

Port databases from MySQL to PostgreSQL

PGConf.EU 2011 – October 18-21, 2011

Andreas 'ads' Scherbaum

Web: <http://andreas.scherbaum.la/> / <http://andreas.scherbaum.biz/>

E-Mail: [andreas\[at\]scherbaum.biz](mailto:andreas[at]scherbaum.biz)

PGP: 9F67 73D3 43AA B30E CA8F 56E5 3002 8D24 4813 B5FE

October 18-21, 2011

Pitfalls

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)
```

Pitfalls

Upper-/Lowercase of identifiers

- 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 if you use frameworks!

CONSTRAINTs 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 :-(

Date and Time Values

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

Example (Date functions)

```
test=# SELECT to_char(NOW(), 'YYYY') AS year,
               to_char(NOW(), 'MM') AS month,
               to_char(NOW(), 'DD') AS day;
 year | month | day
-----+-----+----
 2011 |   10  |   03
(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

LIKE and 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 concatenation versus logical operators

- MySQL uses the || operator for "logical OR"
- The SQL standard specifies – and PostgreSQL uses – the || for text concatenation
- 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

INSERT IGNORE and REPLACE Replacement

- 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

