Port databases from MySQL to PostgreSQL

PGConf.EU 2011 – October 18-21, 2011

Andreas 'ads' Scherbaum

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

October 18-21, 2011
Was ist PostgreSQL?

- Relational Database, released under the PostgreSQL License (similar to BSD-License)
- Worldwide active community
- Numerous features and functions (Foreign Keys, Transactions, Triggers, ...)
- Runs on many Operating Systems
- New releases every year, support for 5 years/releases (whichever is longer)
- Version 9.2 is in development

PostgreSQL – The correct name

- "PostgreSQL " is the product name
- "Postgres" is a widely accepted alias
- "PG" and "PGSQL" are recognized
- "Postgre" is just plain wrong
Andreas Scherbaum

- Work with databases since 1997, with PostgreSQL since 1998
- Founding Member of the European and the German PostgreSQL User Group
- Board of Directors – European PostgreSQL User Group
- Regional Contact PostgreSQL for Germany
- Ran my own company for 7+ years – around PostgreSQL
- Since 2011 with $EMC^2$/Greenplum, Data Computing Division

What does this queer .la mean?

http://andreas.scherbaum.la/
- .la is the TLD of Laos
- .la is used and managed by Los Angeles
- LA is also the plate number for Landshut (Lower Bavaria, Germany)
- I lived there for a long time, my wife liked the domain (and I bought several more TLDs)
History of MySQL

1994: started with version number 3.21
quickly gained importance on the Web, together with PHP (LAMP)
**October 2005:** Oracle acquires InnoDB (Storage Engine for MySQL)
**February 2006:** Oracle acquires Sleepycat (another Storage Engine for MySQL)
2006: attempted takeover by Oracle, background unclear
**February 2008:** acquired by Microsystems
**April 2009:** Oracle acquires Sun Microsystems
among others: Java, MySQL, OpenSolaris, OpenOffice, GlassFish
**November 2010:** Confusion in the press about price increases
Confusion among users

- No clear statement on the future of MySQL by Oracle
- Confusion about price increases
- Problems in other projects undertaken by Oracle (OpenSolaris, OpenOffice, Hudson, ...)
- Forks with a variety of different positionings
- Various storage engines with different functionality

Alternatives for users

- Changing to a different database
Porting to PostgreSQL
Starting points

- Frameworks (eg Hibernate) simplify the work
- Analysis of the application(s), update the documentation
- Porting the database
- Porting the application(s)

AUTO_INCREMENT

- AUTO_INCREMENT increases at every INSERT

Example (AUTO_INCREMENT)

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

Andreas 'ads' Scherbaum  MySQL → PostgreSQL
AUTO_INCREMENT Replacement: SERIAL

- Sequences (SERIAL) in PostgreSQL provide the same functionality ... and more

Example (SERIAL)

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

- Sequences can count up and down
- ... can count in wider steps
- ... can be used for more than one table
- ... can be independent of a table

TIMESTAMP

- The first TIMESTAMP column in a table is set automatically by MySQL

Example (TIMESTAMP)

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

- Disadvantage: this "feature" cannot be deactivated
TIMESTAMP Replacement: Trigger

A trigger sets the current value in a reliable way.

Example (Trigger)

```sql
CREATE TABLE ... (
    changed_at TIMESTAMPTZ,
    ...
);

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

PostgreSQL Advantage: needs no second column

Example (Trigger)

```sql
CREATE FUNCTION trigger_settime ()
    RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        NEW.insert_at := NOW();
        NEW.changed_at := NOW();
    END IF;
    IF TG_OP = 'UPDATE' THEN
        NEW.insert_at := OLD.insert_at;
        NEW.changed_at := NOW();
    END IF;
    RETURN NEW;
END
$$ LANGUAGE plpgsql;
```
CHECK Clause

MySQL accepts but ignores CHECK clauses

Example (TIMESTAMP)
CREATE TABLE ... (  
    password CHAR(32)  
    CHECK (LENGTH(password) = 32),
...  
);

CHECK-Klausel

PostgreSQL accepts CHECK clauses and enforces them
Disadvantage: Now you have to write CHECK clauses ;-)
**DEFAULT Values**

- DEFAULT values are handled similar
- Special treatment for TIMESTAMP columns
- MySQL generates a DEFAULT value for NOT NULL columns:
  - 0 for numbers and integers
  - "" (empty string) for text
  - first value for ENUMs

- in PostgreSQL, DEFAULT are to set explicitly
- a INSERT without value will result in an error

---

**ZEROFILL**

- MySQL knows ZEROFILL to fill columns with 0 at the beginning

Example (ZEROFILL)

