Converting MySQL to PostgreSQL

Contents

- 1 Very Short Intro
- 2 Check The Server Running
- 3 Convert and Import
  - 3.1 Common way with SQL dump
  - 3.2 Export using CSV-files
- 4 The Environment
  - 4.1 Perl
- 5 Changing The Code Quick And Dirty
  - 5.1 Perl
  - 5.2 SQL
    - 5.2.1 Syntax
    - 5.2.2 Data Types
    - 5.2.3 Language Constructs
    - 5.2.4 Functions
- 6 Common Errors
- 7 PL/pgSQL
  - 7.1 Install
  - 7.2 Running A Function
  - 7.3 Administration

Very Short Intro

You may have read a bunch of short articles with the same name on the web, but they were just snippets of information you needed. It's time to put it all together.

You have a project in MySQL (http://www.mysql.com/) and suddenly you find out that you need to switch to PostgreSQL (http://www.postgresql.org/). Suddenly you see that there are many flavours of SQL and that your seemingly basic constructions throw a lot of errors. You don't have time to really rewrite your code from scratch, it may come later...

Actually, there may be some good reasons to switch...

- you can sell your product with total peace of mind (PostgreSQL is BSD licensed (http://www.postgresql.org/about/licence), MySQL is more complicated (http://www.mysql.com/company/legal/licensing/))
- you can find articles "Converting from MySQL to PostgreSQL" on the web; you won't find any "Converting from PostgreSQL to MySQL"
- PostgreSQL may not be just another lousy database if Skype (http://jobs.skype.com/2006/02/php_web_based_is_developer.html), Cisco (http://www.cisco.com/en/US/products/sw/voicesw/ps4371/products_user_guide_chapter09186a00800c252c.html), Juniper, IMDb (http://www.imdb.com/help/show_leaf?jobatimdb), Pandora (http://blog.pandora.com/jobs/) decided to rely on it and Sun Microsystems made it database of choice (http://www.oracle.com/technetwork/systems/articles/switching-to-postgres-jsp-138978.html) (which is explicitly funny because Sun acquired MySQL (http://www.mysql.com/news-and-events/sun/)).

With PostgreSQL you may still feel a little like a second-class citizen, but not really the ignored one. There are some major projects like Asterisk (http://www.asterisk.org/), Horde (http://www.horde.org/) or DBMail (http://www.dbmail.org/) that have recognized its qualities and although MySQL was their first choice database, they are showing effort to make things run here too.

Check The Server Running

Most likely you don't need this chapter, but very briefly: after you've installed your package with PostgreSQL on your Linux machine (be it from a package or following these notes (http://scratchpad.wikia.com/wiki/Postgresql)), you need to do something like

```
su -
su - postgres
createdb test
psql test
=# create user username password ' password ';
-- To change a password:
=# alter role username password ' password ';
=# create database databasename with encoding 'utf8';
=# grant all privileges on database databasename to username;
=# \l
=# \c databasename
=# \q
```

vi /etc/postgresql/pg_hba.conf

```
host    all         all         0.0.0.0           0.0.0.0           password
```

be SURE to cover this security issue with iptables!

/etc/init.d/postgresql reload or /usr/lib/postgresql/bin/pg_ctl reload

postmaster successfully signaled

psql -h server -d databasename -U username

databasename=>

Convert and Import

Common way with SQL dump

Dump your tables with

```
mysqldump -u username -p --compatible=postgresql databasename > outputfile.sql
```

but even then you will have to change escaped chars (replacing \t with ^I, \n with ^M, single quote (') with doubled single quote and double (escaped) backslash (\\) with a single backslash). This can't be trivially done with sed command, you may need to write a script for it (Ruby, Perl, etc). There is a MySQL to PostgreSQL python convert script (https://github.com/lanyrd/mysql-postgresql-converter) (you need to use --default-character-set=utf8 when exporting your mysqldump to make it work). It is much better and proven solution to prepend your dump with the following lines

```
SET standard_conforming_strings = 'off';
SET backslash_quote = 'on';
```

These options will force PostgreSQL parser to accept non-ANSI-SQL-compatible escape sequences (Postgre will still issue HINTs on it; you can safely ignore them). Do not set these options globally: this may compromise security of the server!

