises with the provided checkboxes. The deadline ends at 23:59 on the day before the discussion.