Datenbank-Programmierung

Kapitel 2: Benutzung von PostgreSQL

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Sommersemester 2018/19

http://www.informatik.uni-halle.de/~brass/dbp19/

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Den PostgreSQL Datenbank Server starten und stoppen
- SQL-Befehle mittels `psql` an die Datenbank schicken
- `psql`-Skripte schreiben

Inhalt

Geschichte des Systems (1)

- POSTGRES wurde von Michael Stonebraker (+ Team) an der University of California at Berkeley ab 1985 entwickelt.

  Michael Stonebraker hatte zuvor das INGRES-System entwickelt (ebenfalls in Berkeley), eines der beiden ersten relationalen Datenbanksysteme (das andere war System R, Forschungsprototyp von IBM). Michael Stomebraker bekam 2014 den Turing Award, eine Art Nobelpreis für Informatik [https://amturing.acm.org/byyear.cfm].

- Ein Prototyp wurde auf der SIGMOD Konferenz 1988 gezeigt, Version 1 wurde 1989 fertig gestellt.

- POSTGRES war Vorreiter der objektrelationalen Datenbanken.

  Michael Stonebraker gründete zusammen mit anderen die Firma Illustra Information Technologies, die eine kommerzielle Version von POSTGRES herausbrachte. Sie wurde 1997 an Informix verkauft (Informix 2001 von IBM).

Geschichte des Systems (2)

- Das POSTGRES Projekt endete 1994.
- 1994 entwickelten Andrew Yu und Jolly Chen (Studenten in Berkeley) eine Version mit SQL-Unterstützung (statt der Datenbank-Sprache QUEL/POSTQUEL).

  Die erste Version erschien 1995 ("Postgres95"). Die freie Weiterentwicklung war möglich, weil POSTGRES unter einer großzügigen Open Source herausgegeben wurde (MIT Lizenz).
- Der Name "PostgreSQL" wurde 1996 eingeführt.

  Die erste PostgreSQL Release war 6.0, erschienen am 29.01.1997.
- Im Ranking von DB-Engines.com belegt PostgreSQL den vierten Platz (hinter Oracle, MySQL, Microsoft SQL Server).

  [https://db-engines.com/de/ranking]

  Siehe auch [https://www.datanyze.com/market-share/databases].

Verfügbarkeit

- Am 14. Februar 2019 wurden gleichzeitig folgende Versionen veröffentlicht: 11.2, 10.7, 9.6.12, 9.5.16, 9.4.21.

  Die älteren Versionen erhalten im wesentlichen nur Fehlerbehebungen.

  9.4.X wird ab Februar 2020 nicht mehr gepflegt.

- Folgende Plattformen werden mit fertigen Installations-Paketen unterstützt [https://www.postgresql.org/download/]:
  - Linux (Red Hat/CentOS/Fedora, Debian, Ubuntu, SuSE)
  - BSD (FreeBSD, OpenBSD)
  - Windows
  - macOS
  - Solaris

- Der Quellcode ist auch frei verfügbar.

  [https://www.postgresql.org/ftp/source/] [https://git.postgresql.org/gitweb/]

Nutzung von PostgreSQL (1)

- PostgreSQL ist ein Client-Server-System.

  Der Server besteht aus mehreren Prozessen und einem "Shared Memory" Bereich.

- Die Kommunikation zwischen Client und Server erfolgt über eine Netzwerk-Verbindung.

  Wenn Client und Server auf dem gleichen Linux-Rechner laufen, wird ein "UNIX-Domain-Socket" zur Kommunikation verwendet (Inter-Prozess-Kommunikation über temporäre Datei).
  Unter Windows wird aber auch in diesem Fall TCP/IP benutzt.

- Die normale Kommandozeilen-Schnittstelle ist das Programm `psql`.

  Dies ist nur ein Client, der SQL Befehle entgegennimmt und an den Server schickt, dann das Ergebnis abholt und ausgibt. Natürlich gibt es einige Formatierungsmöglichkeiten, Abkürzungen, Einstellungen etc. (s.u.).

Nutzung von PostgreSQL (2)

- Eine bekannte graphische Schnittstelle is pgAdmin.

  [https://www.pgadmin.org/]

- Es ist in erster Linie zur Administration gedacht, erlaubt aber auch, SQL-Anfragen zu stellen.

  Und enthält einen Browser für alle Datenbank-Objekte in einer Baum-Ansicht (Expand/Collapse).

