Datenbanken von MySQL zu PostgreSQL portieren

CLT 2011 – 19./20.03.2011

Andreas 'ads' Scherbaum

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

19./20.03.2011

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 9.1 wird derzeit fleißig entwickelt

Der Dozent

- Name: Andreas Scherbaum
- ~~Selbstständig im Bereich Datenbanken, Linux auf Kleingeräten, Entwicklung von Webanwendungen~~
- Seit Anfang 2011 bei EMC^2/Greenplum, Data Computing Division
- Arbeit mit Datenbanken seit 1997, mit PostgreSQL seit 1998
- Gründungsmitglied der Deutschen und der Europäischen PostgreSQL User Group
- Board of Directors – European PostgreSQL User Group

Was bedeutet dieses komische .la?

`http://andreas.scherbaum.la/`

- `.la` ist die TLD von Laos
- `.la` wird von Los Angeles genutzt und verwaltet
- `LA` ist das KFZ-Kennzeichen von Landshut/Niederbayern
- Dort habe ich längere Zeit gewohnt und meine Frau fand die Domain schön

Inhalte des Vortrags

- Geschichte von MySQL
- Fallstricke beim Portieren
- Hilfsmittel

Die (jüngere) Geschichte von MySQL

- 1994: mit Versionsnummer 3.21 veröffentlicht
- erlangte schnell Bedeutung im Web, zusammen mit PHP (LAMP)
- **Oktober 2005**: Oracle kauft InnoDB (Storage Engine für MySQL)
- **Februar 2006**: Oracle kauft Sleepycat (u. a. Storage Engine für MySQL)
- **2006**: versuchte Übernahme durch Oracle, Hintergründe unklar
- **Februar 2008**: von Sun Microsystems übernommen
- **April 2009**: Oracle übernimmt Sun Microsystems
- unter anderem: Java, MySQL, OpenSolaris, OpenOffice, GlassFish
- **November 2010**: Verwirrungen in der Presse über Preiserhöhungen

Verwirrungen auf Seiten der Anwender

- keine klare Aussage von Oracle zur Zukunft von MySQL
- Verwirrungen um Preiserhöhungen
- Probleme in anderen von Oracle geführten Projekten (OpenSolaris, OpenOffice, Hudson, ...)

- diverse Forks mit unterschiedlicher Positionierung
- diverse Storage Engines mit unterschiedlicher Funktionalität

Alternativen für Anwender

- Wechsel auf eine andere Datenbank

Ansatzpunkte

- Frameworks (z. B. Hybernate) erleichtern die Arbeit
- Analyse der Anwendung(en), Aktualisieren der Dokumentation
- Portieren der Datenbank
- Portieren der Anwendung(en)

AUTO_INCREMENT

- AUTO_INCREMENT zählt bei INSERTs um eins hoch

Beispiel (AUTO_INCREMENT)

```
CREATE TABLE ... (
  id     INTEGER     PRIMARY KEY
                     AUTO_INCREMENT,
  ...
);
```


AUTO_INCREMENT Ersatz: SERIAL

- Sequenzen (SERIAL) in PostgreSQL bieten die gleiche Funktionalität ... und mehr

Beispiel (SERIAL)

```
CREATE TABLE ... (
  id     SERIAL      PRIMARY KEY,
  ...
);
```

- Sequenzen können hoch und runter zählen
- ... können in größeren Schritten zählen
- ... können für mehrere Tabellen verwendet werden
- ... können unabhängig von einer Tabelle verwendet werden

TIMESTAMP

- Die erste TIMESTAMP-Spalte einer Tabelle wird von MySQL automatisch gesetzt

Beispiel (TIMESTAMP)

```
CREATE TABLE ... (
  geaendert_am     TIMESTAMP,
  ...
);
```

