LOXODATA

MIGRATE TO POSTGRESQL

LET'S MEET THE ELEPHANT

PGCONF.EU - 2019-10-15

Nicolas Lutic & Jean-Christophe Arnu

LOXODATA

HEY, WHAT'S THE WIFI CODE

ESSID : `PGCONFEU`

At Portal login, choose the WIFI-Code link at the bottom of the login box.

Portal password : `SLONIK2109`

Or if you don't believe being in the future try `SLONIK2019`

BEFORE WE START

Do you have  ?

@pgconfeu ← follow

#pgconfeu ← to tweet

LOXODATA

WHO ARE WE ?

WHO

Jean-Christophe Arnu

- PostgreSQL since 1998
- Founder of PostgreSQLFr and PGDay.fr
- PostgreSQL Consultant
- @jcarnu

LOXODATA

WHO ARE WE ?

WHO

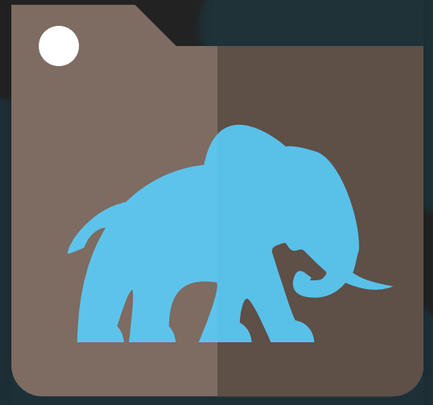
Nicolas Lutic

- ❤ PostgreSQL since 2008
- 💼 PostgreSQL consultant and trainer
- 🐦 @nicolas_lutic

WHO ARE WE ?

LOXODATA

Company built on 3 essential pillars



PostgreSQL

DevOps

Cloud

LOXODATA

WHO ARE WE ?

LOXODATA

A comprehensive service offer

- Architecture
- Consulting
- Teaching
- Administration
- Audit
- Support

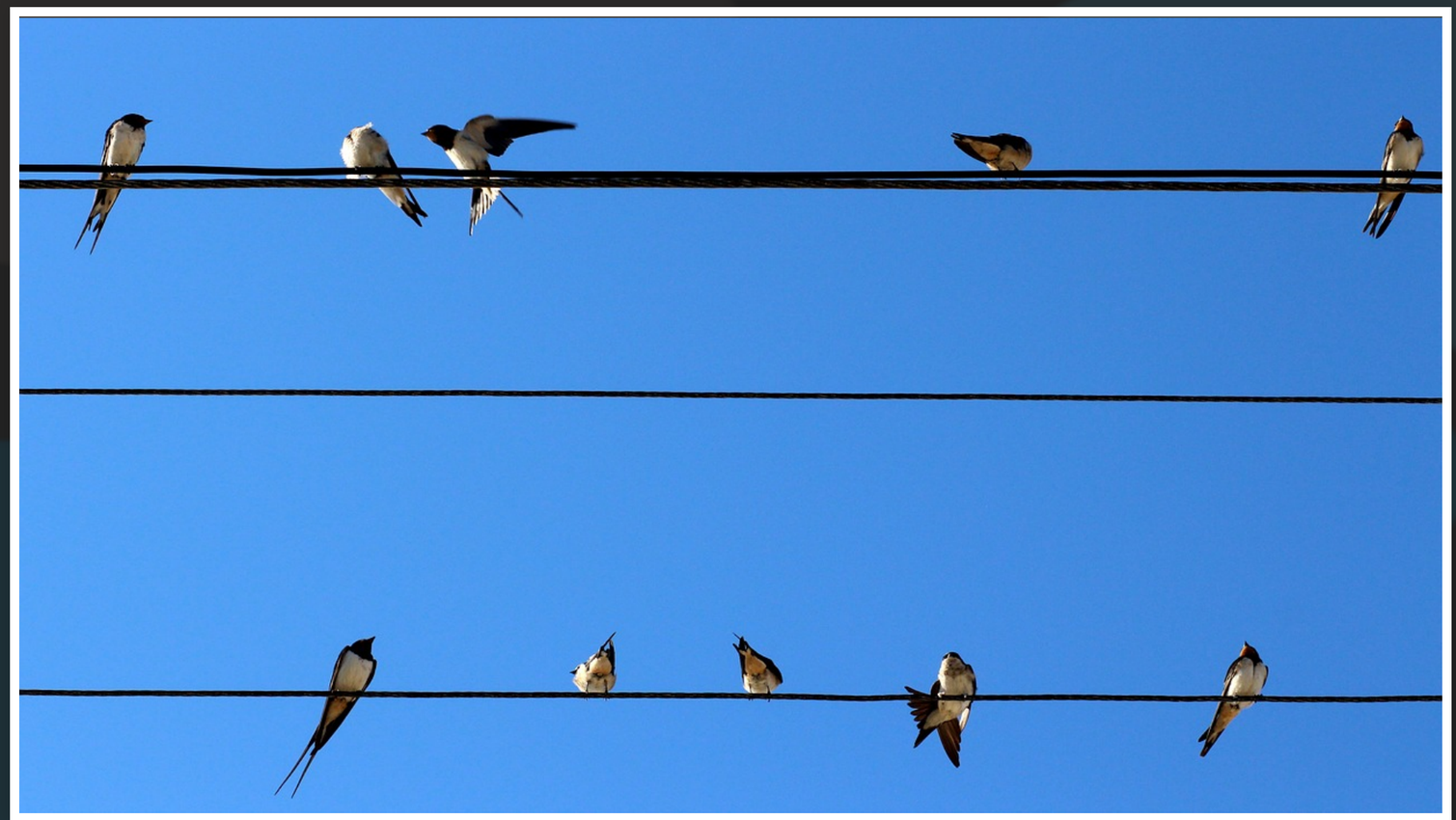
WHAT?! MIGRATING?!