- Es gibt einen Treiber für die JDBC-Schnittstelle zur Kommunikation mit der Datenbank aus Java-Programmen.

  [https://jdbc.postgresql.org/]
  Auch ODBC wird unterstützt: [https://odbc.postgresql.org/]

- Zu den unterstützten Programmiersprachen gehören u.a. Java, C/C++, PHP, Perl, Python.

Inhalt

Starten und Stoppen des Servers (1)

- Wenn man `psql` aufruft, und der Server läuft nicht, bekommt man die Fehlermeldung "`could not connect to server`".
- Man kann auch schauen, ob `postgres` Prozesse laufen.

  Unter Linux z.B.: `ps -ef | fgrep postgres`

  Unter Windows mit dem Taskmanager (Ctrl+Alt+Delete).
- Man kann server-basierte Datenbanken so installieren,
  - dass der Server beim Start des Betriebssystems automatisch mit gestartet wird, oder

    Bequem, aber das Hochfahren des Betriebssystems dauert etwas länger, und einige Systemresourcen sind belegt, auch wenn man die Datenbank nicht nutzt.
  - dass man ihn manuell starten muss (bei Bedarf).

Starten und Stoppen des Servers (2)

- Bei vielen Datenbanken ist es sehr wichtig, sie vor dem Shutdown des Betriebssystems ordnungsgemäß herunterzufahren.

  Sonst kann beim nächsten Hochfahren ein Recovery nötig sein, weil nicht alle veränderten Hauptspeicherinhalte geschrieben wurden. Dabei können die Daten aller beendeten Transaktionen wieder hergestellt werden (wenn alles richtig funktioniert), aber das Hochfahren dauert dann wesentlich länger.

- Linux schickt vor dem endgültigen Shutdown allen Prozessen ein `SIGTERM` Signal, und der PostgreSQL Server reagiert darauf.

  Allerdings erst, wenn sich alle laufenden Sitzungen beendet haben. Eventuell könnte das dem Betriebssystem zu lange dauern.

Starten und Stoppen des Servers (3)

- Das Programm `pg_ctl` ("postgres control") dient u.a. zum Starten und Stoppen des Servers.
- Es kann nur vom Administrator verwendet werden, d.h. unter Linux muss man der Nutzer `postgres` sein.
- Dann kann man eventuell einfach "`pg_ctl start`" eingeben.

  Wenn sich die Daten bei `/var/lib/pgsql/data` befinden oder die Umgebungsvariable `PGDATA` entsprechend gesetzt ist.
- Mit Nutzerwechsel und mehr Parametern sieht es so aus:

```
sudo -u postgres \
pg_ctl -D /data/pg -l /data/logfile -w start
```

  Mit `-D` wird das Hauptverzeichnis von Postgres angegeben, mit `-l` eine Logdatei für Fehlermeldungen (sonst Terminal, von dem `pg_ctl` aufgerufen). Die Option `-w` bedeutet, dass `pg_ctl` wartet, bis der Server gestartet ist.

Starten und Stoppen des Servers (4)

- Entsprechend kann man den Server geordnet herunterfahren mit "`pg_ctl stop`".
- Mit Nutzerwechsel und mehr Parametern:

```
sudo -u postgres \
pg_ctl -D /data/pg -m fast -w stop
```

Mit "`-m fast`" ("shutdown mode fast") kann man verlangen, dass eventuell laufende Sitzungen abgebrochen werden.
Der Default ("`-m smart`") ist, zu warten, bis sich alle laufenden Sitzungen freiwillig beenden.
Die dritte Möglichkeit ist "`-m immediate`", das würde die Prozesse sofort beenden, und damit aber ein Recovery beim nächsten Start nötig machen, weil ein eventuell veränderter Hauptspeicher-Inhalt nicht mehr gesichert wird.

- "`pg_ctl status`" zeigt an, ob der Server aktuell läuft.

Datenbanken (1)

- Ein “Database Cluster” in PostgreSQL besteht aus mehreren Datenbanken, die von einer Instanz des Datenbank-Servers verwaltet wird.

  Bei anderen Datenbanksystemen wäre ein Cluster eher eine Gruppe von Datenbank-Servern (mehrere Rechner, um Performance und Zuverlässigkeit zu steigern). Die GUI “`pgAdmin`” zeigt “Servers” an, darunter “Databases”.

- Ein “Database Cluster” entspricht einem Verzeichnis im Dateisystem, in dem alle Datenbanken gespeichert sind.