- Nachteil: das "Feature" lässt sich nicht deaktivieren

TIMESTAMP Ersatz: Trigger

- Ein Trigger setzt zuverlässig den aktuellen Wert

Beispiel (Trigger)

```
CREATE TABLE ... (
  geaendert_am     TIMESTAMPTZ,
  ...
);

CREATE TRIGGER trigger_zeitstempel
       BEFORE INSERT OR UPDATE ON ...
       FOR EACH ROW EXECUTE PROCEDURE trigger_zeitstempel();
```

- PostgreSQL Vorteil: keine zweite Spalte notwendig

TIMESTAMP Ersatz: Trigger

Beispiel (Trigger)

```
CREATE FUNCTION trigger_zeitstempel ()
       RETURNS TRIGGER AS $$
BEGIN

  IF TG_OP = 'INSERT' THEN
      NEW.eingefuegt_um := NOW();
      NEW.geaendert_um := NOW();
  END IF;
  IF TG_OP = 'UPDATE' THEN
      NEW.eingefuegt_um := OLD.eingefuegt_um;
      NEW.geaendert_um := NOW();
  END IF;

  RETURN NEW;
END
$$ LANGUAGE plpgsql;
```


CHECK-Klausel

- MySQL akzeptiert aber ignoriert CHECK-Klauseln

Beispiel (TIMESTAMP)

```
CREATE TABLE ... (
  passwort     CHAR(32)
               CHECK (LENGTH(passwort) = 32),
  ...
);
```

CHECK-Klausel

- PostgreSQL akzeptiert CHECK-Klauseln und wendet diese an
- Nachteil: Sie müssen jetzt CHECK-Klauseln schreiben ;-)

DEFAULT-Werte

- DEFAULT-Werte werden ähnlich behandelt
- Sonderbehandlung für TIMESTAMP-Spalten
- Für NOT NULL-Spalten generiert MySQL automatisch einen DEFAULT-Wert:
- 0 für Zahlen
- "" (leerer String) für Texte
- bei ENUMs der erste Wert
- in PostgreSQL sind dafür explizit DEFAULT-Werte zu setzen
- ein INSERT ohne Angabe eines Wertes führt zu einem Fehler

ZEROFILL

- MySQL kennt ZEROFILL, um Spalten mit 0 am Anfang aufzufüllen

Beispiel (ZEROFILL)

```
CREATE TABLE ... (
  zahl       INTEGER ZEROFILL,
  ...
);
```

Beispiel (ZEROFILL)

```
+------------+
| zahl       |
+------------+
| 0000000001 |
+------------+
```


ZEROFILL Ersatz: passend formatierte Ausgabe

- in PostgreSQL ist das Zahlenformat Aufgabe der Ausgabe

Beispiel (ZEROFILL)

```
test=# SELECT lpad(1::TEXT, 10, '0');
    lpad
------------
 0000000001
(1 Zeile)
```

GROUP BY

- MySQL erlaubt die Angabe einzelner Spalten bei `GROUP BY`

Beispiel (GROUP BY)

```
CREATE TABLE groupby_test (
  id          INTEGER      PRIMARY KEY,
  daten       VARCHAR(10)  NOT NULL
);

INSERT INTO groupby_test (id, daten) VALUES (1, 'Auto');
INSERT INTO groupby_test (id, daten) VALUES (2, 'Boot');
INSERT INTO groupby_test (id, daten) VALUES (3, 'Flugzeug');
INSERT INTO groupby_test (id, daten) VALUES (4, 'Auto');
INSERT INTO groupby_test (id, daten) VALUES (5, 'Boot');
```


GROUP BY

Beispiel (GROUP BY)

```
mysql> SELECT id, daten FROM groupby_test GROUP BY daten;
+----+----------+
| id | daten    |
+----+----------+
|  1 | Auto     |
|  2 | Boot     |
|  3 | Flugzeug |
+----+----------+
3 rows in set (0.00 sec)
```

- Wir erinnern uns: `Auto` hatte die IDs 1 **und** 4 ...

