Inhalt

[1. Erwartungen 2](#_Toc498088915)

[2. Admin-Tools 2](#_Toc498088916)

[3. Allgemeines 2](#_Toc498088917)

[4. Setup 2](#_Toc498088918)

[4.1. Debian Installation 2](#_Toc498088919)

[4.2. Pfade in Linux 2](#_Toc498088920)

[4.3. Konfiguration von postgresql.conf 3](#_Toc498088921)

[4.3.1. Ressourcen 3](#_Toc498088922)

[4.3.2. Verhalten 3](#_Toc498088923)

[4.4. Konfiguration von pg\_hba.conf 3](#_Toc498088924)

[5. Tools 3](#_Toc498088925)

[5.1. Postgresql-Konsole 3](#_Toc498088926)

[5.2. Benchmarking 3](#_Toc498088927)

[5.2.1. Prozesse 3](#_Toc498088928)

[5.3. pg\_dump/pg\_restore 3](#_Toc498088929)

[5.3.1. Migration 3](#_Toc498088930)

[5.3.2. WAL-Archivierung (inkrementelles Backup) 4](#_Toc498088931)

[5.3.3. WAL-Archivierung wiederherstellen 4](#_Toc498088932)

[5.4. Replikation 4](#_Toc498088933)

[5.5. Wartung 5](#_Toc498088934)

[5.5.1. Vacuum 5](#_Toc498088935)

[5.5.2. Index 5](#_Toc498088936)

[5.5.3. Weiteres 5](#_Toc498088937)

[6. Features 5](#_Toc498088938)

[6.1. ANSI-SQL:2008 5](#_Toc498088939)

[6.2. Objektorientierung 5](#_Toc498088940)

[6.3. Similarity 6](#_Toc498088941)

[6.4. Indizes 6](#_Toc498088942)

[6.5. Procedural Language (Stored Procedures, Triggers) 6](#_Toc498088943)

[7. ToDos 6](#_Toc498088944)

[7.1. Datenbank wechseln 7](#_Toc498088945)

[7.2. Encoding 7](#_Toc498088946)

[7.3. Zugriffsrechte 7](#_Toc498088947)

[7.4. max\_connections 7](#_Toc498088948)

[7.5. Query tuning 7](#_Toc498088949)

[7.6. Public Schema 7](#_Toc498088950)

[7.7. Inkrementelles Backup 7](#_Toc498088951)

[7.8. Lobs 7](#_Toc498088952)

# Admin-Tools

* DBeaver (Autovervollständigung)
* Squirrel (Autovervollständigung)
* pgAdmin 3/4 (am vollständigsten; Autovervollständigung)
  + pgAdmin 4 seit v2.0 deutlich stabiler
* Konsole (PSQL) vollständiger als pgAdmin
* <https://www.navicat.com/en/products/navicat-for-postgresql> (Rich Text Editor with autocomplete, 200$))

# Allgemeines

Früher hieß die Binary von postgresql „postmaster“. Das wurde zunächst zu postgres umbenannt. Als es ANSI-SQL konform wurde, wurde es umbenannt zu postgresql.

# Setup

## Debian Installation

sudo apt-get install postgresql postgresql-client

sudo su postgres

echo "export PGDATA='/var/lib/postgresql/9.6/main'" >> ~/.bashrc

/etc/init.d/postgresql start[[1]](#footnote-1)

## Pfade in Linux

|  |  |
| --- | --- |
| /var/lib/postgresql/9.6/main | Haupt-Verzeichnis (a.k.a. data) |
| /var/lib/postgresql/9.6/main/base | Datenbanken |
| /etc/postgresql/9.6/main  /var/lib/postgresql/9.6/main | Konfig pg\_hba.conf, postgresql.conf |
| /usr/lib/postgresql/9.6/bin | Bin |
| /usr/lib/postgresql/9.6/lib | Lib, e.g. extensions |
| /var/log/postgresql | Log |

Das Hauptverzeichnis lautet ~/<version>/main (in Multi-Installationen, falls z.B. mit apt-get install installiert wurde) bzw. ~/data (in Einfach-Installationen, falls z.B. ein postgres docker vorliegt).

## Konfiguration von postgresql.conf

Nach jeder Änderung ist ein Neustart des Dienstes nötig.

### Ressourcen

shared\_buffers: muss kleiner sein als RAM (RDS: shared\_buffers = RAM/4), siehe

<https://github.com/jollygoodcode/jollygoodcode.github.io/issues/16>

### Verhalten

fsync = off: 4x schneller, aber Datenverlust bei Ausfall der Festplatte (d.h. Transaktion wird beendet, nachdem der Schreibauftrag an das OS übergeben wurde – und nicht, nachdem geschrieben wurde). Risiko kann durch eine USV abgefangen werden.

