From MySQL to PostgreSQL

PostgreSQL Conference Europe 2013

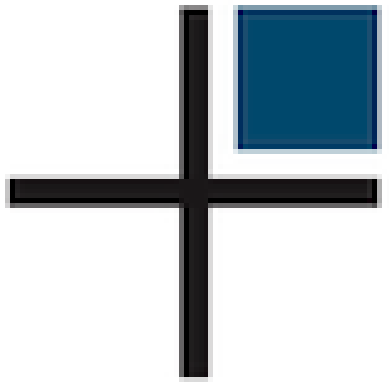
Dimitri Fontaine dimitri@2ndQuadrant.fr
@tapoueh

October, 31 2013

Dimitri Fontaine

2ndQuadrant France
PostgreSQL Major Contributor

- `pgloader`, `prefix`, `skytools`, ...
- `apt.postgresql.org`
- `CREATE EXTENSION`
- `CREATE EVENT TRIGGER`
- MySQL migration tool, new `pgloader` version



Migrating from MySQL to PostgreSQL

A single comand to migrate a whole database

```
load database from mysql://localhost/mydbname
             into postgresql://dim@localhost/pgdbname

with drop tables, truncate, create tables, create indexes,
     reset sequences, downcase identifiers

 set work_mem to '128MB',
      maintenance_work_mem to '512 MB',
     search_path to 'myschema'

cast type datetime to timestamptz drop default using zero-date
     type date drop not null drop default using zero-dates-to-
     type tinyint to boolean using tinyint-to-boolean;
```

Migrating from MySQL to PostgreSQL

Summary output

table name	imported	errors	time
station_mapping_cache	0	0	0.035
test case with errors	5	1	0.138
geo	1	0	0.018
index build completion	0	0	0.0
create index	7	0	0.082
reset sequences	1	0	0.023
Total streaming time	6	1	0.213s

Migrating from MySQL to PostgreSQL

Source available at `http://git.tapoueh.org`

The PostgreSQL Licence

Why Migrating from MySQL to PostgreSQL?

MySQL

- Storage Engine
- Single Application
- Data Loss with Replication
- Weak Data Types Validation
- Either transactions or
- Lack of

PostgreSQL

- Data Access Service
- Application Suite
- Durability and Availability
- Consistency
- Full Text Search, PostGIS
- Proper Documentation

Free your Data!

Cost Analysis

What are the costs?

- Migrating the Data
- Migrating the Code
- Quality Assurance
- Opportunity Cost

Plenty of tools are already available

Those are not solving the *interesting* problems

- `mysql2pgsql` then edit the schema
- `SELECT INTO OUTFILE` on the server, then `COPY`
- MySQL client claims to be sending CSV when redirected
- worst case, some `awk` or `sed` hackery would do
- EnterpriseDB MySQL Migration Wizard
- Python and Ruby scripts

Difficulties when migrating MySQL data

MySQL datatype input functions are really *sloppy*

- Text, empty string and `NULL`
- Depends on the `DEFAULT` value

Difficulties when migrating MySQL data

Dates and The *Gregorian* Calendar

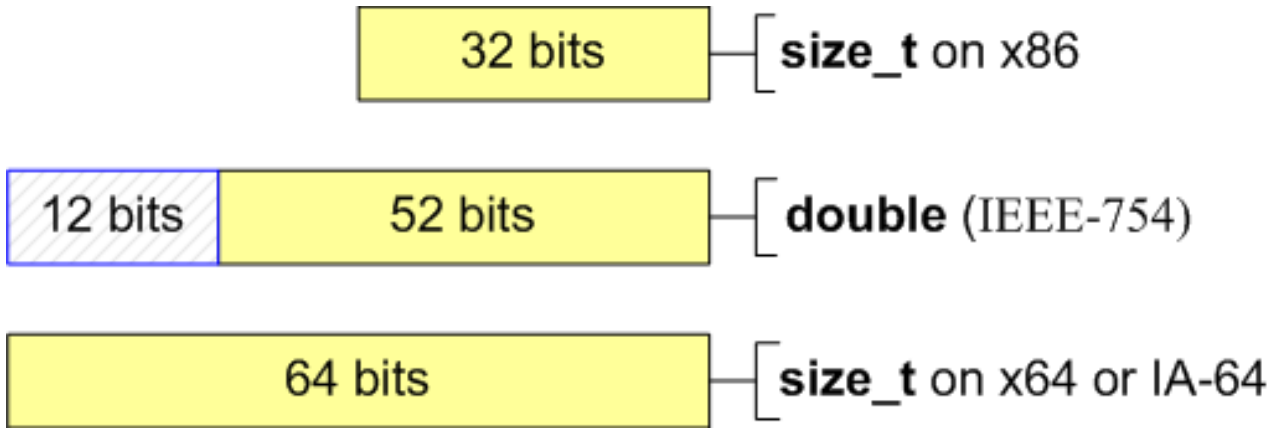
```
MariaDB [talk]> create table dates(d datetime);
MariaDB [talk]> insert into dates
    values('0000-00-00'), ('0000-10-31'), ('2013-10-00');

MariaDB [talk]> select * from dates;
+---------------------+
| d                   |
+---------------------+
| 0000-00-00 00:00:00 |
| 0000-10-31 00:00:00 |
| 2013-10-00 00:00:00 |
+---------------------+
3 rows in set (0.00 sec)
```

