Aufgabenblatt 6
Praktische Übung

- Abgabetermin: **Freitag, 31.01.2020 (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.
- Abgabesystem unter `http://www.dcl.hpi.uni-potsdam.de/submit`
  - für die erste Aufgabe: eine pdf-Datei, bitte mit Namen beschriftet
  - bei die Programmieraufgabe: bitte nur die angegebenen .java-Dateien bzw. eine pdf-Datei für die Fragen, zusammen in einem Zip-Archiv

Hinweis

Dieses Übungsblatt besteht aus zwei Teilen. Im ersten Teil betrachten wir den Optimierer von PostgreSQL, im zweiten Teil gibt es eine Programmierübung. Für den ersten Teil benötigen Sie eine lokale PostgreSQL Installation. Notwendige Schritte vor Beginn der Übung:

PostgreSQL-Installation

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

  Laden Sie den Installer z.B. von der offiziellen PostgreSQL-Seite:

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

  Während der Installation gibt es die Option, Stackbuilder 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 4" 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 das Fenster ähnlich zu Abbildung 1 aussehen.

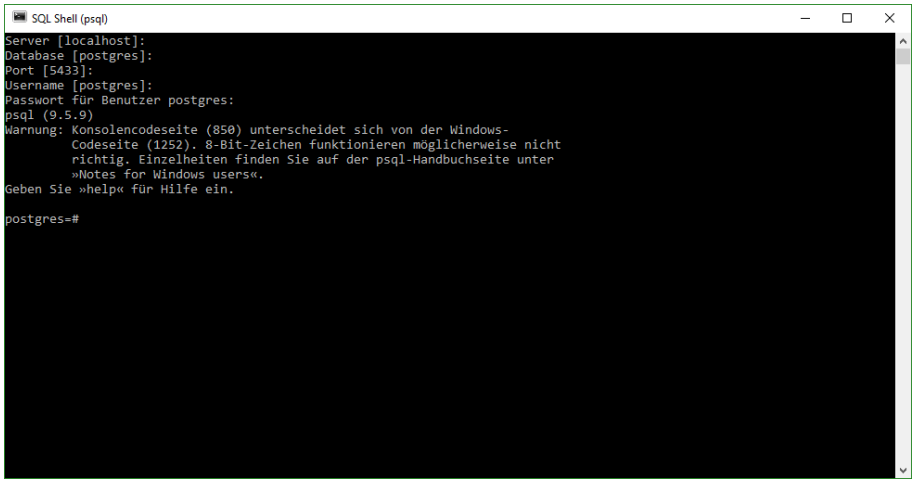

Abbildung 1: Windows SQL Shell nach erfolreicher Installation und Passworteingabe.

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 zum Setup unter Linux:**

  Der bequemste Weg, PostgreSQL unter Linux zu installieren, ist die Verwendung eines Paketverwaltungsystems, wie z.B. APT. Eine Anleitung dazu finden Sie im PostgreSQL-Wiki[1]. Nach der Installation können Sie mittels `sudo -u postgres psql` die psql shell unter dem User postgres starten. Sobald Sie eine Datenbank angelegt haben, können Sie sich mit dieser direkt aus dem Terminal mit `psql <dbname>` verbinden.

Laden der Beispieldaten

Laden Sie die im Material-Verzeichnis zur Vorlesung unter `uebung optimierung` abgelegte Datei `praktischeUebungOptimierung.sql.zip`[2] herunter und entpacken Sie diese.

Importieren Sie die entpackte Datei in eine von Ihnen neu erzeugte Datenbank. Am einfachsten geht dies per PostgreSQL-Kommandozeile:

```
CREATE DATABASE <dbname>;
\c <dbname>;
\i <pathToFile>;
```

Die Daten wurden aus der International Movie Database (IMDB) extrahiert[3]. 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[4].

[1] `https://wiki.postgresql.org/wiki/Apt`

[2] `https://hpi.de/internalfiles/materialien/FG%20Informationssysteme/VL%20DBS%20II/uebung%20optimierung/praktischeUebungOptimierung.sql.zip`

[3] `ftp://ftp.fu-berlin.de/pub/misc/movies/database/frozendata/`

[4] `https://www.postgresql.org/docs/12/index.html`

Aufgabe 1: EXPLAIN und EXPLAIN ANALYZE

Das `EXPLAIN` Statement nimmt eine SQL-Anfrage entgegen und gibt den physischen Anfrageplan 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 in der PostgreSQL-Dokumentation[5].

