Aufgabenblatt 4
Praktische Übung: Optimierung

- Abgabetermin: **Dienstag, 02.01.2018 (23:59 Uhr)**
- Zur Prüfungszulassung muss ein Aufgabenblatt mit mind. 25% der Punkte bewertet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte.
- Die Aufgaben sollen in Zweiergruppen bearbeitet werden.
- Abgabesytem unter
  `http://www.dcl.hpi.uni-potsdam.de/submit`
  - ausschließlich pdf-Dateien
  - eine Datei pro Aufgabe namens Aufgabe-<aufgabenNr>.pdf
  - *jedes* Blatt beschriftet mit Namen

1 Hinweis

In dieser Übung verwenden wir eine lokale PostgreSQL (9.5 oder 9.6) Installation. Andere Versionen sind prinzipiell erlaubt, getestet haben wir allerdings mit nur den genannten Versionen und empfehlen daher diese zu verwenden. Notwendige Schritte vor Beginn der Übung:

a) Installieren Sie PostgreSQL

- **Hinweise zu Windows Setup und häufigen Problemen:**

  Laden Sie den Installer z.B. von dieser Webseite runter:

  `https://www.postgresql.org/download/windows/`

  Während der Installation gibt es die Option, Stack Builder mit zu installieren. Das ist für diese Übung **nicht** notwendig.

  Die Übung kann sowohl mit der SQL Shell "psql" als auch mit der UI "pgAdmin II" gelöst werden. Für das Importieren der Daten in die Datenbank ist die SQL Shell zu bevorzugen. Nach Start der Shell muss Enter gedrückt werden, bis man die Passworteingabe erreicht. Danach sollte es folgendermaßen oder ähnlich aussehen wie in Abbildung 1

  Zugriffsprobleme können im Allgemeinen durch die Ausführung als Administrator gelöst werden. Da der Backslash für Kommandos reserviert ist, müssen Backslashes in Pfaden durch Slashes ersetzt werden.

- **Hinweise zu Linux Setup (Ubuntu 16.04 LTS):** Eine Shell öffnen und *sudo apt-get install postgresql-contrib-9.5* ausführen

  Nun *sudo -u postgres psql* um PostgreSQL unter dem User postgres zu starten

  Nun sind sie mit der psql Shell verbunden. Sobald Sie eine Datenbank angelegt haben können Sie sich mit dieser direkt aus dem Terminal mit *psql <dbname>* verbinden.

b) Laden Sie die im Verzeichnis `Lehrveranstaltungen/FG_Informationssysteme/VL DBS II/uebung optimierung/` abgelegte Datei praktischeUebungOptimierung.sql.gz herunter und entpacken Sie diese. Importieren Sie die entpackte Datei in eine von Ihnen neu erzeug-

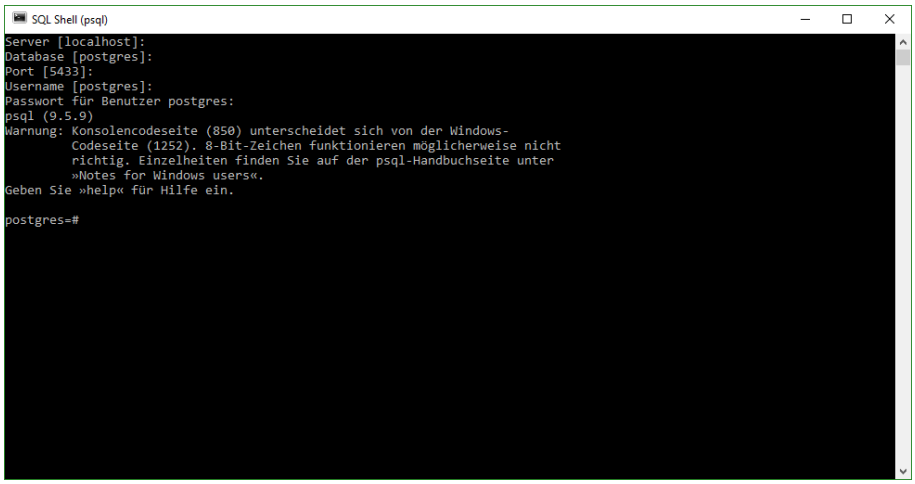

Abbildung 1: Windows SQL Shell nach erfolreicher Installation und Passworteingabe.