Difficulties when migrating MySQL data

MySQL datatype input functions are really *sloppy*

- `int`, `bigint` and `int(11)`
- `decimal(20,2)` and `float(20,2)`



Difficulties when migrating MySQL data

No proper *boolean*, but *sets*

- `tinyint` and `boolean`
- Inline `ENUM` and `SET`
- Geometric datatypes input/output...

```
CREATE TABLE sizes (
    name ENUM('small', 'medium', 'large')
);
```

What is the pgloader added value here?

pgloader is an all-automated declarative migration tool.

What is pgloader?

pgloader `3.x` features

- We're talking about `pgloader version "3.0.50.1"` and later.
- Load data into PostgreSQL
- Using the COPY protocol
- Implementing support for bad rows triaging
- And flexible data input parsing

pgloader supported input

pgloader data source

- CSV and CSV like files
- with per-column `NULL` and *empty string* definitions
- SQLite databases
- MySQL databases
- dBase `DBF` binary files
- Archives (zip) of CSV or DBF files
- Given as an HTTP URL

pgloader feature set

pgloader will happily

- create tables (auto-discovering their schema)
- drop tables first if asked
- or `truncate` them
- create indexes (auto-discovering them)
- in parallel, and in parallel with the next COPY
- normalize identifiers (*quote* or *downcase*)
- reset sequences after the load

Loading zipped dBase files over HTTP

```
LOAD DBF
     FROM http://www.insee.fr/.../.../historiq2013.zip
     INTO postgresql:///pgloader
     WITH truncate, create table
      SET client_encoding TO 'latin1';
```

Loading CSV data with some *projections*

```
LOAD CSV
     FROM INLINE
     INTO postgresql:///pgloader?nulls (f1, f2, f3)
     WITH truncate,
       keep unquoted blanks,
          fields optionally enclosed by '"',
          fields escaped by double-quote,
          fields terminated by ','

  BEFORE LOAD DO
   $$ drop table if exists nulls; $$,
   $$ create table if not exists nulls
      (id serial, f1 text, f2 text, f3 text);
   $$;
```

Some test data

```
"quoted empty string","","should be empty string"
"no value between separators",,"should be null"
"quoted blanks","    ","should be blanks"
"unquoted blanks",    ,"should be null"
"unquoted string",no quote,"should be 'no quote'"
"quoted separator","a,b,c","should be 'a,b,c'"
"keep extra blanks",  test string  , "should be '  test string
```

Loading several CSV files in a ZIP archive over HTTP, 1/6

```
LOAD ARCHIVE
   FROM /Users/dim/Downloads/GeoLiteCity-latest.zip
   INTO postgresql:///ip4r

   BEFORE LOAD DO
     $$ create extension if not exists ip4r; $$,
     $$ create schema if not exists geolite; $$,
     $$ create table if not exists geolite.location
        (
           locid      integer primary key,
           country    text,
           region     text,
           city       text,
           postalcode text,
           location   point,
           metrocode  text,
           areacode   text
```

Loading several CSV files in a ZIP archive over HTTP, 2/6

```
$$ create table if not exists geolite.blocks
  (
     iprange    ip4r,
     locid      integer
  );
$$,
$$ drop index if exists geolite.blocks_ip4r_idx; $$,
$$ truncate table geolite.blocks, geolite.location casca
```

Loading several CSV files in a ZIP archive over HTTP, 3/6

```
LOAD CSV
     FROM FILENAME MATCHING ~/GeoLiteCity-Location.csv/
          WITH ENCODING iso-8859-1
          (
             locId,
             country,
             region     null if blanks,
             city       null if blanks,
             postalCode null if blanks,
             latitude,
             longitude,
             metroCode  null if blanks,
             areaCode   null if blanks
          )
```

Loading several CSV files in a ZIP archive over HTTP, 4/6

```
INTO postgresql:///ip4r?geolite.location
     (
        locid,country,region,city,postalCode,

        location point
 using (format nil "(~a,~a)" longitude latitude),

        metroCode,areaCode
     )
WITH skip header = 2,
     fields optionally enclosed by '"',
     fields escaped by double-quote,
     fields terminated by ','
```