Grey Goose

YEAH! I'M IN!

LOXODATA

LET'S MEET THE ELEPHANT



So are we!!

LOXODATA

TRAINING LAYOUT

This is our path

- Who's involved in the migrations?
- What do I have to know before starting?
- How do I perform a migration?
- Dive into the tech:
  - Tools working for "any" database
  - MariaDB specific
  - Oracle specific

LOXODATA

A HUMAN QUESTION

Yes, it's all about people!

LOXODATA

WHY MIGRATING ?

First of all : **Motivation for the migration**

- Killing costs (licensing) : political
- Killing technological debt
- Major change in application or IT architecture

LOXODATA

WHO HAS DECIDED?

- Who decided that migration:
  - Financial?
  - Technical?
  - Management?
  - Committee?

WHO'S IN CHARGE?

- Who will be in charge of the migration?
  - Company's technical team?
  - Stakeholders? Is there a full breakdown of stakeholders?
- Who was in charge of the current project in the past?
  - Who is currently running the project?
  - Who has created the db design, the architecture?
  - Who was in charge of the app development?
  - Does the database had multiple release in time?
  - Are those people (still) reachable?
- Who decides on technical aspects?
- Who decides on management aspects?
- Who decides on financial aspects?

LOXODATA

WHO'S PART OF THE MIGRATION?

A transversal business view of the above questions:

- Dev? (how will my app work?)
- Ops? (how will the server work?)
- Architects? (how will PG fit on my architecture?)
- IT team? (how will the service be available?)

FITTING IN THE TEAM

- If you are a consultant, you have to understand your customers' business language: vocabulary
- Team project management habits or not
  - SCM
  - Bug tracking
  - CI
- Be sure to be identified by technical officers, managers and financial leaders for migration
- Be sure to be aware of the decisional process

THE PATH

How do we migrate?

LOXODATA

DIGGING INTO THE PROJECT

Focus points:

- Database architecture
- Data workflows
- Database targets
- Side requests
- Best practices

DB ARCHITECTURE

Interfaces:

- With apps
- Data workflows including data
- Data lifecycle
- Maintenance
- Ops interface (HA, backup, virtualization, containerization)
- Scalablity (sharding, logical segmentation)

LOXODATA

DB ARCHITECTURE

How is currently run the service?

- Hardware
- General architecture
- How is implemented HA, backup and other stuffs

WHY?

How were made all the choices

LOXODATA

DATA WORKFLOWS

- Standalone with live maintenance
- Blue-green
  - Schemas
  - Databases
  - Servers

DB TARGET

- Just provide a service
- Heavy load (performance)
- Availability (critical database)

LOXODATA

SIDE REQUESTS

- Optimize data model
- Add/Remove functionnal
- Data workflow changes

BEST PRACTICE IN MIGRATION

Proceed step-by-step:

1. Study source architecture
2. Define migration type
3. Migration prototype with schema + data sample
4. Migrate as is with best suitable mapping
5. Modify types and model
6. Start building Backup+HA for later use
7. Add functionnal
8. Dry-run app testing
9. Add Maintenance and ops
10. Dry run test
11. Final migration

