Proseminar Datenbanksysteme

Universität Innsbruck — Institut für Informatik

Antensteiner T., Bottesch R., Kelter C., Moosleitner M., Peintner A.

16.01.2024

Übungsblatt 11

Diskussionsteil (im PS zu lösen; keine Abgabe nötig)

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?

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?

c) ★★★ Um Datenkbanken sinnvoll zu nutzen, müssen wir oft von Applikationen aus auf eine Datenbank zugreifen, um Daten entweder auszulesen oder in die Datenbank zu schreiben. Beantworten Sie dazu folgende Fragen:

- Was versteht man unter dem Cursor-Prinzip? Wofür wird es verwendet?
- Was sind ODBC und JDBC?

d) ★★★ Zur Vorbereitung auf einen Teil der Hausaufgabe, wollen wir von einer Applikation aus auf die `pagila` Datenbank zugreifen und Daten auslesen. Schreiben Sie zu diesem Zwecke eine kleine Applikation, die sich mit der Datenbank verbindet, die Spalten `title` und `rental_rate` aus der Tabelle `film` ausließt und das Ergebnis auf der Konsole ausgibt. Berechnen Sie in Ihrem Programm weiters die Summe über die Spalte `rental_rate` und geben Sie diese nach den Zeilen der Tabelle aus.

Hinweis ⚠

Starten Sie, je nach Wahl Ihrer Programmiersprache, mit den folgenden Dokumentationsseiten:

- C/C++: https://www.postgresql.org/docs/13/libpq.html
- Java: https://jdbc.postgresql.org/documentation/
- Python: https://www.psycopg.org/docs/

Hausaufgabenteil (Zuhause zu lösen; Abgabe nötig)

Aufgabe 1 (Tooling) [3 Punkte]

Ziel dieser Aufgabe ist es Ihnen zu zeigen, welche Tools PostgreSQL für Datenbanktuning und zur Lösung von Performanceproblemen bereitstellt. Die hier behandelten Tools gehören allesamt zum sogenannten `Statistics Collector` von PostgreSQL. Beginnen Sie daher, indem Sie sich die Dokumentation[1] dazu durchlesen.

Hinweis

Stellen Sie sicher, dass Ihr System so konfiguriert ist, dass Statistiken gesammelt werden. Im Artikel `Settings Parameters`[a] finden Sie Informationen darüber, wie Sie die Einstellungen Ihrer PostgreSQL Datenbank ändern können.

[a] https://www.postgresql.org/docs/15/config-setting.html

a) *1.5 Punkte* Sie administrieren die Datenbank eines Buchhaltungssystems und bei jeder Operation, die Buchungen aus der Datenbank ausliest bzw. Buchungen in diese schreiben soll, "hängt" die Datenbank und damit auch das Buchhaltungssystem. Andere Operationen, wie zum Beispiel das Auslesen von Kundendaten, laufen einwandfrei.

Sehen Sie sich die View `pg_stat_activity` an und beantworten Sie die folgenden Fragen.

- Was könnte hier die Ursache sein?
- 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

b) *1.5 Punkte* Sie werden beauftragt, die aktuelle Indizierungsstrategie für eine Datenbank zu evaluieren. Sie sollen ermitteln, ob die vorhandenen Indizes ihren Zweck erfüllen, ob einige davon eventuell gelöscht werden könnten und ob neue Indizes angelegt werden sollten.

Welche Views, die Ihnen der Statistics Collector zur Verfügung stellt, können Sie hier verwenden? Wie sieht Ihr weiteres Vorgehen aus?

Abgabe

1b.pdf

Aufgabe 2 (Query-Tuning) [3 Punkte]

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).

[1] https://www.postgresql.org/docs/15/monitoring-stats.html

- 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. Löschen Sie zu zuerst die Indizes in Ihrer `pagila` Datenbank (siehe `drop_indices.sql` OLAT) und bearbeiten Sie dann die folgenden Aufgaben.

a) *1.5 Punkte* 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 die 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`

b) *1.5 Punkte* 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 Tabelle `customer` 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`

Aufgabe 3 (Datenbankanbindung) [4 Punkte]

In dieser Aufgabe werden Sie, ähnlich wie in der Diskussionsaufgabe, eine kleine Anwendung schreiben, welche mit der `pagila` Datenbank kommunizieren soll. Für die Wahl der Programmiersprache und Bibliothek, welche Sie für diese Aufgabe verwenden, stellen wir Ihnen folgende Optionen frei:

- C/C++ (mit `libpq`)
- Java (mit `pgJDBC`)
- Python (mit `Psycopg`)

Wählen Sie aus diesen Sprachen jene aus, mit der Sie am meisten Erfahrung haben. Sie finden im OLAT je eine Vorlage (`exercise3.c`, `exercise3.py`, `Exercise3.java`). Diese Vorlagen enthalten ein einfaches Konsolen-Interface, über welches Berichte aus der Pagila-Datenbank abgerufen werden können sollen.

