Was ist neu in PostgreSQL 8.5

CLT 2010 – 13.-14.03.2010

Andreas 'ads' Scherbaum

Web: http://andreas.scherbaum.la/
E-Mail: andreas[at]scherbaum.biz
PGP: 9F67 73D3 43AA B30E CA8F 56E5 3002 8D24 4813 B5FE

13.-14.03.2010

Was ist PostgreSQL?

- Relationale Datenbank unter BSD Lizenz
- Weltweit aktive Community
- Zahlreiche Features und Funktionen (Foreign Keys, Transaktionen, Trigger)
- Läuft auf zahlreichen Betriebssystemen und diverser Hardware
- Weitgehendes Einhalten der SQL-Standards – Dokumentation der Abweichungen
- Im Schnitt pro Jahr ein Release mit neuen Features
- Version 9.0 wird derzeit fleißig entwickelt

Der Dozent

- Name: Andreas Scherbaum
- Selbstständig im Bereich Datenbanken, Linux auf Kleingeräten, Entwicklung von Webanwendungen
- Arbeit mit Datenbanken seit 1997, mit PostgreSQL seit 1998 oder 1999
- Gründungsmitglied der Deutschen und der Europäischen PostgreSQL User Group
- Board of Directors – European PostgreSQL User Group

OSI-approved License

- Die von PostgreSQL verwendete Lizenz ist BSD-artig
- Wurde bisher aber nicht als solche anerkannt
- Mittlerweile von OSI akzeptiert

Named Arguments

- Es ist möglich, die Parameter einer Stored Procedure explizit zu benennen

Beispiel (Stored Procedure)

```
CREATE OR REPLACE FUNCTION my_substr (
  IN data TEXT,
  IN d_start INTEGER DEFAULT 1,
  IN d_length INTEGER DEFAULT 1,
  OUT result TEXT)
AS $$ BEGIN

  result := SUBSTR(data, d_start, d_length);

END; $$ LANGUAGE 'plpgsql' STRICT;
```

Named Arguments

Beispiel (Stored Procedure aufrufen)

```
SELECT my_substr(4 AS d_length, 'PostgreSQL' AS data);
 my_substr
-----------
 Post
(1 Zeile)
```

- Parameter vertauscht
- Parameter in der Mitte (2. Parameter) ausgelassen
- Funktioniert mit allen Sprachen, die benannte Parameter unterstützen

DO Statements

- Mit Hilfe von DO kann Code ohne eine zugehörige Funktion ausgeführt werden

Beispiel (DO)

```
DO $$
DECLARE
...
BEGIN
...
END;
$$ LANGUAGE 'plpgsql';
```

DO Statements

- Liefert keine Ergebnisse zurück (wird als Funktion ohne Argumente und ohne Rückgabewert ausgeführt)
- `LANGUAGE 'plpgsql'` ist optional, wenn `default_do_language` gesetzt ist

CURSOR Operationen in `pl/pgSQL`

- `MOVE FORWARD n`: springt `n` Einträge vorwärts
- `MOVE FORWARD ALL`: springt hinter den letzten Eintrag im Cursor
- `MOVE BACKWARD n`: springt `n` Einträge rückwärts
- `MOVE BACKWARD ALL`: springt zum ersten Eintrag im Cursor

Per User/Database GUC (Grand Unified Configuration)

- Es ist jetzt möglich, Settings per Nutzer oder per Datenbank festzulegen

Beispiel (Per User GUC)

```
ALTER ROLE import SET client_encoding = 'latin1';
```

Beispiel (Per Database GUC)

```
ALTER DATABASE stats
  SET search_path = 'statistics, public';
```


Per User und Database GUC

Beispiel (Per User GUC)

```
ALTER ROLE import IN DATABASE stats
  SET client_encoding = 'latin1';
```

Beispiel (Per User GUC)

```
ALTER ROLE web23 IN DATABASE blogs
  SET search_path = 'web23, auth, public';
```

- Quasi-Konfiguration für Programme möglich, die Schemata nicht unterstützen
- \drds Befehl in `psql` zeigt die Settings an

GRANT ALL

- Es ist möglich, die Rechte an allen Objekten mit einem Befehl zu ändern

Beispiel (alle Rechte ändern)

```
GRANT ALL ON ALL TABLES IN SCHEMA public TO web23;
REVOKE ALL ON TABLE users FROM web23;
```


GRANT ALL

- Es ist möglich, die Rechte für zukünftig erstellte Objekte anzugeben

Beispiel (Default Rechte setzen)

```
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT ALL ON TABLES TO web23, postgres;
```