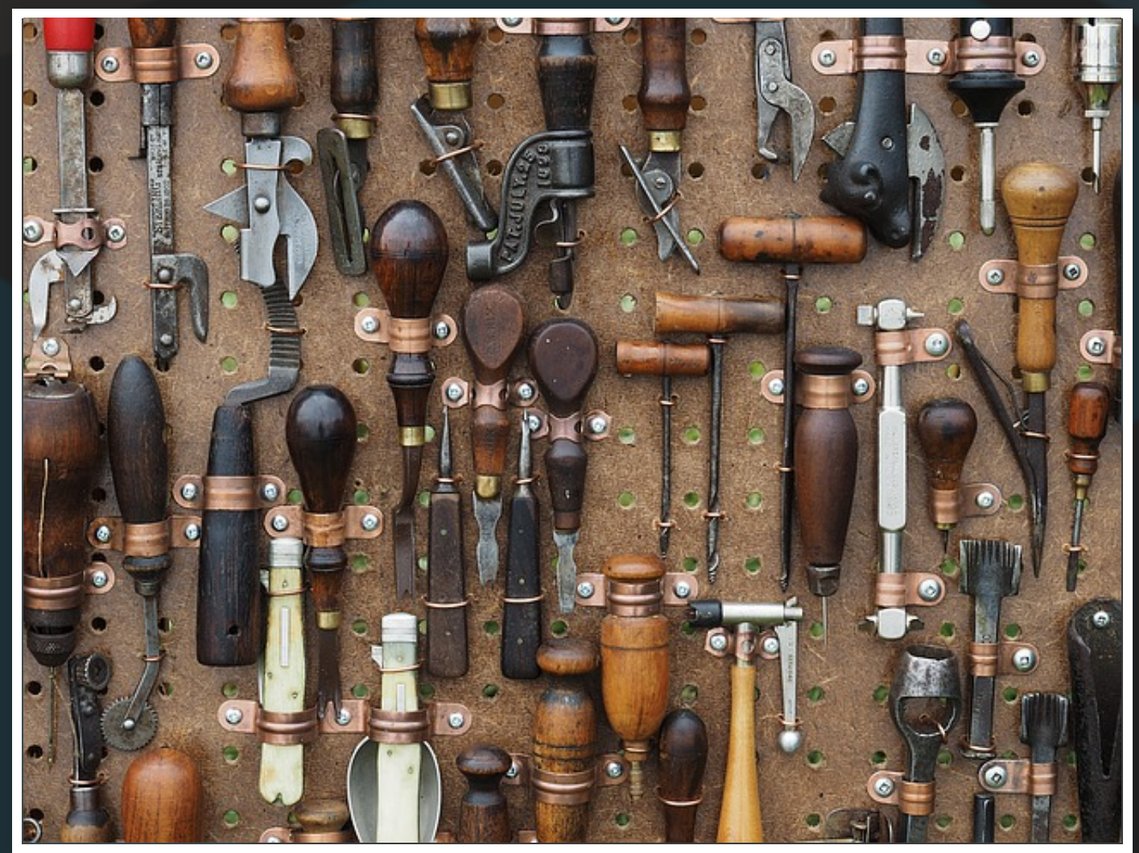
SAKILA DB

The Database we want to migrate: a video store database.

What's difficult with that base?

- Structure:
  - Columns type (i.e. booleans, numeric fixed precision)
  - Constraints
  - Views
- Functional
  - There are functions or procedures (sometimes packages)
  - There are triggers
  - Functional typing

TOOLS AND DATABASE ENGINES



LOXODATA

YOUR VM

SOME WORDS ON YOUR VM

- Virtualbox box created with Vagrant
- CentOS 7 with PGDG rpm source
- PostgreSQL 11

