Datenbanken II

Wintersemester 2018/19

Praktische Aufgabe 2

abzugeben bis: Montag, 04. Februar 2019, 23:55 Uhr

---

1. Allgemeines

Geben Sie bitte Ihr fertiges und *selbst geschriebenes* PostgreSQL-Skript bis spätestens **Montag, 04. Februar 2019, 23:55 Uhr**, über das Abgabesystem [1] ab. Das System führt anschließend automatisierte Tests durch und Sie erhalten eine entsprechende Rückmeldung vom Abgabesystem.

Beachten Sie, dass Sie mindestens eine korrekte Programmieraufgabe abgeben müssen, damit Sie das Proseminar positiv absolvieren können. Eine Programmieraufgabe gilt als korrekt, sobald alle automatisch durchgeführten Tests korrekt abgearbeitet wurden. Bitte beachten Sie auch, dass nur die **letzte** Abgabe gewertet wird.

1.1. Betreuung

Bei Unklarheiten oder Verständnisproblemen gibt es für die Studierenden folgende Möglichkeiten diese abzuklären:

1. Slack Channel **#db2** [2] (bevorzugter Kommunikationskanal)
2. Tutorium (Christoph Siller, Sebastian Landl, mittwochs, 16:00 - 17:00 im SR T06)

Nehmen Sie *rechtzeitig* eine der Möglichkeiten wahr, um Fragen abzuklären. Je früher Sie sich mit der Programmieraufgabe auseinandersetzen, desto früher können Sie Fragen stellen und desto einfacher ist es für die LV-Leitung bzw. den Tutor diese zu beantworten.

---

[1] `https://abgaben.cosy.sbg.ac.at`
[2] `https://dbteaching.slack.com`

2. Aufgabenstellung

In dieser Aufgabe werden Sie Anfragen auf einem Teil der IMDB Datenbank [3] formulieren und optimieren. Weiters werden Sie Fragen zu den jeweiligen Anfragen und Optimierungen beantworten. Alle PostgreSQL-Befehle und Ihre Antworten werden in einer Datei `questions.txt` gesammelt abgegeben. Jede/r korrekte/r Antwort/Befehl bringt Ihnen 0.4 Punkte (0.4p). Es werden insgesamt 25 Fragen/Befehle gestellt.

Die Datensätze der IMDB Datenbank werden von uns bereitgestellt (`https://kitten.cosy.sbg.ac.at/index.php/s/9tSGgRjjdDB4Ryo`). Die Anfragen sollen *ausschließlich* auf diesen Datensätzen formuliert werden. Es werden 3 Tabellen mit den folgenden Schemata bereitgestellt:

Tabellenname: `titles`

```
tconst CHAR(9) PRIMARY KEY
titleType TEXT
primaryTitle TEXT
originalTitle TEXT
isAdult BOOLEAN
startYear INTEGER
endYear INTEGER
runtimeMinutes INTEGER
genres TEXT[]
```

Tabellenname: `names`

```
nconst CHAR(10) PRIMARY KEY
primaryName TEXT
birthYear INTEGER
deathYear INTEGER
primaryProfession TEXT[]
knownForTitles CHAR(9)[]
```

Tabellenname: `principals`

```
tconst CHAR(9)
ordering INTEGER
nconst CHAR(10)
category TEXT
job TEXT
characters TEXT
```

[3] `https://www.imdb.com/interfaces/`

Im ersten Schritt muss ein lokaler PostgreSQL-Server auf Ihrem Rechner installiert werden. Erst danach können die Tabellen angelegt und die Datensätze importiert werden. Anleitungen zur Installation von PostgreSQL finden Sie im Internet, bspw. für Debian-basierte Systeme [4]. Sie müssen auch eine Datenbank erstellen, bspw. erstellt der Befehl `createdb mydb` eine Datenbank namens `mydb` (ausgeführt in der Debian-Konsole). Danach können Sie sich per `psql mydb` zu dieser Datenbank verbinden und mit der Aufgabe beginnen.

Für das Erstellen der 3 Tabellen wird eine SQL-Datei `create_db.sql` von uns bereitgestellt (`\i <path-to-create_db.sql>` in der PostgreSQL-Konsole). Um die Daten in ihre lokale PostgreSQL-Datenbank zu importieren, empfehlen wir den `\copy`-Befehl von PostgreSQL zu benutzen. Dieser wird wie folgt per `psql`-Konsole ausgeführt:

```
\copy titles FROM title.basics_no_header_array_format.tsv WITH DELIMITER E'\t'
\copy names FROM name.basics_no_header_array_format.tsv WITH DELIMITER E'\t'
\copy principals FROM title.principals_no_header.tsv WITH DELIMITER E'\t'
```

Nachdem die Datensätze in die Datenbank importiert wurden, können Sie sich die Tabelleneigenschaften mit dem Befehl `\d <tablename>` ansehen. Damit werden die einzelnen Attribute, alle vorhandenen Indizes und eine eventuell vorhandene Sortierung (`CLUSTER`) angezeigt.