  Z.B. `/usr/local/pgsql/data`, `/var/lib/pgsql/data`,
  `C:\Program Files (x86)\PostgreSQL\9.5\data`
  Wenn man `psql` als Nutzer `postgres` startet (“`sudo -u postgres psql`”), kann das Verzeichnis mit “`show data_directory;`” angezeigt werden.
  Unter Linux findet man es auch in “`ps -ef | fgrep postgres`” als Wert der Option `-D` des ersten Postgres Prozesses (“Postmaster”).

Datenbanken (2)

- Nachdem ein DB-Cluster angelegt wurde (mit `initdb` im Rahmen der Installation), werden darin drei Datenbanken angelegt: `postgres`, `template0` und `template1`.

  "`SELECT datname from pg_database;`" oder "`\l`" in `psql`.

  Die Datenbank `postgres` gibt es ganz von Anfang an, man kann sich mit ihr verbinden, um andere Datenbanken anzulegen (oder wenn man sonst keinen Datenbank-Namen kennt). Sie scheint im wesentlichen leer zu sein.

- Nutzer-Datenbanken werden normalerweise durch Clonen von `template1` angelegt.

  Man kann die Template-Datenbank aber auch explizit angeben:

  ```
  CREATE DATABASE dbname TEMPLATE template0;
  ```

  Es gibt zwei Template-Datenbanken, weil man `template1` lokal modifizieren kann (z.B. würden dort angelegte Tabellen automatisch mitkopiert), aber `template0` das Original bleiben soll. Direkt nach der Installation sind beide Template-Datenbanken gleich.

Datenbanken (3)

- Am einfachsten heisst die Datenbank wie der Betriebssystem-Nutzer.

  Wenn man die Kommandozeilen-Schnittstelle `psql` ohne explizite Angabe von Benutzer und Datenbank aufruft, wird der Benutzername des aktuellen Betriebssystem-Nutzers für beide Angaben eingesetzt.
  Wenn mein Login also `brass` ist, wäre es gut, einen Datenbank-Nutzer `brass` anzulegen, dem die Datenbank `brass` gehört.

- Wenn man PostgreSQL frisch installiert hat, gibt es natürlich noch keine Datenbank für bestimmte Nutzer.

- Man kann dann zuerst den Betriebssystem-Nutzer als Datenbank-Nutzer mit dem Programm `create_role` anlegen, anschliessend die Datenbank mit `createdb`.

  Beides geht auch direkt mit SQL-Befehlen, siehe nächste Folie.

Datenbanken (4)

- Nutzer und Datenbank unter Linux mit SQL anlegen:
  - Benutzer "`postgres`" werden:
    ```
    sudo -u postgres -i
    ```
  - Kommandoschnittstelle aufrufen:
    ```
    psql
    ```
  - Nutzer anlegen:
    ```
    CREATE ROLE brass LOGIN CREATEDB;
    ```
    Das `CREATEDB` Recht wäre nicht nötig.
  - Datenbank anlegen:
    ```
    CREATE DATABASE brass OWNER brass
           ENCODING 'UTF8';
    ```
  - `psql` verlassen:
    ```
    \q
    ```

Tablespaces

- Ein Tablespace ist ein Verzeichnis auf dem Server, in dem Tabellen gespeichert werden (physischer Container).
- Am Anfang hat eine PostgreSQL Installation zwei Tablespaces:
  - `pg_default`: Für alle normalen Tabellen.
  - `pg_global`: Tabellen, die zu mehreren Datenbanken gehören.

    Z.B. die Liste aller Datenbanken: `pg_database`.
- Man kann zusätzliche Tablespaces anlegen.

  ```
  CREATE TABLESPACE space1 LOCATION '/mnt/sda1/postgresql/data';
  ```

  Das ist eher etwas für Experten und verkompliziert spätere Backups.
- Man kann eine Tabelle einem Tablespace zuordnen:
  `CREATE TABLE r(a int) TABLESPACE space1;`

Schemata (1)

- Eine Datenbank kann mehrere Schemata enthalten.
- Jede neu angelegte Datenbank enthält ein Schema "`public`".

  Jeder, der sich mit der Datenbank verbinden kann, kann dieses Schema nutzen.