YOUR VM

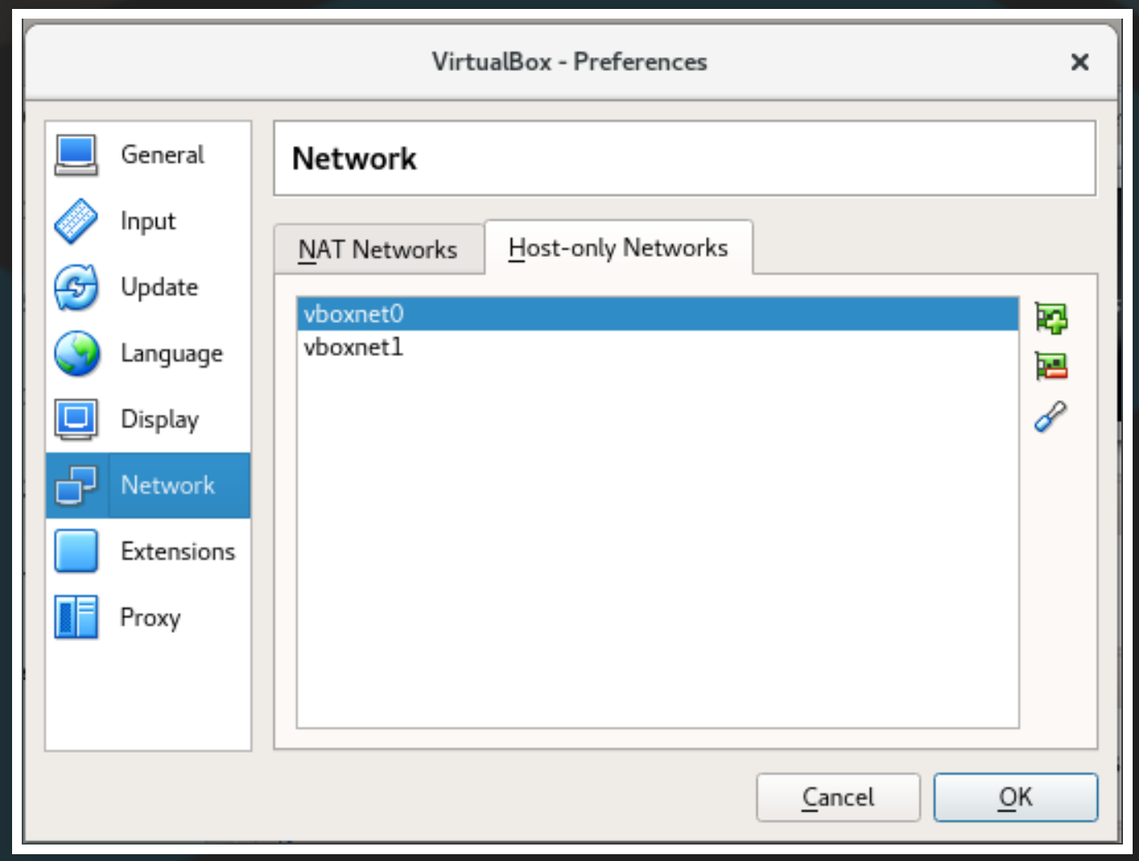
VM IMPORT AND NETWORK INSTALLATION

- `import_m2pg.sh` works on MacOS and Linux with VirtualBox 5.1 and 5.2
- `import_m2pg.ps1` is the Windows PowerShell script tested against 5.2 (should work on 5.1 as well)
- You can also do the job manually (import VM, configure hostonly network)