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. Geben Sie die veränderte Anfrage und die Ausgabe des EXPLAIN-Befehls auf dieser veränderten Anfrage an. **2 P**

b) Betrachten Sie folgende Anfrage:

```
EXPLAIN SELECT * FROM movies
                 JOIN directors
                 ON movies.title = directors.title
                 WHERE directors.pseudonym IS NOT NULL;
```

Welchen Join-Algorithmus verwendet der Optimierer hier? Nennen Sie zwei verschiedene Möglichkeiten die Anfrage zu verändern, sodass ein anderer Join-Algorithmus verwendet wird (die veränderte Anfrage soll weiterhin die beiden Tabellen und die gleiche Join-Bedingung verwenden, die Semantik der neuen Anfrage darf eine völlig andere sein). Geben Sie die veränderten Anfragen und die Ausgaben des `EXPLAIN`-Befehls auf diesen an. **4 P**

c) Schreiben Sie eine sinnvolle Anfrage, bei welcher der Optimierer einen Merge-Join verwendet. Geben Sie die Anfrage und die Ausgabe des `EXPLAIN`-Befehls auf dieser an. **2 P**

Das `EXPLAIN` Statement kann durch `ANALYZE` ergänzt werden, wodurch die Query ausgeführt wird, sodass in der Ausgabe, sowohl die geschützten, 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.

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

[5] https://www.postgresql.org/docs/12/sql-explain.html

Aufgabe 2: Join-Implementierung

In dieser Aufgabe geht es um die Implementierung von Join-Algorithmen in einem dafür zur Verfügung gestellten Java-Framework[6]. Sie können sich bei Ihrer Implementierung an der Nested Loop Join-Implementierung in der Klasse `NestedLoopEquiJoin` orientieren. Um uns auf den Algorithmus konzentrieren zu können, machen wir folgende Vereinfachungen:

- Es gibt nur den `String` Datentyp.
- Wir implementieren nur den Equi-join mit einer einzigen Bedingung, nämlich der Äquivalenz zweier Spalten.

Die Schnittstelle für den Join sowie Hilfskassen, Build-Prozess, Blockzustände etc. sind in der Markdown-Datei `README.md` im Root-Verzeichnis beschrieben. Beim Bearbeiten der Aufgabe gelten folgende Regeln:

- Der an den Join-Algorithmus übergebene `BlockManager` dient als einzige Schnittstelle, um Blöcke einzulesen bzw. neue Blöcke zu erzeugen bzw. deren Speicher wieder freizugeben. Der verfügbare Speicher ist begrenzt.
  Um diese Beschränkung einzusehen, gibt es die `getFreeBlockCount`-Methode.
- Sämtlicher weiterer Speicher, den Sie für den Algorithmus benötigen (z.B. für Datenstrukturen, die Referenzen auf die bereits reservierten Blöcke speichern), sollte so gering wie möglich gehalten werden.

Lösen Sie die folgenden Aufgaben:

a) Implementieren Sie in der schon angelegten Datei `HashEquiJoin.java` einen einfachen Hash-Join (keinen Hybrid-Hash-Join). Stellen Sie sicher, dass Ihre Implementierung ohne Änderungen in anderen Dateien auskommt und geben Sie auch nur diese Datei ab. **8 P**

b) Beantworten Sie außerdem folgende Fragen: **2 P**

- Warum weicht die Schätzung beim Hash-Join oft von den gemessenen IO-Kosten ab?
- Wie lässt sich diese Abweichung erklären? Auf welche Annahme aus der Vorlesung geht diese Abweichung zurück?
- Bei welchen Parameter-Einstellungen wird die Abweichung größer? Auf welche Annahme(n) aus der Vorlesung geht diese zusätzliche Abweichung zurück?

c) **Bonus:** Implementieren Sie in einen Sort-Merge-Join in einer neuen Klasse `SortMergeJoin` im Package `join.algorithms`. Stellen Sie sicher, dass Ihre Implementierung ohne Änderungen in anderen Dateien auskommt und geben Sie auch nur diese Datei ab. Zum Testen können Sie Ihren Algorithmus in die Liste `joinsToEvaluate` in der `Main`-Klasse einfügen. **1 Bonus(klausur)punkt**

Hinweise:

- Die Klasse `Main` kann verwendet werden, um die Join-Implementierungen zu testen. Drei Testdateien sind ebenfalls in den Materialien[7] zu finden. Die drei Relationen haben die Film-ID als erstes Attribut, welches sich gut als Join-Attribut eignet. Das Join-Ergebnis für einen solchen Join von `title.basics.sample` and `title.principals.sample` umfasst 3563 Tupel.
- Nutzen Sie gerne die Mailingliste, z.B. für Probleme beim build bzw. allgemeine Fragen.

[6] `https://github.com/HPI-Information-Systems/dbs2-join-exercise`

[7] `https://hpi.de/internalfiles/materialien/FG%20Informationssysteme/VL%20DBS%20II/Programmier%C3%BCbung/imdb.sample.zip`