You also have to manually modify the data types etc. as discussed later.

After you convert your tables, import them the same way you were used to in MySQL, that is

```
psql -h server -d databasename -U username -f data.sql
```

Export using CSV-files

When you have a large sql dump and a binary data inside, it will be uneasy to modify the data structure, so there is another way to export your data to PostgreSQL. Mysql have an option to export each tables from database as separate .sql file with table structure and .txt file with table's data in CSV-format:

```
mysqldump -u username -p -T/path/to/export databasename
```

Notice that /path/to/export should be writeable by user who runs mysqld, in most case it mysqld. After that you should modify your table structure according PostgreSQL format:

- convert data types (http://www.postgresql.org/docs/8.4/static/datatype.html)
- create separate keys definitions
- replace escape characters

When table structure will be ready, you should load it as it was shown above. You should prepare data files: replace carriage return characters to "\r" and remove invalid characters for your data encoding. Here is an example bash script how you can do this and load all the data in your database:

```
#!/bin/bash

CHARSET="utf-8" #your current database charset
DATADIR="/path/to/export"
DBNAME="databasename"

for file in `ls -1 $DATADIR/*.txt`; do
  TMP=${file%.*}
  TABLE=${TMP##*/}
  echo "preparing $TABLE"

  #replace carriage return
  sed 's/\r/\\r/g' $file > /tmp/$TABLE.export.tmp

  #cleanup non-printable and wrong sequences for current charset
  iconv -t $CHARSET -f $CHARSET -c < /tmp/$TABLE.export.tmp > /tmp/$TABLE.export.tmp.out

  echo "loading $TABLE"
  /usr/bin/psql $DBNAME -c "copy $TABLE from '/tmp/$TABLE.export.tmp.out'"

  #clean up
  rm /tmp/$TABLE.export.tmp /tmp/$TABLE.export.tmp.out
done
```

The Environment

Perl

You will need to install an appropriate DBD package. In Debian/Ubuntu run `apt-get install libdbd-pg-perl`.

Changing The Code Quick And Dirty

Perl

MySQL	PostgreSQL	comments
		All you have to do is changing mysql to Pg. Beware the case

`$db=DBI->connect("dbi:mysql:database= ... )`	`$db=DBI->connect("dbi:Pg:database= ... )`	sensitivity.

SQL

Syntax

MySQL	PostgreSQL	comments
`#`	`--`	MySQL accepts nonstandard # to begin a comment line; PostgreSQL uses ANSI standard double dash; use the ANSI standard, both databases understand it. (However, MySQL requires a space after --, whilst it is not mandatory in PostgreSQL)
`" vs. ` `	`' vs. "`	MySQL uses ' or " to quote values (i.e. `WHERE name = "John"`). This is not the ANSI standard for databases. PostgreSQL uses only single quotes for this (i.e. `WHERE name = 'John'`). Double quotes are used to quote system identifiers; field names, table names, etc. (i.e. `WHERE "last name" = 'Smith'`). MySQL uses ` (accent mark or backtick) to quote system identifiers, which is decidedly non-standard. Note: you can make MySQL interpret quotes like PostgreSQL using `SET sql_mode='ANSI_QUOTES'`.
`.. WHERE lastname="smith"`	`.. WHERE lower(lastname)='smith'`	PostgreSQL is case-sensitive for string comparisons. The value 'Smith' is not the same as 'smith'. This is a big change for many users from MySQL (in MySQL, VARCHAR and TEXT columns are case-insensitive unless the "binary" flag is set) and other small database systems, like Microsoft Access. In PostgreSQL, you can either: ▪ Use the correct case in your query. (i.e. `WHERE lastname='Smith'`) ▪ Use a conversion function, like `lower()` to search. (i.e. `WHERE lower(lastname)='smith'`) ▪ Use a case-insensitive operator, like `ILIKE` or `*~`
`LastName` = `lastname` and maybe not?	`"LastName" <> "lastname"`	Database, table, field and columns names in PostgreSQL are case-independent, unless you created them with double-quotes around their name, in which case they are case-sensitive. In MySQL, table names can be case-sensitive or not, depending on which operating system you are using. Note that PostgreSQL actively converts all non-quoted names to lower case and so returns lower case in query results!
`'foo' || 'bar'` means OR	`'foo' || 'bar'` means string concatenation (= 'foobar')	MySQL accepts C-language operators for logic, SQL requires `AND`, `OR`; use the SQL standard keywords for logic, both databases understand it.