GROUP BY Ersatz: Anfragen richtig schreiben

- PostgreSQL verlangt:
- alle Spalten werden in GROUP BY aufgeführt
- oder in einer Aggregatfunktion verwendet

Beispiel (GROUP BY)

```
test=# SELECT id, daten FROM groupby_test GROUP BY daten;
ERROR:  column "groupby_test.id" must appear in the GROUP BY
        clause or be used in an aggregate function
ZEILE 1: SELECT id, daten FROM groupby_test GROUP BY daten;
```


GROUP BY Ersatz: Anfragen richtig schreiben

Beispiel (GROUP BY)

```
test=# SELECT MIN(id), daten FROM groupby_test GROUP BY daten;
 min |  daten
-----+----------
   2 | Boot
   1 | Auto
   3 | Flugzeug
(3 Zeilen)
```

- Vorteil: eindeutige Ergebnisse

Sortieren und NULL-Werte

- MySQL sortiert NULL-Werte an den Anfang der Ergebnisliste
- PostgreSQL an das Ende

Beispiel (NULL)

```
CREATE TABLE null_test (
  id       INTEGER          PRIMARY KEY,
  daten    VARCHAR(10)
);

INSERT INTO null_test (id, daten) VALUES (1, 'a');
INSERT INTO null_test (id, daten) VALUES (2, NULL);
INSERT INTO null_test (id, daten) VALUES (3, 'b');
INSERT INTO null_test (id, daten) VALUES (4, NULL);
INSERT INTO null_test (id, daten) VALUES (5, 'c');
```


Sortieren und NULL-Werte

Beispiel (NULL: in MySQL)

```
mysql> SELECT id, daten FROM null_test ORDER BY daten;
+----+-------+
| id | daten |
+----+-------+
|  2 | NULL  |
|  4 | NULL  |
|  1 | a     |
|  3 | b     |
|  5 | c     |
+----+-------+
5 rows in set (0.06 sec)
```

Sortieren und NULL-Werte

Beispiel (NULL: in PostgreSQL)

```
test=# SELECT id, daten FROM null_test ORDER BY daten;
 id | daten
----+-------
  1 | a
  3 | b
  5 | c
  2 |
  4 |
(5 Zeilen)
```


Sortieren und NULL-Werte: NULLS FIRST

Beispiel (NULL: NULLS FIRST)

```
test=# SELECT id, daten FROM null_test ORDER BY daten NULLS FIRST;
 id | daten
----+-------
  2 |
  4 |
  1 | a
  3 | b
  5 | c
(5 Zeilen)
```

IFNULL()

- MySQL kennt `IFNULL()` und `COALESCE()`
- unterscheiden sich jedoch nicht wesentlich

Beispiel (IFNULL())

```
mysql> SELECT IFNULL(NULL, 10);
+------------------+
| IFNULL(NULL, 10) |
+------------------+
|               10 |
+------------------+
1 row in set (0.00 sec)
```


COALESCE()

Beispiel (COALESCE())

```
mysql> SELECT COALESCE(NULL, 10, 20);
+-----------------------+
| COALESCE(NULL, 10, 20) |
+-----------------------+
|                    10  |
+-----------------------+
1 row in set (0.00 sec)
```

IFNULL() Ersatz: COALESCE()

- Einfach überall `IFNULL()` gegen `COALESCE()` austauschen
- Unterschied:
- `IFNULL()` kennt nur 2 Parameter
- `COALESCE()` kann mehrere Parameter verarbeiten

Groß-/Kleinschreibung der Bezeichner

- In MySQL bestimmt (bei einigen Tabellentypen) das Dateisystem die Groß-/Kleinschreibung
- Unter Windows ist die Groß-/Kleinschreibung egal
- Unter einigen Unix-Systemen ist die Groß-/Kleinschreibung wichtig
- PostgreSQL ist das Dateisystem egal ;-)