- \ddp zeigt die Defaultrechte in `psql` an
- Funktioniert auch für Sequenzen, Funktionen ect.

pg_hba.conf: samehost und samenet

- `samehost` entspricht allen IP-Adressen des Servers
- `samenet` entspricht allen direkt erreichbaren IP-Adressen

Beispiel (samehost / samenet)

```
eth0      Link encap:Ethernet  HWaddr ce:04:a6:a0:42:be
          inet addr:192.168.30.1
          Bcast:192.168.30.255  Mask:255.255.255.0

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
```

- `samehost`: 192.168.30.1, 127.0.0.1
- `samenet`: 192.168.30.0 - 192.168.30.255, 127.0.0.0 - 127.255.255.255

Konfiguration gegen einen RADIUS-Server

- Die Nutzung eines Radius-Servers zur Authentifizierung ist jetzt möglich

maschinenlesbare EXPLAIN Ausgabe

- die Ausgabe von EXPLAIN ist in verschiedenen Formaten möglich

Beispiel (neue Syntax)

```
EXPLAIN ( costs off ) SELECT ...
EXPLAIN ( analyze on ) SELECT ...
EXPLAIN ( analyze on, costs off ) SELECT ...
```

Beispiel (Ausgabe formatieren)

```
EXPLAIN ( analyze on, format xml ) SELECT ...
EXPLAIN ( analyze on, format json ) SELECT ...
```


Wert bei Unique-Key Verletzung wird ausgegeben

Beispiel (Beispieltabelle)

```
CREATE TABLE pkey_test ( data INT4 PRIMARY KEY );
```

Beispiel (Unique-Key Verletzung)

```
INSERT INTO pkey_test (data) VALUES (4);
INSERT INTO pkey_test (data) VALUES (5);
INSERT INTO pkey_test (data) VALUES (4);
ERROR: duplicate key value violates
       unique constraint "pkey_test_pkey"
DETAIL: Key (data)=(4) already exists.
```

Deferrable Uniqueness Constraints

Beispiel (Beispieltabelle)

```
CREATE TABLE pkey_test ( data INT4 PRIMARY KEY );
INSERT INTO pkey_test (data) VALUES (1), (2), (3);
```

Beispiel (Werte verändern - bisheriges Verhalten)

```
UPDATE pkey_test SET data = data + 1;
ERROR: duplicate key value violates
       unique constraint "pkey_test_pkey"
DETAIL: Key (data)=(3) already exists.
```


Deferrable Uniqueness Constraints

Beispiel (Beispieltabelle)

```
CREATE TABLE pkey_test ( data INT4 PRIMARY KEY
                         DEFERRABLE INITIALLY DEFERRED);
INSERT INTO pkey_test (data) VALUES (1), (2), (3);
```

Beispiel (Werte verändern - neues Verhalten)

```
UPDATE pkey_test SET data = data + 1;
```

DROP IF EXISTS: Spalten und Constraints

Beispiel (DROP IF EXISTS)

```
ALTER TABLE users DROP COLUMN IF EXISTS email2;
```

Beispiel (DROP IF EXISTS)

```
ALTER TABLE users DROP CONSTRAINT IF EXISTS id_pkey;
```


SQL-State im Log: %e

Beispiel (Konfiguration anpassen)

```
log_line_prefix = '%m {%e}'
log_min_duration_statement = 0
```

Beispiel (Ausgabe im Logfile)

```
2010-02-26 22:35:58.750 CEST {00000} LOG: duration: 0.105 ms statement: BEGIN;
2010-02-26 22:36:05.009 CEST {00000} LOG: duration: 0.177 ms statement: SELECT 5;
2010-02-26 22:36:11.284 CEST {42703} ERROR: column "ertz" does not exist at character 8
2010-02-26 22:36:11.284 CEST {42703} STATEMENT: SELECT ertz;
2010-02-26 22:36:15.140 CEST {25P02} ERROR: current transaction is aborted, commands ignored until end of
2010-02-26 22:36:15.140 CEST {25P02} STATEMENT: SELECT 1;
2010-02-26 22:36:27.820 CEST {00000} LOG: duration: 0.088 ms statement: ROLLBACK;
2010-02-26 22:36:29.614 CEST {00000} LOG: duration: 0.198 ms statement: SELECT 2;
```

\d+ in psql zeigt alle abgeleiteten Tabellen

Beispiel (abgeleitete Tabellen erstellen)

```
CREATE TABLE master ();
CREATE TABLE child1 () INHERITS (master);
CREATE TABLE child2 () INHERITS (master);
CREATE TABLE child3 () INHERITS (master);
```