YOUR VM
VIRTUALBOX CONFIGURATION

Add `vboxnet0` with static IP `10.0.0.1`

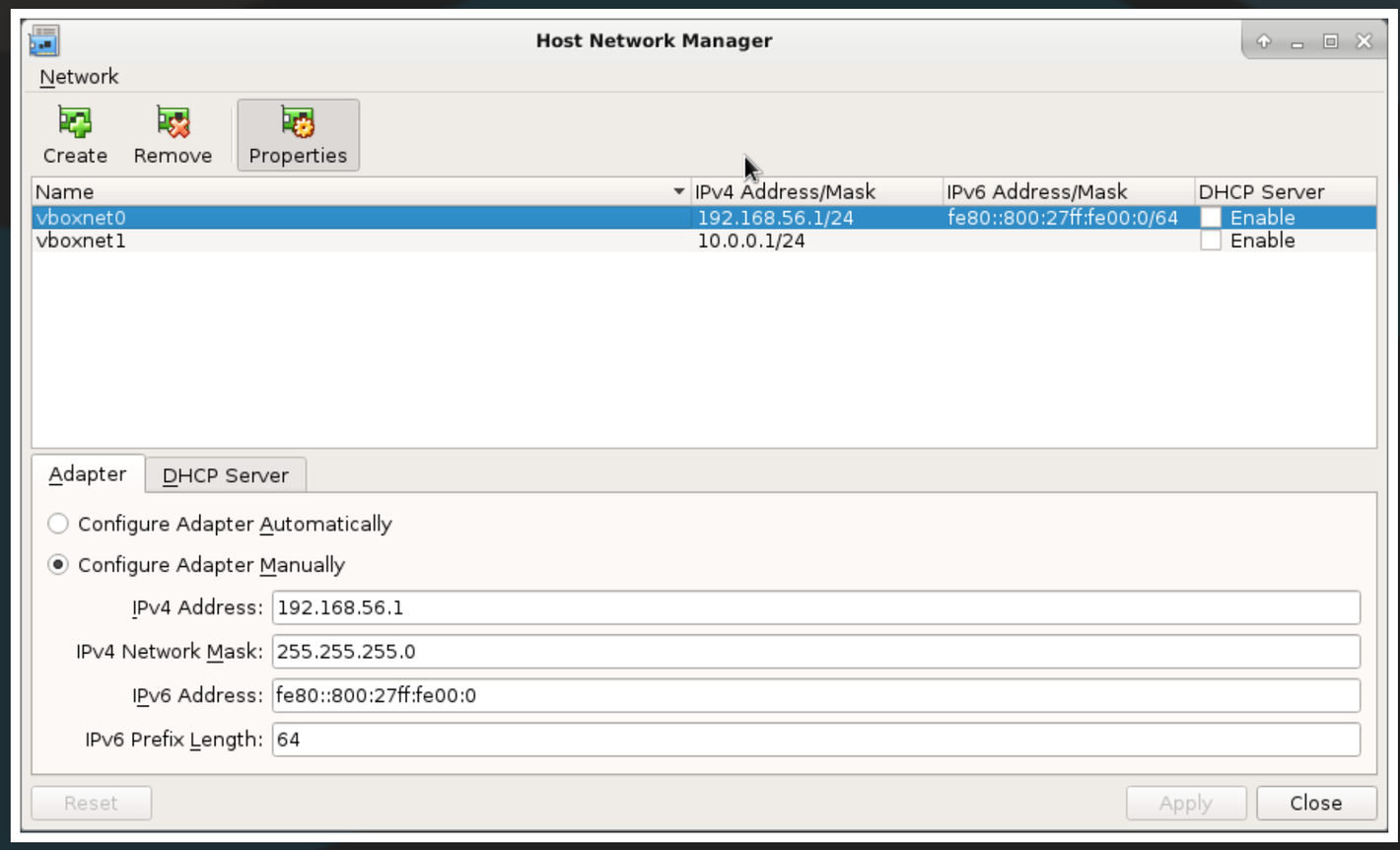
Virtualbox 5.1



YOUR VM
VM INSTALLATION

Add `vboxnet0` with static IP `10.0.0.1`

Virtualbox 5.2



LOXODATA

YOUR VM

MAC USERS

Use ICH6 ide Controller

YOUR VM

SOME WORDS (AGAIN)

- VM were already initialized with vagrant (`provision.sh` in gitlab repo)
- Configuration files in `/var/lib/pgsql11/11/`
- User: `attendee@10.0.0.125` password `migrationTraining19`
- `attendee` is sudoer so you can use `sudo -i -u postgres` (and type `migrationTraining19` again as password to become postgres user)
- All exercises should be run as postgres user

- **DON'T USE FOR PRODUCTION !**
- **DON'T !**

YOUR VM

WHAT'S INSIDE YOUR VM

- PostgreSQL 11
- MariaDB 5.5
- Microsoft SQL Server Express 2019
- Oracle Express 18c
- tds_fdw, ora_fdw, ora_migrator, ora2pg, pgloader, pg_chameleon
- Sakila DB schema and data from JOOQ (a DVD rental store database)
- Each engine/DB is accessible from DBeaver (on the usb stick)
- Better use SSH to connect