Groß-/Kleinschreibung der Bezeichner

- In MySQL bestimmt `lower_case_table_names` das Verhalten
- 0: case-sensitive (nicht unter Windows oder Mac OS X verwenden!)
- 1: Tabellennamen werden klein geschrieben, und verglichen
- 2: Tabellennamen werden wie angegeben geschrieben, aber klein verglichen

Groß-/Kleinschreibung der Bezeichner

- Der SQL-Standard schreibt alle Bezeichner groß
- PostgreSQL schreibt alle Bezeichner klein

Beispiel (Groß-/Kleinschreibung)

```
test# SELECT 1 AS GROSS;
 gross
-------
     1
(1 Zeile)
```

Beispiel (Groß-/Kleinschreibung)

```
test# SELECT 1 AS GeMiScHt;
 gemischt
----------
        1
(1 Zeile)
```

Groß-/Kleinschreibung der Bezeichner

- Möchte man Bezeichner groß schreiben sind "" zu verwenden

Beispiel (Groß-/Kleinschreibung)

```
test# SELECT 1 AS "GeMiScHt";
 GeMiScHt
----------
        1
(1 Zeile)
```

- Vorsicht bei der Verwendung von Frameworks

CONSTRAINTs und REFERENCES

- Einige MySQL Tabellentypen kennen CONSTRAINTs und REFERENCES
- andere nicht
- Folge: kaum jemand setzt das ein

- Weitere Besonderheiten:
- beide Spalten müssen die gleiche Definition haben (gleicher Datentyp, NULL/NOT NULL)
- beide Spalten müssen einen Index haben

Datums- und Zeitangaben

- Datentypen für Zeit- und Datumsangaben unterscheiden sehr stark
- Funktionen zum Formatieren der Ausgabe sind unterschiedlich
- `TIMESTAMP` in PostgreSQL hat eine Auflösung im Mikrosekundenbereich
- zusätzlich hat ein `TIMESTAMPTZ` eine Zeitzone
- Operationen mit Zeitangaben geben in PostgreSQL den Typ `INTERVAL` zurück
- Fazit: viel Detail- und Handarbeit notwendig :-(

Datums- und Zeitangaben

- Beispiel: `year()`, `month()` und `day()`

Beispiel (Datumsfunktionen)

```
test=# SELECT to_char(NOW(), 'YYYY') AS year,
             to_char(NOW(), 'MM') AS month,
             to_char(NOW(), 'DD') AS day;
 year | month | day
------+-------+-----
 2010 | 12    | 03
(1 Zeile)
```

ORDER BY RAND()

- Funktion zum Erzeugen von Zufallszahlen
- in MySQL: `RAND()`
- in PostgreSQL: `RANDOM()`
- Suchen/Ersetzen sollte ausreichen

LIKE und ILIKE

- `LIKE` in MySQL unterscheidet nicht zwischen Groß-/Kleinschreibung
- `LIKE` in PostgreSQL ist case-sensitive
- für case-insensitive Suchen: `ILIKE`

LIKE und ILIKE

Beispiel (LIKE)

```
test# SELECT 'AUTO' LIKE 'auto';
 ?column?
----------
 f
(1 Zeile)
```

Beispiel (ILIKE)

```
test# SELECT 'AUTO' ILIKE 'auto';
 ?column?
----------
 t
(1 Zeile)
```


Boolesche Werte

- MySQL kennt keinen (richtigen) Boolean-Wert
- stattdessen wird ein `SMALLINT(1)` verwendet
- führt dazu das ein Boolean auch mal 7 verschiedene Werte haben kann ...
- PostgreSQL kennt einen richtigen `BOOLEAN`-Datentyp

String-Verknüpfungen versus logische Operatoren

- MySQL verwendet den || Operator für "logisch ODER"
- Der SQL-Standard – und PostgreSQL– verwenden || für Textverknüpfungen
- Spaß ist vorprogrammiert
- "logisch ODER" wird in PostgreSQL mit Hilfe des `OR`-Operators durchgeführt