a) *0.5 Punkte* Implementieren Sie die Verbindungsverwaltung für den Zugriff auf die Datenbank. Achten Sie hier besonders auf korrekte Fehlerbehandlung und darauf, dass die Anwendung sinnvolle Fehlermeldungen ausgibt, wenn beim Verbindungsaufbau etwas schiefgeht.

b) *1.5 Punkte* Implementieren Sie den Befehl `rental_report`, mit welchem sich ein Bericht über die erfolgten Entleihungen in einem frei wählbaren Zeitraum angezeigt werden soll. Verwenden Sie hierfür die SQL-Abfrage aus der Datei `3_query.sql` (OLAT) - der Bericht gibt aufgeschlüsselt nach Kategorie spaltenweise die Anzahl der entliehenen Filme pro Wochentag aus.

Der Datumsfilter für den Bericht muss über die Kommandozeile eingelesen werden. Ein Beispiel für die Verwendung dieses Befehls könnte etwa wie folgt aussehen:

```
(Pagila)> rental_report
From: 2005-06-01
To: 2005-06-30
|category_name| mon | tue | wed | thu | fri | sat | sun | total |
|-------------|-----|-----|-----|-----|-----|-----|-----|-------|
|Action       |  30 |  38 |  75 |  74 |  73 |  83 |  91 |   464 |
|Animation    |  34 |  46 |  89 |  85 |  82 |  81 |  72 |   489 |
|Children     |  23 |  34 |  68 |  57 |  80 |  64 |  80 |   406 |
|Classics     |  19 |  39 |  53 |  54 |  81 |  80 |  58 |   384 |
|Comedy       |  37 |  30 |  58 |  61 |  75 |  69 |  53 |   383 |
|Documentary  |  30 |  33 |  81 |  69 |  68 |  65 |  83 |   429 |
|Drama        |  36 |  31 |  79 |  73 |  84 |  87 |  73 |   463 |
|Family       |  27 |  41 |  88 |  65 |  62 |  83 |  94 |   460 |
|Foreign      |  33 |  35 |  73 |  68 |  83 |  70 |  70 |   432 |
```

```
|Games       |  25 |  30 |  67 |  77 |  59 |  67 |  67 |   392 |
|Horror      |  22 |  27 |  62 |  71 |  56 |  61 |  67 |   366 |
|Music       |  18 |  36 |  66 |  56 |  64 |  56 |  52 |   348 |
|New         |  28 |  36 |  64 |  68 |  66 |  60 |  67 |   389 |
|Sci-Fi      |  41 |  30 |  76 |  77 |  73 |  73 |  92 |   462 |
|Sports      |  28 |  39 |  99 |  71 |  89 |  81 |  90 |   497 |
|Travel      |  30 |  30 |  55 |  55 |  58 |  67 |  50 |   345 |
```

Achten Sie auch hier wieder auf eine sinnvolle Fehlerbehandlung.

c) *2 Punkte* In der vorherigen Unteraufgabe wurde eine komplexere Abfrage direkt im Programmcode verwendet. Eine Alternative dazu ist es, in der Datenbank eine tabellenwertige Funktion zu erstellen und diese aufzurufen, um das Resultat zu erhalten. Sehen Sie sich hierzu die PostgreSQL-Dokumentation für benutzerdefinierte Funktionen[2] an, im Speziellen die Funktionen mit einem `RETURNS TABLE` Ausdruck.

Legen Sie dann in Ihrer Pagila-Datenbank eine Funktion an, welche denselben Bericht wie in Unteraufgabe b) generiert (die Funktion muss also auch einen Datumsbereich als Parameter entgegennehmen!). Implementieren Sie anschließend in Ihrer Applikation den Befehl `rental_report_function`. Dieser Befehl soll die Query nicht direkt wie in Unteraufgabe b) übergeben, der Aufruf soll folgende Form haben:

```
SELECT * FROM report_function(from, to);
```

Abgabe

- `exercise3.py` oder
- `exercise3.c` oder
- `Exercise3.java`

  Wenn Sie Ihr Programm über mehrere Source-Dateien verteilt haben, geben Sie alle davon mit ab.

- `3c.sql`: Code der tabellenwertigen Funktion

Hinweis

Für die Lösung dieser Aufgabe verwenden Sie low-level Schnittstellen, um über einen Cursor auf die Datenbank zuzugreifen. In der Praxis verwendet man heute Werkzeuge wie Objekt-Relational-Mapper (ORM), um sauberer auf die Datenbank zuzugreifen. Sie werden in späteren Lehrveranstaltungen im Studium lernen, wie man diese verwendet.

Wichtig: Laden Sie bitte Ihre Lösung in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 16:00 ab.

[2] https://www.postgresql.org/docs/15/sql-createfunction.html