LOXODATA

YOUR VM

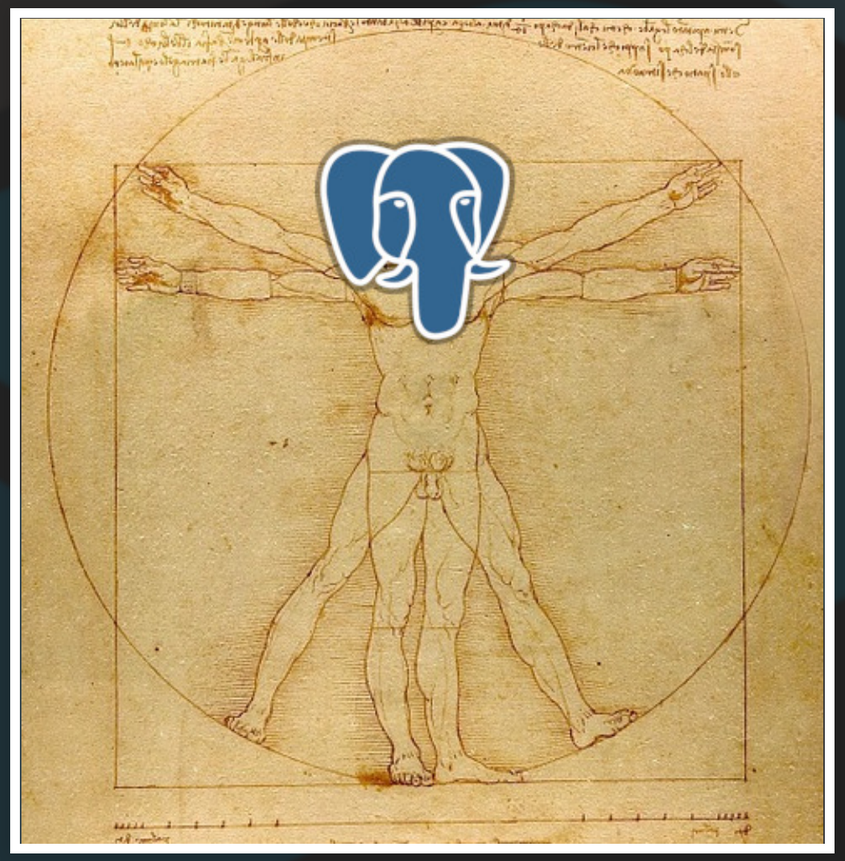
ACCESS YOUR VM

SSH :

- User : attendee
- Password : migrationTraining19
- Host : 10.0.0.125
- `ssh attendee@10.0.0.125`

PAUSE

MULTIPLE ENGINE TOOLS



LOXODATA

FOREIGN DATA WRAPPERS

Foreign Data Wrappers (FDW) make remote table visible into PostgreSQL : SQL/MED

LOXODATA

FDW MIGRATION WORKFLOW

1. Create extension
2. Define a server (a connection)
3. Define a user mapping
4. Import foreign schema
5. No referential integrity
6. No functional migration (functions/procedures/triggers)
7. No views recoding (only data)
8. Query foreign information schema?

WHICH ENGINE CAN I CONNECT

- PostgreSQL
- MySQL/MariaDB
- SQLServer
- Oracle
- many others (sqlite, csv, hadoop, ...)

SO NOW?

```
SELECT 'CREATE TABLE sakila.'|| table_name || ' AS SELECT * FROM foreign_sakila.'||table_name
  FROM  information_schema.tables
  WHERE table_schema = 'foreign_sakila' and table_type = 'FOREIGN' \gexec
```

LOXODATA

MARIADB

- Import MariaDB information_schema?

```
CREATE SCHEMA IF NOT EXISTS foreign_information_schema;
GRANT USAGE ON SCHEMA foreign_information_schema TO attendee;
ALTER SCHEMA foreign_information_schema OWNER TO ATTENDEE;

IMPORT FOREIGN SCHEMA information_schema FROM SERVER mariadb INTO foreign_information_schema;
```

- Easy!

ORACLE

- Import Oracle information_schema?

```
sakila_ora_ora_fdw=> IMPORT FOREIGN SCHEMA information_schema FROM SERVER mariadb INTO foreign_informat
ERROR:  server "mariadb" does not exist
sakila_ora_ora_fdw=> IMPORT FOREIGN SCHEMA information_schema FROM SERVER oradb INTO foreign_informatio
ERROR:  remote schema "information_schema" does not exist
HINT:  Enclose the schema name in double quotes to prevent case folding.
```

- You have to explore the Oracle system catalog to create a matching

SQLSERVER

- Import SQLServer information_schema?

```
sakila_mssql_tds_fdw=> IMPORT FOREIGN SCHEMA information_schema FROM SERVER mssql
INTO foreign_information_schema;
ERROR:  syntax error at or near ")"
LINE 2: ) SERVER mssql
          ^
QUERY:  SELECT t.table_name,  c.column_name,   c.data_type,   c.column_default,   c.is_nullable,
  c.character_maximum_length,   c.numeric_precision,   c.numeric_precision_radix,
  c.numeric_scale,   c.datetime_precision FROM INFORMATION_SCHEMA.TABLES t
  LEFT JOIN INFORMATION_SCHEMA.COLUMNS c ON     t.table_schema = c.table_schema
  AND t.table_name = c.table_name WHERE t.table_type = 'BASE TABLE'
  AND t.table_schema = 'information_schema' ORDER BY t.table_name, c.ordinal_position
) SERVER mssql
OPTIONS (schema_name 'information_schema', table_name '');
```

- This schema exists but you can't explore it
- The returning message is not meaningful

PREPARING FOR MIGRATION (MARIADB)