Data Types

The ideas for this table were partially derived from automated dump converting script [1] (http://www.xach.com/aolserver/mysql2psql.pl). Official documentation:

- PostgreSQL 8.4 Data Types (http://www.postgresql.org/docs/8.4/static/datatype.html)
- MySQL 5.1 Data Types (http://dev.mysql.com/doc/refman/5.1/en/data-types.html)

List of available data types can be reached also by using psql's internal slash command `\dT`.

MySQL	PostgreSQL	ANSI Standard SQL	comments
TINYINT SMALLINT MEDIUMINT BIGINT	SMALLINT SMALLINT INTEGER BIGINT	INTEGER INTEGER INTEGER NUMERIC(20)	see [2] (http://www.postgresql.o INT); integer size in Pos +2147483647)
TINYINT UNSIGNED SMALLINT UNSIGNED MEDIUMINT UNSIGNED INT UNSIGNED BIGINT UNSIGNED	SMALLINT INTEGER INTEGER BIGINT NUMERIC(20)	INTEGER INTEGER INTEGER NUMERIC(10) NUMERIC(20)	SQL doesn't know UNSIGNED
FLOAT FLOAT UNSIGNED	REAL REAL	FLOAT4 FLOAT4	
DOUBLE	DOUBLE PRECISION	FLOAT8	
BOOLEAN	BOOLEAN	BOOLEAN	MySQL Booleans are an al convert numbers into boo

TINYTEXT TEXT MEDIUMTEXT LONGTEXT	TEXT TEXT TEXT TEXT	TEXT TEXT TEXT TEXT	
BINARY(n) VARBINARY(n) TINYBLOB BLOB MEDIUMBLOB LONGBLOB	BYTEA BYTEA BYTEA BYTEA BYTEA BYTEA	BIT(n) BIT VARYING(n) TEXT TEXT TEXT TEXT	
ZEROFILL	not available	not available	
DATE TIME DATETIME TIMESTAMP	DATE TIME [WITHOUT TIME ZONE] TIMESTAMP [WITHOUT TIME ZONE] TIMESTAMP [WITHOUT TIME ZONE]	DATE TIME TIMESTAMP TIMESTAMP	
column SERIAL equals to: column BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE or: column INT DEFAULT SERIAL equals to: column INT NOT NULL AUTO_INCREMENT UNIQUE	column SERIAL equals to: CREATE SEQUENCE name; CREATE TABLE table (column INTEGER NOT NULL DEFAULT nextval(name));	column GENERATED BY DEFAULT	Note for PostgresSQL: SERIAL = 1 - 2147483647 BIGSERIAL = 1 - 9223372 SERIAL is in fact an ent the rest of your table. dropping a table, you al topic... (http://www.sit Note for MySQL: column SERIAL PRIMARY KEY or column SERIAL, PRIMARY KEY(column) Will result in having 2 the PRIMARY KEY constraint present in the SERIAL ali be corrected.
column ENUM (value1, value2, [...])	column VARCHAR(255) NOT NULL, CHECK (column IN (value1, value2, [...])) or CREATE TYPE mood AS ENUM ('sad','ok','happy'); CREATE TABLE person (current_mood mood ...)	column VARCHAR(255) NOT NULL, CHECK (column IN (value1, value2, [...]))	PostgreSQL doesn't have simulate it with contrai