## Konfiguration von pg\_hba.conf

Nach jeder Änderung ist ein reload nötig.

pgAdmin: Im Editor für Konfigurationen > Datei > Serverkonfiguration laden

Debian: /etc/init.d/postgresql reload

# Tools

## Postgresql-Konsole

psql

## Benchmarking

pgbench -i -s 50 training

Legt 50.000.000 Datensätze in 4 Tabellen in der Datenbank training an.

pgbench -c 10 -t 100000 training

Führt einen Benchmark aus.

### Prozesse

top/htop

* Prozesse **COMMIT** oder **UPDATE** können problemlos gekillt werden.
* **Niemals** einen **WRITER**-Prozess beenden!

## pg\_dump/pg\_restore

Wenn für den dump das Format custom, tar oder dir gewählt wird, kann man aus dem dump strukturierte Infos beim restore zurückgewinnen – nicht aber im plain text format. Z.B. nur bestimmte Tabellen.

### Migration

Migration von Datenbank A nach Datenbank B:

pg\_dumpall <params> | psql <params>

### WAL-Archivierung (inkrementelles Backup)

WAL: Write Ahead Log

1. Verzeichnis für backups erstellen, mit zwei Unterverzeichnissen (z.B. base und pg\_xlog\_archive[[2]](#footnote-2)). Rechte für postgres müssen gesetzt sein!
2. postgresql.conf:

archive\_mode = on

archive\_command = 'cp %p /var/lib/postgresql/backup/pg\_xlog\_archive/%f'

wal\_level = replica

1. Dienst neu starten

/etc/init.d/postgresql start

1. Testen

pgbench -c 10 -t 1000 training

1. Base sichern

Kann im laufenden Modus durchgeführt werden. Stellt dabei sicher, dass alle schreibenden Zugriffe ins WAL erfolgen, dadurch kann das vollständige Backup erzeugt werden.

1. pg\_hba.conf

local replication postgres peer

host replication postgres 127.0.0.1/32 md5

1. postgresql.conf

max\_wal\_senders = 1

1. Postgres neustarten
2. pg\_basebackup -D ~/backup/base -l testlabel -P[[3]](#footnote-3)
3. pg\_archivecleanup ~/backup/pg\_xlog\_archive <gerade erzeugte xxx.backup>

Param1: Verzeichnis, in dem aufgeräumt wird / Param2: alles bis dahin wird aufgeräumt

### WAL-Archivierung wiederherstellen

1. Kaputtes/altes data-Verzeichnis umbenennen
2. ~/backup/base als neues data-Verzeichnis kopieren
3. cp /usr/share/postgresql/9.6/recovery.conf.sample /var/lib/postgresql/9.6/main/recovery.conf
4. nano recovery.conf

restore\_command = 'cp /var/lib/postgresql/backup/pg\_xlog\_archive/%f %p'

recovery\_target\_time = '<date time> <timezone>'

Dabei Zeitpunkt des wiederherzustellenden Standes wählen und unbedingt auf die korrekte Zeitzone achten!

## Replikation

*Achtung: Replikation und inkrementelles Backup gehen nicht gleichzeitig! Es ist aber möglich, diese zu verketten, z.B. repliziert der master auf den slave, und der slave macht ein inkrementelles Backup.*

Anleitung siehe Kapitel 12 in PostgeSQL Administration

**Zusätzliche Einstellungen in pg\_hba.con**f

wal\_level = replica

synchronous\_standby\_names = '\*'

max\_replication\_slots = 5

**Zuletzt den Slave erstellen**

psql -c "SELECT \* FROM pg\_create\_physical\_replication\_slot('sklave\_one');"

**Wieder ausschalten**

psql -c "SELECT \* FROM pg\_drop\_replication\_slot('sklave\_one');"

## Wartung

|  |  |
| --- | --- |
| **Linux** | **SQL** |
| vacuumdb | vacuum |
| reindexdb | reindex |
| clusterdb | cluster |

**Hilfe**

Linux: <Linuxbefehl> --help

SQL: \h <SQL-Befehl>

### Vacuum

* Defragmentiert
* Es wird nicht viel zusätzlicher Speicherplatz benötigt – bei fast voller Festplatte kann das helfen
* Wird standardmäßig im Hintergrund ausgeführt
* Lässt sich in der postgresql.conf konfigurieren

### Index

* Repariert kaputte Indizes
* ~~Balanciert nicht mehr performante Indizes (Index-Bäume können einseitig-lastig werden) 🡪 Falsch: postgres nutzt B-Trees, die immer balanciert sind~~
* Beim Aufruf wird zweiter Index temporär daneben gelegt