Example of a query:

```
SELECT
        'CREATE INDEX '|| "INDEX_NAME" ||'_'|| "TABLE_NAME"
        ||' ON '||"TABLE_NAME"
        || ' USING btree ('||string_agg ("COLUMN_NAME",',')||');'
FROM
    foreign_information_schema."STATISTICS"

WHERE
    "TABLE_SCHEMA" = 'sakila'
     AND
    "INDEX_NAME" <> 'PRIMARY'
GROUP BY    "INDEX_NAME" , "TABLE_NAME";
```

IN YOUR VM WITH SAKILA

PostgreSQL database: sakila_mariadb_mysql_fdw

`~attendee/exercises/mariadb/fdw/_migration.sql` contains:

- `FDW creation`
- `Importing schema`
- `FK, indexes, functions, triggers creation`

`migration_helper/` contains some queries which help you prepare the migration

`~attendee/exercises/mariadb/checking_migration` contains some queries which help you compare the foreign and the destination schemata

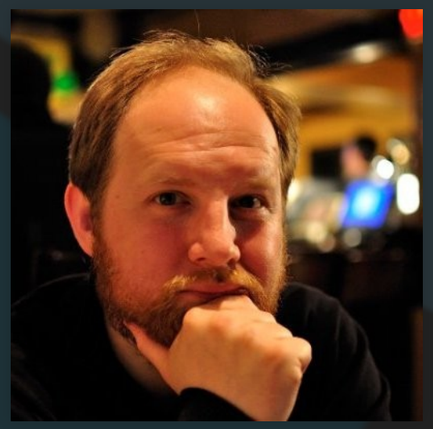
PGLOADER

Pgloader(v3) exists for at least 6 years.

It is written in LISP.

Dimitri Fontaine @tapoueh

https://pgloader.io



(yeah, you know that guy!)
(No? Then welcome to the community!)

FEATURES

Simple and efficient data migration tool with on the fly transform options.

Multiple Databases:

- MySQL
- PostgreSQL
- Microsoft SQLServer
- SQLite

Features :

- Schema discovery
- Schema only migration
- Data only migration

LOXODATA

THE LIMITATIONS

- Views are not migrated
- Triggers are not migrated
- Functions and Procedures are not migrated
- Of the geometric datatypes, only the POINT database has been covered

MIGRATION WORKFLOW

1. Install pgLoader
2. Configure PostgreSQL (create database, user...)
3. Prepare the pgLoader configuration file
4. Execute pgLoader with the schema only mode
5. Analyse the destination schema
6. Write Pre/Post SQL commands and/or ajust the configuration
7. Execute pgLoader
8. **Congrats !**

MAIN GENERAL OPTIONS

```
-v, --verbose

-q, --quiet

-d, --debug Show debug level information messages.

-D, --root-dir Set the root working directory (default to “/tmp/pgloader”).

-L, --logfile Set the pgloader log file (default to “/tmp/pgloader/pgloader.log”).

--log-min-messages
    One of critical, log, error, warning, notice, info or debug.

--client-min-messages
    One of critical, log, error, warning, notice, info or debug.

-S, --summary <filename> To copy the summary output. Support .json, .copy, .csv

--dry-run Test .load file and connexion

--on-error-stop : By default pgLoader ignores the bad data. Useful to debug data processing
```

LOAD FILE

```
LOAD DATABASE
FROM      mysql://pgloader_user:replication19@localhost/sakila
INTO postgresql://attendee:migrationTraining19@localhost:5432/sakila_mariadb_pgloader

WITH snake_case identifiers, drop schema, disable triggers, include drop, create tables,
     create indexes, reset sequences

-- WITH schema only

CAST column staff.active to boolean drop typemod,
    type timestamp with extra on update current timestamp to timestamptz keep default

BEFORE LOAD DO $$ SQL COMMAND $$
AFTER LOAD DO $$ SQL COMMAND $$

-- ALTER TABLE NAMES MATCHING ~/_list$/, 'sales_by_store', ~/sales_by/IN SCHEMA 'dbo' SET SCHEMA 'mv'
-- INCLUDING ONLY TABLE NAMES MATCHING ~/film/, 'actor'
-- EXCLUDING TABLE NAMES MATCHING ~<ory>

-- ALTER TABLE NAMES MATCHING 'film' RENAME TO 'films'
-- ALTER TABLE NAMES MATCHING ~/./ SET (fillfactor='40')
-- ALTER SCHEMA 'sakila' RENAME TO 'pagila'
```

IN YOUR VM WITH SAKILA

Source : **MariaDB**

PostgreSQL database: `sakila_mariadb_pgloader`

Pre-configured skeleton in `~attendee/exercises/mariadb/pgloader/`

`config.load` contains:

- `MariaDB DSN`
- `CAST`
- `Pre-processing`
- `Post-processing`

Start the migration with the command `pgloader config.load`

IN YOUR VM WITH SAKILA

Source : **SQLServer**

PostgreSQL database: `sakila_mssql_pgloader`

Pre-configured skeleton in `~attendee/exercises/mssql/pgloader/`

`config.load` containts:

- `MariaDB DSN`
- `CAST`

Start the migration with the command `pgloader config.load`

BEWARE: TYPING

Take care of your destination types.