```sql
CREATE TABLE ... (
    number      INTEGER ZEROFILL,
    ... 
);
```

```
+------------+
| number |
+------------+
| 0000000001 |
+------------+
```

Example (ZEROFILL)

```text
+--------------+
| number |
+--------------+
| 0000000001 |
+--------------+
```
Pitfalls

**ZEROFILL Replacement: suitable formatted output**

- In PostgreSQL the number format is part of the output function.

Example (ZEROFILL)

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

Andreas 'ads' Scherbaum

MySQL → PostgreSQL

---

**GROUP BY**

- MySQL allows GROUP BY with single columns.

Example (GROUP BY)

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

INSERT INTO groupby_test (id, data) VALUES (1, 'Car');
INSERT INTO groupby_test (id, data) VALUES (2, 'Ship');
INSERT INTO groupby_test (id, data) VALUES (3, 'Aircraft');
INSERT INTO groupby_test (id, data) VALUES (4, 'Car');
INSERT INTO groupby_test (id, data) VALUES (5, 'Ship');
```

Andreas 'ads' Scherbaum

MySQL → PostgreSQL
GROUP BY

Example (GROUP BY)

```sql
mysql> SELECT id, data FROM groupby_test GROUP BY data;
+----+----------+
| id | data    |
+----+----------+
| 1  | Car     |
| 2  | Ship    |
| 3  | Aircraft |
+----+----------+
3 rows in set (0.00 sec)
```

We remember: Car had the IDs 1 and 4 ...

PostgreSQL requires:
- All columns must appear in GROUP BY
- Or must be used in an aggregate function

Example (GROUP BY)

```sql
test=# SELECT id, data FROM groupby_test GROUP BY data;
ERROR: column "groupby_test.id" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT id, data FROM groupby_test GROUP BY data;
```
GROUP BY Replacement: Write correct queries

Example (GROUP BY)

```
test=# SELECT MIN(id), data FROM groupby_test GROUP BY data;
min | data
-----+----------
 2   | Ship
 1   | Car
 3   | Aircraft
(3 rows)
```

- Advantage: unambiguous results

Sort and NULL Values

- MySQL sorts NULL values first
- PostgreSQL at the end

Example (NULL)

```sql
CREATE TABLE null_test (id INTEGER PRIMARY KEY, data VARCHAR(10));
INSERT INTO null_test (id, data) VALUES (1, 'a');
INSERT INTO null_test (id, data) VALUES (2, NULL);
INSERT INTO null_test (id, data) VALUES (3, 'b');
INSERT INTO null_test (id, data) VALUES (4, NULL);
INSERT INTO null_test (id, data) VALUES (5, 'c');
```
Sort and NULL Values

Example (NULL: in MySQL)

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

Example (NULL: in PostgreSQL)

```sql
test=# SELECT id, data FROM null_test ORDER BY data;
 id | data
----+-------
 1 | a
 3 | b
 5 | c
 2 |
 4 |
(5 rows)
```
Sort and NULL Values: NULLS FIRST

Example (NULL: NULLS FIRST)

```sql
test=# SELECT id, data FROM null_test ORDER BY data NULLS FIRST;
```

```
id | data
----+-------
 2 | 4 |
 1 | a
 3 | b
 5 | c
```

(5 rows)

--

IFNULL()

- MySQL knows IFNULL() and COALESCE()
- However, do not differ significantly

Example (IFNULL())

```sql
mysql> SELECT IFNULL(NULL, 10);
```

```
+------------------+
| IFNULL(NULL, 10) |
+------------------+
| 10 |
```

1 row in set (0.00 sec)
COALESCE()

Example (COALESCE())

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

IFNULL() Replacement: COALESCE()

- Replace every IFNULL() with COALESCE()
- Difference:
  - IFNULL() knows only two parameters
  - COALESCE() can handle more parameters
Upper-/Lowercase of identifiers

In MySQL (depends on the table type) the file system specifies the upper/lowercase handling of identifiers.