te Datenbank. Am einfachsten geht dies per PostgreSQL-Kommandozeile:

$$CREATE\ DATABASE\ < dbname >;$$
$$\backslash c\ < dbname >;$$
$$\backslash i\ < pathToFile >;$$

Die Daten wurden aus der International Movie Database (IMDB) extrahiert. Die Orginaldaten sind unter `ftp://ftp.fu-berlin.de/pub/misc/movies/database/temporaryaccess/` verfügbar.
Das Schema ist relativ überschaubar. Die Zentrale Tabelle ist *movies*, welche als Schlüssel den Titel eines Films hat. Die Tabellen *countries*, *directors*, *editors* und *actors* sind über Fremdschlüssel mit dieser Tabelle verknüpft. Es ist sicher hilfreich wenn Sie sich einen knappen Überblick über die Tabellen und deren Spalten verschaffen bevor Sie mit den Aufgaben anfangen. Ebenso kann es hilfreich sein sich einen kurzen Überblick über die grundlegenden Befehle zum Umgang mit Datenbanken und Tabellen in PostgreSQL zu verschaffen: `https://www.postgresql.org/docs/9.5/static/index.html`

Aufgabe 1: Erste Schritte mit Explain

Das Explain Statement nimmt eine SQL-Anfrage entgegen und gibt den Physischen Anfrageplan (QEP) des PostgreSQL Optimierers aus. Hinweis: Die Ausgabe in der Kommandozeile ist etwas kompliziert zu lesen, angenehmer ist die graphische Darstellung in pgAdmin. Die Ausgabe in der Kommandozeile ist aber völlig ausreichend um alle Aufgaben zu lösen. Eine detaillierte Dokumentation mit Erklärungen zum Explain-Befehl finden Sie unter `https://www.postgresql.org/docs/9.2/static/using-explain.html`.

a) Betrachten Sie folgende Anfrage:

```
EXPLAIN SELECT title FROM movies;
```

Warum wird hier ein sequential Scan ausgeführt obwohl ein Index zur Verfügung steht? Erweitern Sie die Anfrage, sodass ein Index-Scan verwendet wird. **2 P**

b) Betrachten Sie folgende Anfrage:

```
EXPLAIN SELECT * FROM movies,directors
        WHERE movies.title = directors.title
        AND directors.pseudonym IS NOT null;
```

Nennen Sie zwei verschiedene Möglichkeiten die Anfrage zu verändern, sodass kein Hashjoin verwendet wird (lediglich die beiden Tabellen, sowie die JOIN-Bedingung muss gleich bleiben, die Semantik der neuen Anfrage darf eine völlig andere sein). **4 P**

c) Schreiben Sie eine sinnvolle Anfrage, bei welcher der Optimierer einen Merge-Join verwendet. **2 P**

Aufgabe 2: EXPLAIN ANALYZE

Das EXPLAIN Statement kann durch ANALYZE ergänzt werden, wodurch die Query ausgeführt wird, sodass in der Ausgabe, sowohl die geschätzten, als auch die tatsächlichen Ausgabegrößen (angegeben in Tupeln, genannt 'rows'), sowie die tatsächliche Ausführungszeit zu sehen sind. Gegeben sei folgende Anfrage:

```
EXPLAIN ANALYZE SELECT m.releaseyear,d.name,p.author
                FROM movies m JOIN directors d JOIN editors e JOIN plots p
                ON (e.title = p.title) ON (d.title = e.title) ON (m.title = e.title)
                WHERE d.name = e.name AND p.author = 'Anonymous';
```

Führen Sie die Anfrage aus und studieren Sie die Ausgabe. Von nun an bezeichnen wir den hier resultierenden Plan als den O-QEP (Optimierer-Query Execution Plan).

a) An welcher Stelle hat der Optimierer sich deutlich verschätzt? **2 P**

b) Der PostgreSQL Optimierer kann in seinen Möglichkeiten eingeschränkt werden, indem man die Nutzung bestimmter Operatoren verbietet. Dies kann für für eine queryspezifische, eigens durchgeführte Optimierung genutzt werden. Die möglichen Parameter sind in der Dokumentation erklärt: `https://www.postgresql.org/docs/9.5/static/runtime-config-query.html`. Die Syntax für die Kommandozeile ist:

$SET < paramname > = < paramvalue >;$
$SHOW < paramname >;$ *(Zum Anzeigen)*
$SET < paramname > = DEFAULT;$ *(Zum Wiederherstellen des Defaultwertes)*