Intro MySQL Portieren → PostgreSQL Fragen & Antworten
Fallstricke

String-Verknüpfungen versus logische Operatoren

- MySQL kennt auch `&&` für "logisch UND"
- das hat jedoch keine Entsprechung in anderen Datenbanken
- ist daher einfach zu finden und zu beseitigen

Binäre Daten

- MySQL verwendet `VARBINARY` oder `BINARY`
- PostgreSQL verwendet `BYTEA`
- Anwendung wird primär durch die Anwendung bestimmt

`INSERT IGNORE` und `REPLACE`

- MySQL erlaubt mittels `INSERT IGNORE` das Überspringen von Unique-Key Verletzungen
- PostgreSQL erlaubt das nicht
- `REPLACE` überschreibt die Zeile mit dem Primärschlüssel mit den neuen Daten
- Quasi ein `INSERT ODER UPDATE`
- Nachteil: kein SQL-Standard

INSERT IGNORE und REPLACE Ersatz

- Abhilfe für INSERT IGNORE
- die PostgreSQL Online-Dokumentation enthält eine Beispielfunktion
- Stichwort: UPSERT

- Abhilfe für REPLACE: MERGE
- Demnächst auch in ihrer bevorzugten Datenbank

Daten aus Datei laden

- MySQL verwendet LOAD DATA zum Importieren von Daten
- PostgreSQL verwendet COPY

- Anwendung ist ähnlich

Daten in Datei schreiben

- MySQL verwendet `SELECT ... INTO OUTFILE` zum Exportieren von Daten
- PostgreSQL verwendet wiederum `COPY`

Kommentare

- MySQL versteht als Kommentar:
- die Raute: `#`
- doppelte Bindestriche: `--`
- für mehrzeilige Kommentare: `/* ... */`
- PostgreSQL versteht die Raute (#) nicht als Kommentar

Anführungszeichen

- MySQL erlaubt die Verwendung von einfachen und doppelten Anführungszeichen für Daten und Bezeichner
- PostgreSQL verlangt einfache Hochkommas für Daten (laut SQL-Standard)
- PostgreSQL verlangt doppelte Anführungszeichen für Bezeichner (laut SQL-Standard)

- MySQL erlaubt die Verwendung von Akzentmarkern (Backticks) für Bezeichner
- Ein Anfang wäre das Exportieren mittels "ansi":

Beispiel (mysqldump)

```
mysqldump --compatible=ansi
```


Storage-Engines

- MySQL kennt verschiedene Storage-Engines:
- MyISAM, Innodb, Memory, Archive, CSV, PBXT, Solid, Falcon, NDB, GEMINI, BerkeleyDB, Blackhole, Federated, Merge, IBMDB2I, Maria, ScaleDB, XtraDB, Calpont, InfoBright, Kickfire, TokuDB, HEAP, Example, Isam, Q4M, OQGraph, FederatedX, Spider, Sphinx, AWSS3

- Problem: jede Storage Engine bietet andere Vor- und Nachteile
- Implementierungsdetails (Volltextsuche, Transaktionen, Foreign Keys, Check Constraints, Groß-/Kleinschreibung, ...) sind abhängig von der Engine
- Viel Spaß bei der Auswahl des passenden Typs :-(

Storage-Engines in PostgreSQL

- PostgreSQL kennt keine Storage-Engines ;-)
- jede Tabelle kann alle Features
- Alle `ENGINE`- oder `TYPE`-Parameter können entfernt werden

Transaktionen

- MySQL kennt Transaktionen – bei einigen Storage Engines
- PostgreSQL nutzt überall Transaktionen
- Nutzen Sie das!