- On Windows there is no difference between upper and lowercase names.
- On some Unix Systems the upper and lowercase makes a difference.
- In PostgreSQL the filesystem doesn’t matter ;-)

In MySQL there is a config parameter:

- **lower_case_table_names**

  0: case-sensitive (do not use on Windows or Mac OS X!)
  1: Table names are lowercase, and also compared lowercase
  2: Table names are written as specified, but compared lowercase
Upper-/Lowercase of identifiers

- The SQL standard requires all identifiers to be uppercase
- PostgreSQL makes all identifiers lowercase

Example (Upper-/Lowercase)

```
test# SELECT 1 AS BIG;
big
-----
  1
(1 row)
```

Example (Upper-/Lowercase)

```
test# SELECT 1 AS MiXeD;
mixed
-------
  1
(1 row)
```

If you want to write the identifier in uppercase, you have to use ""

Example (Upper-/Lowercase)

```
test# SELECT 1 AS "MiXeD";
MiXeD
-------
  1
(1 row)
```

Pay attention is you use frameworks!

Andreas 'ads' Scherbaum
CONTRAINTs and REFERENCES

- Some MySQL table types know CONSTRAINTs and REFERENCES
- Others not
- Result: they are rarely used (data integrity, anyone?)

Further characteristics:
- Both columns must have the same definitionn (same data type, NULL/NOT NULL)
- Both columns must have an index

Date and Time Values

- Data types for Date and Time values differ greatly
- Output format functions vary
  - TIMESTAMP in PostgreSQL uses a microsecond resolution
  - in addition: TIMESTAMPTZ includes a time zone
  - Operations involving time values return a type INTERVAL in PostgreSQL
- Conclusion: much manual work is needed :-(

Andreas 'ads' Scherbaum
Date and Time Values

- Example: `year()`, `month()` and `day()`

Example (Date functions)

```sql
test=# SELECT to_char(NOW(), 'YYYY') AS year,
        to_char(NOW(), 'MM') AS month,
        to_char(NOW(), 'DD') AS day;

<table>
<thead>
<tr>
<th>year</th>
<th>month</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>10</td>
<td>03</td>
</tr>
</tbody>
</table>

(1 row)
```

ORDER BY RAND()

- Function to generate random numbers
- in MySQL: `RAND()`
- in PostgreSQL: `RANDOM()`
- Search and Replace should be sufficient
LIKE and ILIKE

- LIKE in MySQL does not distinguish between upper and lower case
- LIKE in PostgreSQL is case sensitive
- for case insensitive searches: ILIKE

Example (LIKE)
```
test# SELECT 'SHIP' LIKE 'ship';
?column?
--------
f
(1 row)
```

Example (ILIKE)
```
test# SELECT 'SHIP' ILIKE 'ship';
?column?
--------
t
(1 row)
```
**Boolean Values**

- MySQL has no (real) boolean value
- A `SMALLINT(1)` is used to emulate a boolean
- It might happen that your boolean actually contains a 7 as a value
- PostgreSQL knows a real `BOOLEAN` data type

---

**String concentration versus logical operators**

- MySQL uses the `||` operator for "logical OR"
- The SQL standard specifies – and PostgreSQL uses – the `||` for text concentration
- You will have fun
- "logic OR" in PostgreSQL is the `OR` operator
String concentration versus logical operators

- MySQL also knows `&&` for "logic AND"
- Fortunately this has no meaning in other databases
- More easy to spot

Binary Data

- MySQL uses `VARBINARY` or `BINARY`
- PostgreSQL uses `BYTEA`
INSERT IGNORE and REPLACE

- MySQL allows skipping of unique key violations by using INSERT IGNORE
- PostgreSQL does not allow that

- REPLACE replaces the line with the same primary key with the new data
- Virtually a INSERT OR UPDATE
- Disadvantage: not in the SQL standard