Im Zuge dieser Aufgabe werden Sie die folgenden PostgreSQL-Befehle benötigen:

- `EXPLAIN`: Zeigt den Ausführungsplan an *ohne* die nachfolgende Anfrage auszuführen.

  Beispiel:
  `EXPLAIN SELECT * FROM names WHERE birthyear > 1900;`
- `EXPLAIN ANALYZE`: Zeigt den Ausführungsplan an und führt die nachfolgende Anfrage aus.

  Beispiel:
  `EXPLAIN ANALYZE SELECT * FROM names WHERE deathyear > 1900;`
- `CREATE INDEX`: Erstellt einen Index auf einem definierten Attribut einer definierten Tabelle. Es kann auch die Datenstruktur, die für den Index verwendet wird, vorgegeben werden. Wenn ein Index auf einer Tabelle erstellt wird, hat dies keinerlei Auswirkungen auf die Sortierung der betroffenen Tabelle.

  Beispiel:
  `CREATE INDEX names_deathyear_idx ON names(deathyear);`
- `CLUSTER`: Um einen Clustering Index auf einer Tabelle zu erstellen (d.h. die Tabelle physisch zu sortieren), muss zuerst ein Index auf dem entsprechenden Attribut erstellt werden. Danach kann die Tabelle nach diesem Index geclustered werden.

  Beispiel:
  ```
  CREATE INDEX names_deathyear_idx ON names(deathyear);
  CLUSTER names USING names_deathyear_idx;
  ```

[4] `https://wiki.debian.org/PostgreSql`

- `ANALYZE`: Aktualisiert die Statistiken einer definierten Tabelle. Die ist vor allem nach `CLUSTER` wichtig, da die Datenbank sonst nicht weiß, dass eine Umsortierung der betroffenen Tabelle stattgefunden hat.

  Beispiel:
  `ANALYZE names;`

Sie werden drei Anfragen formulieren und optimieren. Alle Fragen basieren auf diesen drei Anfragen (und deren Optimierung) und sind in der Datei `questions.txt` zu finden.

2.1. Anfragen:

1. Geben Sie alle Titel mit einer Laufzeit (`runtimeMinutes`) von 1000 Minuten aus. (Tabelle `titles`)
2. Geben Sie alle Attribute der Tabelle `name` aus, die nach 1968 aber vor 1970 geboren sind (`birthyear`). (Tabelle `names`)
3. Führen Sie einen natürlichen Join auf die Tabellen `names` und `principals` aus. Das Ergebnis soll nur Einträge mit Primärnamen (`primaryName`) 'Harrison Ford' und Geburtsjahr (`birthyear`) 1942 enthalten.

3. Abgabe

Die Abgabe erfolgt über unser Abgabesystem [5]. Beachten Sie, dass nur die jeweils **letzte** Abgabe gewertet wird.

3.1. Aufwandsschätzung

Hier finden Sie eine Zusammenfassung unserer Referenzlösung, erzeugt durch den Befehl `git log --stat=80`. Die letzte Zeile zeigt die Anzahl an eingefügten und gelöschten Zeilen.

Die Referenzlösung stellt nur eine mögliche Lösung dar. Es sind auch andere Lösungen möglich, die sich entsprechend in der Anzahl der Änderungen unterscheiden. Die Zusammenfassung der Referenzlösung dient lediglich zur Orientierung.

```
questions.txt | 171 ++++++++++++++++++++-
1 file changed, 169 insertions(+), 2 deletions(-)
```

4. Zusätzliche Informationen

- \copy-Befehl:
  `https://www.postgresql.org/docs/9.6/app-psql.html#APP-PSQL-META-COMMANDS-COPY`

[5] `https://abgaben.cosy.sbg.ac.at`

- `EXPLAIN`-Befehl:
  `https://www.postgresql.org/docs/9.6/sql-explain.html`
- `ANALYZE`-Befehl:
  `https://www.postgresql.org/docs/9.6/sql-analyze.html`
- `CREATE INDEX`-Befehl:
  `https://www.postgresql.org/docs/9.6/sql-createindex.html`
- `CLUSTER`-Befehl:
  `https://www.postgresql.org/docs/9.6/sql-cluster.html`
- Using `EXPLAIN`:
  `https://www.postgresql.org/docs/9.6/using-explain.html`
- Controlling the Planer with Explicit `JOIN` Clauses:
  `https://www.postgresql.org/docs/9.6/explicit-joins.html`
- Indexes in PostgreSQL:
  `https://www.postgresql.org/docs/9.6/indexes.html`
- Index-Only Scans:
  `https://www.postgresql.org/docs/9.6/indexes-index-only-scans.html`

Anhang A Beispiel

Siehe Datei `example.txt` für Beispielantworten.