Beispiel (Ausgabe in psql)

```
\d+ master
              Table "public"."master"
...
Child tables: child1,
              child2,
              child3
```


Indexmethode anzeigen

Beispiel (Indexes anzeigen)

```
test=# \di
                      List of relations
 Schema |     Name      | Type  |   Owner    | Table  | Method
--------+---------------+-------+------------+--------+--------
 public | test_id_idx   | index | postgresql | table1 | btree
 public | test_name_idx | index | postgresql | table1 | hash
(2 rows)
```

ByteA-Format unterstützt jetzt HEX

- Das ByteA-Format wird jetzt per Default in Hex ausgegeben
- Das alte Format wird weiterhin unterstützt
- Wird über `bytea_output` GUC konfiguriert (mögliche Werte: escape, hex)

Beispiel (ByteA-Format)

```
SET bytea_output TO hex;
SELECT E'\\xDeAdBeEf'::bytea;
```


Join Removal

- Der Planer entfernt Tabellen aus einem LEFT JOIN
- Wenn: die Spalte(n) nicht in der Ergebnismenge auftauchen
- Wenn: die rechte Spalte eindeutig ist (erfordert einen UNIQUE Constraint)

CREATE LIKE verbessert

- Kopiert jetzt Kommentare und Storage Optionen mit

Trigger auf Spalten

- Ein Trigger kann nun eine WHERE-Bedingung enthalten

Beispiel (ByteA-Format)

```
CREATE TRIGGER test123 BEFORE INSERT ON test
   FOR EACH ROW
  WHEN NEW.data != OLD.data
EXECUTE PROCEDURE trigger_function();
```

- NEW und OLD stehen wie innerhalb der Triggerfunktion zur Verfügung
- Subselects sind in der WHERE-Bedinung nicht möglich

Development

VACUUM FULL beschleunigt

- `VACUUM FULL` nutzt jetzt intern die Funktionalität von `CLUSTER`

Development

Passwortstärke kann jetzt geprüft werden

- Über einen neu bereitgestellten Hook in PostgreSQL kann der Admin die Stärke eines neu vergebenen Passworts prüfen lassen
- Ein Beispiel wird in contrib mitgeliefert

Application Name

- Programme können jetzt einen Application Name angeben
- Dieser erscheint unter anderem in den Logs und in `pg_stat_activity`
- z. B. JDBC verwendet eine derartige Funktionalität

Exclusion Constraints

- Nicht verwechseln: Exclusion Constraints != `constraint_exclusion`
- Bietet ein Framework für Constraints

- Nicht überlappende geografische Daten (PostGIS)
- Nicht überlappende Zeiträume (Buchen eines Besprechungszimmers)

- Hilfreich: `PERIOD` Datentyp (auf pgFondry)
- Ansonsten: ein Datentyp mit Start- und Endzeit

Exclusion Constraints

Beispiel (Beispiel für Exclusion Constraints)

```
CREATE TABLE buchungen (
  beschreibung TEXT,
  raum         TEXT,
  zeit         PERIOD,
  EXCLUDE USING gist
  (raum WITH =, zeit WITH &&)
```

- && ist der `overlap(period1, period2)` Operator
- Derzeit ist nur `gist` unterstützt
- Dafür können Spalten und Ausdrücke angegeben werden
- Mehrfache Prüfungen sind möglich, einfach mehrfach `EXCLUDE` angeben

Notify mit Payload

- `LISTEN/NOTIFY` wurde neu implementiert
- Payloads (Zusatzinformationen) sind jetzt möglich (bis 8000 Zeichen)
- Gleiche Notifies (gleiches Listen plus gleiche Payload) werden weiterhin in ein Notify zusammengefasst
- Notifies werden beim Commit (in Commit Order) zugestellt

Beispiel (NOTIFY mit Payload)

```
NOTIFY something_changes 'Table abc, Row 3 changed';
```

Betriebssysteme

Windows Support verbessert

- "could not reattach to shared memory" Ursache ist beseitigt
- Support für Windows 64-Bit hinzugefügt

Stored Procedures

Verbessertes hstore

- 64k Limit Keys und Werte entfällt
- Unterstützung für Btree und Hash Operatorklassen hinzugefügt
- notwendig für: `GROUP BY`, `DISTINCT`
- Format der Dateien ändert sich, altes Format ist weiterhin lesbar
- Eine Reihe neuer Operatoren, unter anderem für Array-Operationen in hstore