### Weiteres

<https://www.postgresql.org/docs/9.6/static/functions-admin.html>

Es macht Sinn, Funktionen daraus im Frontend anzubieten. Beispiel:

SELECT \* FROM pg\_database\_size('db\_name');

## Scalability

<https://www.javaworld.com/article/3244345/open-source-tools/why-old-school-postgresql-is-so-hip-again.amp.html>

# Features

## ANSI-SQL:2008

Postgresql ist dem ANSI-Standard von allen Dbs am nächsten. Allerdings ergänzt Postgresql auch Konstukte, wie z.B. der Umgang mit Objekten: Postgresql ist objekt-relational, ANSI definiert relationale Dbs. Zudem ist für jeden Befehl dokumentiert, ob er ANSI-konform ist.

## Objektorientierung

Postgres ist objekt-relational und kann Daten relational, objektorientiert und kombiniert speichern. Objekt-Datentypen sind:

* Arrays
* Komplexe
* XML
* JSON/JSONB
* HashMaps

## Similarity

-- extensions müssen ins Schema public installiert werden

CREATE EXTENSION fuzzystrmatch;

SELECT \* FROM schema.tabelle WHERE levenshtein(feld, 'tar') <= 2;

## Indizes

<https://www.postgresql.org/docs/9.5/static/sql-createindex.html>

|  |  |  |
| --- | --- | --- |
| **Vorhandene Indizes** | **Abfrage** | **Gewählter Index** |
| {x}, {y} | WHERE x = $1 | {x} |
| {x}, {y} | WHERE x = $1 AND y = $2 | {x} oder {y} |
| {x}, {x,y} | WHERE x = $1 AND y = $2 | {x,y} mit hoher Wahrscheinlichkeit |
| {x,y} | WHERE x = $1 | {x,y} |
| {x,y} | WHERE y = $1 | keinen! |
| {x}, {y} | WHERE x = $1 OR y = $2 | schwer zu optimieren (wg. OR) |

* Fall nicht eindeutig, sucht sich der Query-Optimizer den statistisch besten.
* Falls kein Index passt, wird ein full Tablescan durchgeführt.
* EXPLAIN gibt aus, welche Indizes gewählt werden.
* Ein Index kann einem Tablespace zugeordnet werden.

**Datum**

Für Indizes auf einem Datums-Feld können Abfragen derart gestaltet werden:

WHERE date\_part('yyyy', datum) = 1997

WHERE datum BETWEEN '1997-01-01' AND '1997-12-31'

**Index Definition**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Methode** | **multicolumn** | **unique** | **Laufzeit** | **Optimal für** |
| B-tree (default) | **X** | **X** | O(log(n)) | Ranges (z.B. <, >, <=, >=) |
| hash[[4]](#footnote-4) |  |  | O(1) | Gleichheit |
| GiST | **X** |  |  | "nearest-neighbor" searches |
| SP-GiST |  |  |  |  |
| GIN | **X** |  |  |  |
| BRIN | **X** |  |  |  |

**Partial Indexes**

CREATE INDEX … **WHERE …**

Erzeugt einen Index über einen Teil der Rows.

## Procedural Language (Stored Procedures, Triggers)

Per CREATE LANGUAGE können weitere Sprachen (Java, Python, …) installiert werden.

Nachteile:

* debugging?
* Java 🡪 langsamer wg. VM

## Conditional Triggers

CREATE TRIGGER name ON table\_name **WHEN ( condition )** EXECUTE PROCEDURE function\_name;

## Common Table Expressions

* With Clause
* Temporary Tables, kind of helper functions
* Ad-hoc view
* Lesbarkeit: Prozedural, weniger Klammern
* Rekursion

WITH A AS (

SELECT …

),

B AS (

SELECT ….

)

SELECT A.y, z.B

FROM A

INNER JOIN B ON A.x = B.x

WITH RECURSIVE monate AS (

SELECT 1 AS Monat

UNION ALL

SELECT Monat+1 FROM monate

WHERE Monat < 12

)

SELECT \*

FROM monate;

WITH RECURSIVE cat AS (

SELECT \*,id::text as child\_path from schema.category WHERE id = '<id>'

UNION ALL

SELECT parent.\*,concat(parent.id, '/', child\_path) as child\_path FROM schema.category AS parent

INNER JOIN cat ON parent.tid = cat.parent\_tid

)

SELECT \* FROM cat

INNER JOIN schema.category\_attribute\_values AS attr\_val ON cat.tid = attr\_val.category\_tid

WHERE attr\_val.locale = 'de';

## Windowing Functions

Zur Analyse, Auswertung und Reporting.