Loading several CSV files in a ZIP archive over HTTP, 5/6

```
AND LOAD CSV
      FROM FILENAME MATCHING ~/GeoLiteCity-Blocks.csv/
           WITH ENCODING iso-8859-1
           (
              startIpNum, endIpNum, locId
           )
      INTO postgresql:///ip4r?geolite.blocks
           (
              iprange ip4r
                using (ip-range startIpNum endIpNum),
              locId
           )
      WITH skip header = 2,
           fields optionally enclosed by '"',
           fields escaped by double-quote,
           fields terminated by ','
```

Loading several CSV files in a ZIP archive over HTTP, 6/6

```
FINALLY DO
  $$ create index blocks_ip4r_idx
                  on geolite.blocks
               using gist(iprange);
  $$;
```

Loading several CSV files in a ZIP archive over HTTP

1,790,461 rows imported in 18s is about **100,000 rows/s**

table name	imported	errors	time
extract	0	0	1.01
before load	0	0	0.077
geolite.location	438386	0	10.352
geolite.blocks	1790461	0	18.045
finally	0	0	31.108
Total import time	2228847	0	1m0.592s

How pgloader migrates your data

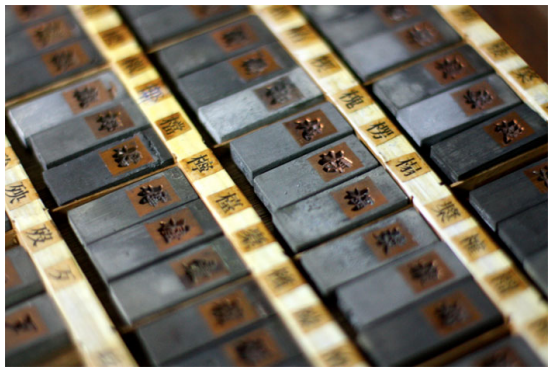
pgloader Architecture Choices and features

- Streaming with the COPY protocol
- Asynchronous IO with Threads
- User Editable Casting Rules
- User provided Transforms Functions

User Editable Casting Rules

User Editable Casting Rules

- Used for *schema* conversion
- Including *default* values handling
- Defines *transform functions*
- Those are applied in the streaming *pipeline*



User Editable Casting Rules

A detailed example

```
type datetime
  to timestamptz
     drop default
     drop not null
     using zero-dates-to-null
```

User Editable Casting Rules

Some use cases

- Data types with different input/ouput behaviour
- `int(11)` actually is a `bigint`
- `auto_increment` means `serial` or `bigserial`
- `datetime` almost quite the same as `timestamptz`
- `NULL` converted as zero dates on `INSERT`
- no proper `boolean`, only `tinyint`
- inline `ENUM` support to `CREATE TYPE`
- geometric datatypes *specific* input/output

How pgloader migrates your data

User provided transformation functions

- On the fly client-side rewritting of values
- Includes rewritting *default* values
- A set of transformation functions is included

User provided transformation functions

Rewriting MySQL data on the fly, when necessary

- `zero-dates-to-null` handles `"0000-00-00"`
- `date-with-no-separator` handles `"20041002152952"`
- `tinyint-to-bolean`
- `convert-mysql-point` expects `astext(column)` output
- `astext` used automatically for `point`

Other necessary transformation

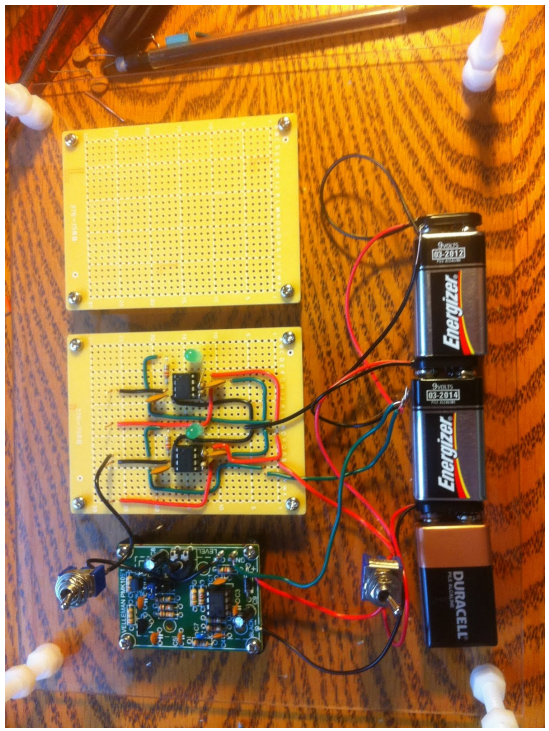
Text like data and `NULL` values

- MySQL returns text `NULL` as an empty string
- pgloader adds `column is null` output
- and process the empty strings depending on `IS NULL` column

pgloader limitations

The 80% rule, Not Implemented Yet are

- Views
- Triggers
- Foreign Keys
- `ON UPDATE CURRENT_TIMESTAMP`
- Geometric datatypes
- Per Column Casting Rules



Questions?

Now is the time to ask!

`http://2013.pgconf.eu/feedback`