Stored Procedures in C++

- Das PostgreSQL-Backend unterstützt jetzt Aufrufe aus C++
- Z. B. können SPI-Funktionen aus C++ aufgerufen werden
- Beim ./configure muss --enable-cplusplus angegeben werden

Skriptsprachen

- Umfangreiche Änderungen am Code für die Perl-Unterstützung
- (Auch in Hinsicht auf Perl/Parrot)
- Unterstützung für Python 3.1

Streaming Replication + Hot Standby

- PostgreSQL 9.0 enthält Streaming Replication
- PostgreSQL 9.0 enthält Hot Standby

Streaming Replication

- Archive Logs (16 MB) werden wie gehabt zum Slave übertragen
- Zusätzlich pollt der Slave aktuelle Änderungen vom Master
- Bei zu großem Lag werden (zuerst) die Logfiles eingespielt

- Im Master wird weiterhin der `archive_mode` aktiviert
- Die Anzahl Clients über `max_wal_senders = n` konfiguriert

Streaming Replication – der Slave

- Normale Konfiguration in `recovery.conf`
- Zusätzlich: `standby_mode = true`
- Zusätzlich: `primary_conninfo = 'host=masterdb port=5432'`
- Zusätzlich: `trigger_file = /tmp/trigger.hs`

- Nachteil: es ist weiterhin ein Base-Backup vom Master notwendig
- Die Transaktionslogs müssen weiterhin über einen externen Weg kopiert werden

Streaming Replication – der Master

- Benötigt (derzeit) einen Superuser

Beispiel (Superuser Zugang für Replikation)

```
# TYPE   DATABASE      USER   CIDRADDRESS     METHOD
host     replication   rep    192.168.30.5    md5
host     replication   all    0.0.0.0/0       reject
```

Hot Standby

- Der Slave kann für (lesende) Anfragen genutzt werden
- Transaktionen auf dem Slave sind komplett, aber ggf. einige Zeit hinter dem Master
- Einige Lockingmodi können verwendet werden, andere nicht
- Two-Phase Commits sind nicht möglich (erfordern einen WAL-Eintrag)
- Nicht möglich: `LISTEN`, `NOTIFY`, `UNLISTEN`

Hot Standby – Konfliktlösungen

- Der Slave wartet `max_standby_delay` Sekunden bei einem Konflikt
- Transaktionen und Abfragen, die einen Lock halten, werden beendet
- Idle Transaktionen werden beendet
- Konflikte durch Aufräumarbeiten (Entfernen veralteter Records auf dem Master) informieren die Anfrage, diese beendet sich selbst (ansonsten: falsche Ergebnisse)

Hot Standby – Failover

- Im Failover Fall (Slave beendet Recovery) bleiben alle Verbindungen bestehen
- Transaktionen werden schreibbar

Wichtiger Termin

Wichtiger Termin
`FrOSCon 2010`

- 21./22. August 2010
- St. Augustin (bei Bonn)

Webseite: `http://www.froscon.de/`

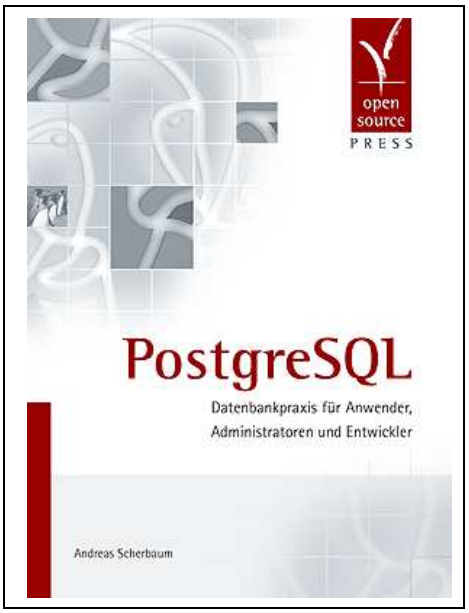
Wichtiger Termin

Wichtiger Termin
PGDay 2010

- irgendwann im 2. Halbjahr (wahrscheinlich Septemter)
- irgendwo in den Niederlanden (wahrscheinlich Amsterdam)

Webseite: `http://www.pgday.eu/`

PostgreSQL Buch



PostgreSQL – Datenbankpraxis für Anwender, Administratoren und Entwickler

Erschien im Juli 2009 im Verlag Open Source Press
Umfang: ca. 520 Seiten

Ende

http://andreas.scherbaum.la/

Fragen?

Andreas 'ads' Scherbaum <andreas@scherbaum.biz>
PostgreSQL User Group Germany
European PostgreSQL User Group

PostgreSQL Service & Support