Optional Kombination mit CTE (nicht notwendig). **Fett**: Windowing (s.u.).

WITH occurences AS (

SELECT d.id as diagram\_id, n.entity\_tid, count(n.tid) as anzahl FROM schema.node n

INNER JOIN schema.diagram d ON d.tid = n.diagram\_tid

INNER JOIN schema.entity e ON e.tid = n.entity\_tid

GROUP BY d.tid, n.entity\_tid

),

zwischenergebnis AS (

SELECT \*, **sum(anzahl) OVER (PARTITION BY diagram\_id)** as number\_of\_occurences, **count(anzahl) OVER (PARTITION BY diagram\_id)** as number\_of\_entities FROM occurences

)

SELECT \* FROM zwischenergebnis

WHERE number\_of\_occurences > 30;

# ToDos

## Datenbank wechseln

postgres ist die Wartungsdatenbank – nur für Wartungszwecke geeignet.

Z.B. kann keine Kopie von der Db erstellt werden, in der man eingeloggt ist.

🡪 Datenbank umbenennen von postgres in einen domänenspezifischen Namen.

## Encoding

Encoding auf der Db sollte aus Performance-Gründen dasselbe sein wie das Encoding unserer Software (Hibernate/Spring Repositories) 🡪 sonst wird immer umgerechnet.

* Hibernate/Spring verwenden UTF-16?
* Unsere Dbs sind derzeit auf en\_US.UTF-8 eingestellt
  + besser: de\_DE.UTF-16

## Zugriffsrechte

postgresql.conf > listen\_addresses

## max\_connections

Auf RDS überprüfen

## Query tuning

Verhältnis zwischen seq\_page\_cost und random\_page\_cost gemäß den Werten der zugrundeliegenden Festplatte/SSD anpassen. Für eine SSD ist das 20 Jahre alte Verhältnis ¼ definitiv nicht mehr passend.

## Public Schema

Anstelle des Löschens des public Schemas könnte alternativ in der postgresql.conf die Variable search\_path so gesetzt werden, dass nicht im public gesucht wird (und damit auch nichts dort erzeugt wird).

* testen, ob Skripte mit fehlendem Schema ausfehlern (z.B. anlegen von Triggern)
* DB anpassen
* Konfiguration für testcontainers anpassen

Extensions müssen im public Schema installiert werden!

## Inkrementelles Backup

Siehe Kapitel

Fehleranfällig – im Pairing!

Wie geht es für RDS?

## Lobs

Varianten

1. Falls ein externes Objekt verwendet wird, muss anstelle eines text Feldes ein oid Feld verwendet werden.

<https://www.postgresql.org/docs/9.5/static/largeobjects.html>

<https://www.postgresql.org/docs/9.5/static/lo-funcs.html>

Es gibt Stackoverflow-Artikel zum Problem, dass Hibernate @Lob für postgres falsch speichert.

Google: „hibernate blob bytea“

1. Verwendung von Blob (bytea).
2. **Verwendung von text (unlimitiert) oder varchar (limit selbst definierbar). 🡨 beste Lösung**

## Fehlende Indizes

z.B. Tabelle association: kein Index auf tid

## Sonstiges

### Postgres docker image

* nano (und ein paar andere Tools) hinzufügen
* data dir mounten

# PSQL

Siehe auch <https://www.postgresql.org/docs/9.6/static/app-psql.html>, Sektion Meta-Commands.

|  |  |
| --- | --- |
| Hilfe anzeigen | \h [<command>] |
| Verlassen | \q |
| Datenbank wählen | \c <database> |
| Schemas anzeigen | SELECT schema\_name FROM information\_schema.schemata; |
| Search\_path anzeigen | SHOW search\_path; |
| Search\_path wechseln | SET search\_path TO <schema>; |
| Alles anzeigen | \d |
| Tabellen anzeigen | \dt |

1. Alternative: /usr/lib/postgresql/9.6/bin/pg\_ctl -D <data-dir> [↑](#footnote-ref-1)
2. pg\_xlog ist ein anderer Begriff für WAL, x steht bei postgres für Transaktion. Ab Version 10 wird der Begriff WAL einheitlich verwendet. [↑](#footnote-ref-2)
3. Alternative: i) SELECT pg\_start\_backup() ii) cp … iii) SELECT pg\_start\_backup() [↑](#footnote-ref-3)
4. „Caution: Hash index operations are not presently WAL-logged, so hash indexes might need to be rebuilt with REINDEX after a database crash if there were unwritten changes. Also, changes to hash indexes are not replicated over streaming or file-based replication after the initial base backup, so they give wrong answers to queries that subsequently use them. For these reasons, hash index use is presently discouraged.“ [↑](#footnote-ref-4)