Um dem Optimierer zu verbieten die Join-Reihenfolge zu ändern, setzen Sie *join_collapse_limit = 1*. Damit die Neusortierung der joins verhindert werden kann müssen explizite JOIN-Statements verwendet werden (z.B. $a\ JOIN\ (b\ JOIN\ c\ ON\ (..))\ ON\ (..)$), bei einer Auflistung der Tabellen mit Kommata funktioniert dies nicht.
Nutzen Sie diese Optionen um einen anderen Plan zu erzeugen, welcher möglichst schnell (im Idealfall schneller als der O-QEP) ist. Mit Ausnahme der Join-Ordnung darf nichts an der Syntax der Query verändert werden. Vergleichen Sie nur die tatsächliche Gesamtzeit, welche von EXPLAIN ANALYZE ausgegeben wird. Geben Sie die von Ihnen veränderten Parameter, den neuen QEP und die beiden Zeiten an. Begründen Sie kurz, warum ihr Plan sinnvoll ist. **6 P**

c) Entwerfen Sie nun analog zu b) einen QEP, der die längstmögliche Zeit braucht. Erklären Sie kurz warum dieser Plan ungünstig ist. Sollte ihr QEP länger als die 100-fache Zeit des O-QEPs brauchen dürfen Sie die Zeitmessung abbrechen. **4 P**

WICHTIG: Vergessen Sie danach nicht für die folgenden Aufgaben die Veränderungen rückgängig zu machen!

Aufgabe 3: Optimierer-Einstellungen

Im PostgreSQL-Optimierer lassen sich diverse Parameter einstellen welche für die Kostenschätzung relevant sind. Insbesondere sind dies:

- seq_page_cost
- random_page_cost
- effective_cache_size

Eine genaue Dokumentation aller Parameter finden Sie unter Abschnitt 18.7.2. der PostgreSQL-Dokumentation: `https://www.postgresql.org/docs/9.5/static/runtime-config-query.html`.

a) Betrachten Sie die Default Werte für seq_page_cost und random_page_cost. Warum ist es sinnvoll, dass random_page_cost größer als seq_page_cost ist? **1 P**

b) Wie würden Sie das Verhältnis von seq_page_cost und random_page_cost ändern wenn Sie davon ausgehen, dass die gesamte Datenbank stets im Hauptspeicher ist? Begründen Sie ihre Antwort. **2 P**

c) Spielen Sie mit den Parametern und beobachten Sie die Veränderungen im ursprünglich ausgewählten Plan für die Query:

```
EXPLAIN SELECT a.name FROM plots p JOIN actors a
        ON p.title = a.title
        WHERE p.description LIKE '%England%'
        AND a.name LIKE '%Emma%'
        AND EXISTS ( SELECT 1 FROM countries c
        WHERE c.title = a.title);
```

Wählen Sie drei verschiedene Konfigurationen, welche alle zu einer Änderung des Plans führen. Geben Sie für jeden veränderten Plan die veränderten Parameter und den Plan selbst an und erklären Sie kurz warum diese Änderungen durch die neue Konfiguration entstehen.

WICHTIG: Vergessen Sie danach nicht für die folgenden Aufgaben die Veränderungen rückgängig zu machen! **6 P**

Aufgabe 4: Fehlschätzungen

Das in der Vorlesung erwähnte runstats() Program heißt in PostgreSQL ANALYZE. Analyze holt Statistiken über die entsprechenden Relationen ein, welche vom Optimierer genutzt werden um die Selektivität von Bedingungen und damit die Kosten von Anfrageplänen abzuschätzen. ANALYZE wird beim Importieren der Datenbank automatisch ausgeführt. Einen Einblick in die gesammelten Statistiken erhalten Sie über den view pg_stats (`https://www.postgresql.org/docs/9.3/static/view-pg-stats.html`). Schreiben Sie eine Query für die der Optimierer sich bezüglich der Ausgabegröße so sehr wie möglich verschätzt, mindestens jedoch um 40.000 Tupel. Eventuell sind die Statistiken hier hilfreich. Diese können Sie mithilfe von SELECT-Anfragen an pg_stats einsehen. Geben Sie ihre Query und die Ausgabe von EXPLAIN ANALYZE an. Erklären Sie an welchen Stellen der Optimierer sich verschätzt und geben Sie eine Vermutung ab wie die Fehlschätzung zustande kommt. **5 P**