Language Constructs

- MySQL 5.1 SQL Statement Syntax (http://dev.mysql.com/doc/refman/5.1/en/sql-syntax.html)
- PostgreSQL 8.1 SQL Commands (http://www.postgresql.org/docs/8.1/static/sql-commands.html)
- PostgreSQL 8.2 SQL Commands (http://www.postgresql.org/docs/8.2/static/sql-commands.html)

MySQL	PostgreSQL	comments
	Using psql: \d table or SELECT a.attname AS Field, t.typname || '(' || a.atttypmod || ')' AS Type, CASE WHEN a.attnotnull = 't' THEN 'YES' ELSE 'NO' END AS Null, CASE WHEN r.contype = 'p' THEN 'PRI' ELSE '' END AS Key, (SELECT substring(pg_catalog.pg_get_expr(d.adbin, d.adrelid), '\'(.*)\''')	PostgreSQL doesn't implen

DESCRIBE table	FROM pg_catalog.pg_attrdef d WHERE d.adrelid = a.attrelid AND d.adnum = a.attnum AND a.atthasdef) AS Default, '' as Extras FROM pg_class c JOIN pg_attribute a ON a.attrelid = c.oid JOIN pg_type t ON a.atttypid = t.oid LEFT JOIN pg_catalog.pg_constraint r ON c.oid = r.conrelid AND r.conname = a.attname WHERE c.relname = 'tablename' AND a.attnum > 0 ORDER BY a.attnum	it uses psql's internal s (Be careful: in the mysq shorthand for DROP TABLE)
DROP TABLE IF EXISTS table	DROP TABLE IF EXISTS table	IF EXISTS in DROP TABLE clau since PostgreSQL 8.2.

REPLACE [INTO] table [(column, [...])] VALUES (value, [...])
 	
or
 	
INSERT INTO table
(column1, column2, [...])
VALUES
(value1, value2, [...])
ON DUPLICATE KEY UPDATE
column1 = value1, column2 = value2
	CREATE FUNCTION someplpgsqlfunction() RETURNS void AS $$
	BEGIN
	IF EXISTS(SELECT * FROM phonebook WHERE name = 'john doe') THEN
	UPDATE phonebook
	SET extension = '1234' WHERE name = 'john doe';
	ELSE
	INSERT INTO phonebook VALUES('john doe', '1234');
	END IF;
	
	RETURN;
	END;
	$$ LANGUAGE plpgsql;
		PostgreSQL doesn't implen
		extension. The presented
		PL/pgSQL.
		
		(Note: MySQL REPLACE INTO
		and inserts the new, inst
		place.)

SELECT ... INTO OUTFILE '/var/tmp/outfile'	COPY (SELECT ...) TO '/var/tmp/outfile'	
SHOW DATABASES	Run psql with -l parameter or using psql: \l or SELECT datname AS Database FROM pg_database WHERE datistemplate = 'f'	PostgreSQL doesn't implen
SHOW TABLES	Using psql: \dt or SELECT c.relname AS Tables_in FROM pg_catalog.pg_class c LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace WHERE pg_catalog.pg_table_is_visible(c.oid) AND c.relkind = 'r' AND relname NOT LIKE 'pg_%' ORDER BY 1	PostgreSQL doesn't implen it uses psql's internal s
SELECT ... LIMIT offset, limit or SELECT ... LIMIT limit OFFSET offset	SELECT ... LIMIT limit OFFSET offset	
CREATE TABLE table (column ... , {INDEX|KEY} [name] (column, [...])) or CREATE INDEX name ON table (column, [...])	CREATE INDEX name ON table (column, [...])	
	Using psql:	