Storage-Engines in PostgreSQL

- PostgreSQL kennt keine Storage-Engines ;-)
- jede Tabelle kann alle Features
- Alle `ENGINE`- oder `TYPE`-Parameter können entfernt werden

Datentypen

- Die Datentypen unterscheiden sich, teilweise sogar sehr stark

Datentyp	Ersatz
TINYINT	SMALLINT
SMALLINT	SMALLINT
MEDIUMINT	INTEGER
INT	INTEGER
BIGINT	BIGINT
INT(1)	BOOLEAN oder SMALLINT
INT(4)	INTEGER
INT(11)	BIGINT
FLOAT	FLOAT
DOUBLE	DOUBLE PRECISION
REAL	DOUBLE PRECISION
FLOAT(4,7)	FLOAT
NUMERIC	NUMERIC
DECIMAL	DECIMAL

Datentypen

Datentyp	Ersatz
CHAR	CHAR
VARCHAR	VARCHAR
BINARY	BYTEA oder TEXT
VARBINARY	BYTEA oder TEXT
BLOB	TEXT
TEXT	TEXT
LONG	TEXT
ENUM	ENUM oder 1:n-Tabelle
SET	kein Ersatz, 1:n-Tabelle möglich

Datentypen

Datentyp	Ersatz
DATETIME	TIMESTAMP oder TIMESTAMPTZ
DATE	DATE
TIMESTAMP	TIMESTAMP oder TIMESTAMPTZ
TIME	TIME
YEAR	DATE

Datentyp	Ersatz
BIT	BIT oder BIT VARYING

mysql2pgsql

- `mysql2pgsql` kann einen Datenbankdump portieren
- funktioniert recht gut, trotzdem ist Handarbeit notwendig
- Import dauert länger, da `INSERT`s statt `COPY` verwendet werden

Webseite: `http://pgfoundry.org/projects/mysql2pgsql/`

mysql2pgsql

Beispiel (mysql2pgsql)

```
perl mysql2pgsql.perl mysql-dump.sql pg-dump.sql
```

mysql2pgsql – nützliche Optionen

- `--debug` liefert umfangreiche Ausgaben
- `--char2varchar` wandelt alle `CHAR`-Spalten in `VARCHAR`-Typen um
- `--nodrop` entfernt alle `DROP TABLE`-Anweisungen
- `--schema` gibt ein Schema für Objekte an
- `--enc_in` Charset des MySQL-Dumps
- `--enc_out` Charset der Ausgabedatei für PostgreSQL

mehrere Indexes

- PostgreSQL kann mehrere Indexes pro Anfrage verwenden

Feedback

- Feedback ist wichtig für:
- die Konferenz
- den Vortragenden

Webseite: `http://chemnitzer.linux-tage.de/2011/`

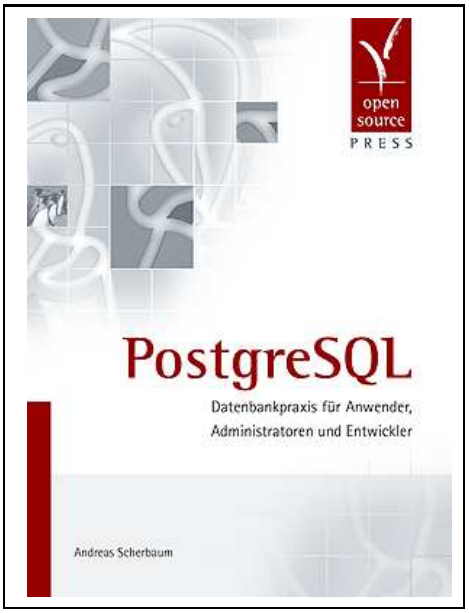
Wichtiger Termin

Wichtiger Termin
PGConf 2011

- When: tbd
- Where: tbd

Webseite: `http://2010.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.biz/

http://andreas.scherbaum.la/

Fragen?

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

PostgreSQL Service & Support