  Man kann das Schema mit "`DROP SCHEMA public`" löschen, wenn gewünscht.
- Man kann ein Schema anlegen mit

  ```
  CREATE SCHEMA name;
  ```

  Es gibt auch eine Variante, bei dem man das Schema einem Nutzer als Besitzer zuordnet: "`CREATE SCHEMA name AUTHORIZATION nutzer;`".
- Man kann eine Tabelle in einem Schema mit folgender Notation ansprechen: "`Schema.Tabelle`", z.B.:

  ```
  SELECT * FROM public.STUDENTEN;
  ```

Schemata (2)

- Es gibt einen Suchpfad für Datenbank-Schemata. Dieser wird benutzt, wenn ein Schema nicht explizit angegeben ist:

```
SELECT * FROM STUDENTEN;
```

- Man kann den Suchpfad in `psql` anzeigen mit

```
SHOW search_path;
```

- Der Default ist: `"$user",public`.
- Das erste existierende Schema wird benutzt, um Tabellen anzulegen, für die nicht explizit ein Schema angegeben ist.

  Solange es kein Schema mit dem Namen des Nutzers gibt, wird also das `public` Schema verwendet.

- Der Suchpfad kann mit folgendem Befehl verändert werden:

```
SET search_path TO myschema,public;
```

Schemata (3)

- Man kann eine Tabelle in einem Schema anlegen mit:

```
CREATE TABLE schema.name ( ... );
```

- Durch den Suchpfad und das `public`-Schema braucht man sich nicht unbedingt mit Schemata zu befassen.

  Allerdings zeigen Werkzeuge wie `pgAdmin` die Schemata an, auch bei Anfragen an den Systemkatalog wird man öfters Schemata sehen. Deswegen ist es gut, das Konzept zu kennen.

- Eine mögliche Struktur ist, dass es nur eine Datenbank gibt, und darin ein Schema pro Nutzer.

  Das Schema heisst so wie der Nutzer. Man kann in PostgreSQL keine datenbank-übergreifenden Anfragen stellen, wohl aber Anfragen an Tabellen verschiedener Schemata.

Schemata (4)

- `\dn` in `psql` listet alle Schemata.

  Das "`n`" steht wohl für "Namespace".

- Jede Datenbank hat ein Schema `pg_catalog` für die System-Tabellen ("Data Dictionary").

- Dieses Schema ist implizit ganz vorn in jedem Suchpfad.

  Sofern es nicht explizit enthalten ist. Bei bedarf kann man es also weiter hinten einfügen.

- Die dort enthaltenen Tabellen beginnen mit "`pg_`".

  Man sollte solche Namen für eigene Tabellen vermeiden.

- Beispiel: `pg_catalog.pg_tables`.

Inhalt

Verbindung zum Server

- Wenn man nicht explizit einen Server angibt, verbindet sich `psql` über einen Unix-domain socket oder TCP/IP zu `localhost` zu einem Server auf dem lokalen Rechner.
- Der Port ist dabei typischerweise 5432.

  Die Default Portnummer kann kann vor dem Compilieren konfiguriert werden.
- In `psql` zeigt `\conninfo` die Verbindung zum Server an.
- ...

Nutzung: Das Wichtigste

- Man kann SQL-Befehle über mehrere Zeilen verteilt eingeben und muss sie dann mit einem Semikolon “;” abschließen.
- Außerdem gibt es Befehle, die von `psql` selbst ausgeführt werden. Sie beginnen mit einem Rückwärts-Schrägstrich “\” und werden direkt ausgeführt, sobald Enter gedrückt wird.
- `\h`: Hilfe zu SQL-Anweisungen.
- `\?`: Hilfe zu `psql`-Befehlen.
- `\d`: Liste aller Tabellen/Sichten etc.
- `\d TAB`: Schema der Tabelle `TAB` anzeigen.
- `\q`: `psql` beenden (“quit”).

Literatur/Quellen

- Wikipedia: PostgreSQL (englisch, deutsch)
  [https://en.wikipedia.org/wiki/PostgreSQL]
  [https://de.wikipedia.org/wiki/PostgreSQL]
- Homepage des Projekts
  [https://www.postgresql.org/]
- Deutsche PostgreSQL Homepage
  [http://www.postgres.de/]
- PostgreSQL Dokumentation: `psql`
  [https://www.postgresql.org/docs/9.5/app-psql.html]
- PostgreSQL Tutorial:
  [http://www.postgresqltutorial.com/]
- pgAdmin (GUI zur Administration, auch SQL-Zugang)
  [https://www.pgadmin.org/]
- Rachid Belaid: Introduction to PostgreSQL physical storage.
  [http://rachbelaid.com/introduction-to-postgres-physical-storage/]