`USE database ;`	`\c database`	
`UNLOCK TABLES;`	`-- nothing`	"There is no UNLOCK TABLE always released at transa http://www.postgresql.org lock.html)

Functions

- MySQL 5.1 Functions and Operators (http://dev.mysql.com/doc/refman/5.1/en/functions.html)
- PostgreSQL SQL Functions and Operators (http://www.postgresql.org/docs/current/static/functions.html)
- mysqlcompat, a reimplementation of most MySQL functions in PostgreSQL (http://pgfoundry.org/projects/mysqlcompat/)

MySQL	PostgreSQL	comments

LAST_INSERT_ID() (http://dev.mysql.com/doc/refman/5.1/en/mysql-insert-id.html)	CURRVAL('serial_variable') (http://www.postgresql.org/docs/faqs.FAQ.html#item4.11.2)	NOTE: it is not only "subsitute string" solution as you need to know the name of SERIAL variable (unlike AUTO_INCREMENT in MySQL). Also note that PostgreSQL can play with the OID of the last row inserted by the most recent SQL command.
		
		NOTE2: Even better way to replace LAST_INSERT_ID() is creating a rule, because this cannot suffer from race-conditions:
		
		`CREATE RULE get_{table}_id_seq AS ON INSERT TO {table} DO SELECT currval('{table}_id_seq'::text) AS id;`
		
		(usage is somehow strange, you get a result from an INSERT-statement, but it works very well)
		
		NOTE3: Another, more readable way:
		
		`INSERT INTO mytable VALUES (...) RETURNING my_serial_column_name;`

Common Errors

- ERROR: relation "something" does not exist - usually table doesn't exist as you probably didn't make it with the new datatypes or syntax. Also watch out for case folding issues; PostgreSQL = postgresql != "PostgreSQL".
- prepared statement "dbdpg_X" does not exist -

PL/pgSQL

Install

In versions prior to 9.0, you have to make it available explicitly for every database:

```
your_unix$ su - postgres
your_unix$ .../pgsql/bin/createlang plpgsql -h localhost -d databasename
```

(On BSD systems, the username is pgsql)

Running A Function

```
SELECT definedfunction();
```

Administration

To use the same backup technique as used with MySQL, in `/etc/logrotate.d/postgresql-dumps`:

```
/dumps/postgresql/*/*.dump.gz {
        daily
        rotate 20
        dateext
        nocompress
        sharedscripts
        create
        postrotate
                for i in $(su - postgres -c "psql --list -t" | awk '{print $1}' | grep -vE '^$|^template[0-9]'); do
                        if [ ! -e /dumps/postgresql/$i ]; then mkdir -m 700 /dumps/postgresql/$i; fi
                        # compress even in custom format, because it can be compressed more
                        su - postgres -c "pg_dump --format=custom $i" | gzip > /dumps/postgresql/$i/$i.dump.gz
                done
        endscript
}
/dumps/postgresql/*/*.sql.gz {
        daily
        rotate 20
        dateext
        nocompress
        sharedscripts
        create
        postrotate
                for i in $(su - postgres -c "psql --list -t" | awk '{print $1}' | grep -vE '^$|^template[0-9]'); do
                        if [ ! -e /dumps/postgresql/$i ]; then mkdir -m 700 /dumps/postgresql/$i; fi
                        su - postgres -c "pg_dump --format=plain $i" | gzip > /dumps/postgresql/$i/$i.sql.gz
                done
        endscript
}
/dumps/postgresql/*/*.tar.gz {
        daily
        rotate 20
        dateext
        nocompress
        sharedscripts
        create
        postrotate
                for i in $(su - postgres -c "psql --list -t" | awk '{print $1}' | grep -vE '^$|^template[0-9]'); do
                        if [ ! -e /dumps/postgresql/$i ]; then mkdir -m 700 /dumps/postgresql/$i; fi
                        su - postgres -c "pg_dump --format=tar $i" | gzip > /dumps/postgresql/$i/$i.tar.gz
                done
        endscript
}
```