Replacement for INSERT IGNORE
the PostgreSQL online documentation contains an example
Catchword: UPSERT

Replacement for REPLACE: MERGE
Several attempts to implement it in PostgreSQL, so far we don’t have it
Load data from files

- MySQL uses the `LOAD DATA` syntax
- PostgreSQL uses `COPY`
- Usage is similar

Write data into files

- MySQL uses the `SELECT ... INTO OUTFILE` syntax
- PostgreSQL again uses `COPY`
Comments

- MySQL recognizes as a comment:
  - the hash: `#`
  - double hyphen: `--`
  - for multiline comments: `/* ... */`

- PostgreSQL does not recognize the hash (`#`) as a comment

Quotes

- MySQL allows to use single and double quotes for data and for identifiers
- PostgreSQL requires single quotes for data (SQL standard)
- PostgreSQL requires double quotes for identifier (SQL standard)
- MySQL allows backticks for the identifier
- Export using the "ansi": option of `mysqldump` is a good start

Example (mysqldump)

```
mysqldump --compatible=ansi
```
Storage Engines

MySQL knows a great deal of different storage engines, pick some:

- 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: each storage engine offers different advantages and disadvantages
- Implementation details (full text search, transactions, foreign keys, check constraints, upper / lowercase, ...) are dependent upon the engine
- Enjoy selecting the appropriate type :-(

Storage Engines in PostgreSQL

- PostgreSQL does not know different storage engines ;-)
- every table has all features
- All ENGINE or TYPE parameters must be removed
Transactions

- MySQL knows transactions – in some storage engines
- PostgreSQL uses transactions everywhere
- Use it!

Data Types

- The data types are different, sometimes very greatly

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>MEDIUMINT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>INT</td>
<td>INTEGER</td>
</tr>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
</tr>
<tr>
<td>INT(1)</td>
<td>BOOLEAN or SMALLINT</td>
</tr>
<tr>
<td>INT(4)</td>
<td>INTEGER</td>
</tr>
<tr>
<td>INT(11)</td>
<td>BIGINT</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>REAL</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>FLOAT(4,7)</td>
<td>FLOAT</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL</td>
</tr>
</tbody>
</table>
## Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>BINARY</td>
<td>BYTEA or TEXT</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>BYTEA or TEXT</td>
</tr>
<tr>
<td>BLOB</td>
<td>TEXT</td>
</tr>
<tr>
<td>TEXT</td>
<td>TEXT</td>
</tr>
<tr>
<td>LONG</td>
<td>TEXT</td>
</tr>
<tr>
<td>ENUM</td>
<td>ENUM or 1:n table</td>
</tr>
<tr>
<td>SET</td>
<td>no replacement, 1:n table possible</td>
</tr>
</tbody>
</table>

## Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP or TIMESTAMPTZ</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP or TIMESTAMPTZ</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
</tr>
<tr>
<td>YEAR</td>
<td>DATE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>BIT or BIT VARYING</td>
</tr>
</tbody>
</table>
Multiple Indexes

- PostgreSQL can use multiple indexes per request

Tools

mysql2pgsql

- mysql2pgsql can port a database dump
- Works pretty well, but some manual work is still needed
- Import takes longer because INSERTs are used instead of COPY

Website: http://pgfoundry.org/projects/mysql2pgsql/
mysql2pgsql

Example (mysql2pgsql)
perl mysql2pgsql.perl mysql-dump.sql pg-dump.sql

mysql2pgsql – Useful options

- --debug enables debugging
- --char2varchar transforms all CHAR columns into VARCHAR columns
- --nodrop removes all DROP TABLE statements
- --schema defines a schema for the objects
- --enc_in character set of the MySQL dump
- --enc_out character set for PostgreSQL
Feedback

- Feedback is important for:
  - The conference team
  - The speaker

Website: http://2011.pgconf.eu/feedback

Upcoming Event

German-speaking PostgreSQL Conference

- November 11th
- in Oberhausen, Germany

Website: http://2011.pgconf.de/
PostgreSQL Buch

PostgreSQL – Datenbankpraxis für Anwender, Administratoren und Entwickler

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

Andreas 'ads' Scherbaum
MySQL → PostgreSQL

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