Inspect your datamodel

Example:

- Table : `customer`
- Columns : `active`
- SQL Server type : `char(1)`
- PostgreSQL type : `boolean`
- config.load :

```
CAST column customer.active to boolean drop typemod
```

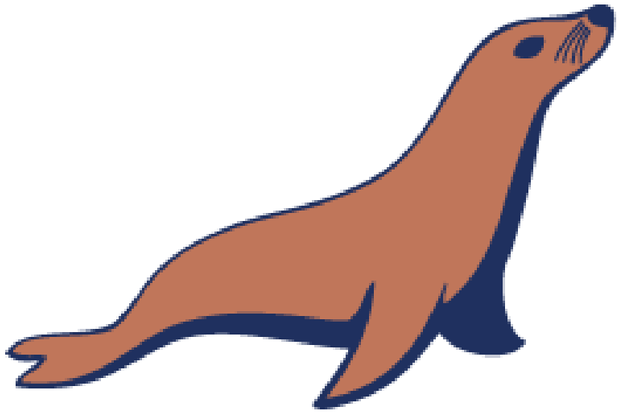
BEWARE: TYPING

Take care of your destination types.

Inspect your datamodel

- Table : `film`
- Columns : `active`
- MariaDB type : `set(Trailers,Commentaries,...)`
- PostgreSQL type : `film_special_features[]`
  `film_special_features`:
  `enum(Trailers,Commentaries,....)`
- 2 solutions :
  - Change the application code to manage text array
  - CAST `set` to `text` and add a data checking in the post processing

MARIADB SPECIFIC



Foundation

PGCHAMELEON

A replication tool from MySQL to PostgreSQL

Written in Python 3.

Version 2 exists since 2017

Federico Campoli
@4thdoctor_scarf
https://pgchameleon.org

FEATURES

- Read from multiple MySQL schemata and restore them into a target PostgreSQL database.
- The source and target schema names can be different.
- Support enumerated and binary data types.
- Data type override.
- Possibilty to refresh single tables or single schema.
- Basic replica monitoring.
- Basic DDL Support (CREATE/DROP/ALTER TABLE, DROP PRIMARY KEY/TRUNCATE, RENAME).

LOXODATA

MIGRATION WORKFLOW (SETUP)

1. Install pgChameleon (need python 3)
2. Enabling Replication for MariaDB (need restart)
3. Configure PostgreSQL (create database, user...)
4. Setting up pgChameleon (connection, schema mapping, type override...)

MIGRATION WORKFLOW

1. Initialize the replication (schema creation, import data)
2. Fix conversion data types (Write pre-initialisation replication SQL file) and/or ajust configuration
3. ReInitialize the replication
4. Write SQL migration (triggers, functions, views, aggregates, indexes)
5. Import SQL without triggers
6. Start replication
7. Check the replication
8. Stop replication
9. Detach source (pgChameleon imports FK and partial constraints)
10. Enable trigger and constraints
11. **Have fun !** (i.e. change app datasource to PG)

MAIN CONFIGURATION

`.pg_chameleon/configuration/mariadb_sakila.yml`

Here are the target PostgreSQL configuration options :

```
pg_conn:
  host: "127.0.0.1"
  port: "5432"
  user: "attendee"
  password: "migrationTraining19"
  database: "sakila_mariadb_pgchameleon"
  charset: "utf8"
```

Here are the source connection options :

```
sources:
  mariadb_sakila:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "replica_user"
      password: "replication19"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      sakila: sakila
    limit_tables:
    skip_tables:
    skip_events:
      insert:
          'sakila.only_insert_table'
```

Here are the type override options :

```
type_override:
  "tinyint(1)":
    override_to: boolean
    override_tables:
      - "*"
"timestamp":
  override_to: timestamptz
  override_tables:
    - "*"
"year(4)":
  override_to: year
  override_tables:
    - "*"
```

IN YOUR VM WITH SAKILA

PostgreSQL database: `sakila_mariadb_pgchamelon`

MariaDB is already configured to replicate.

Skeleton in `~.pg_chameleon/configuration/mariadb_sakila.yml` containts preconfigured

- `MariaDB DSN`
- `CAST`

In `~.pg_chameleon/configuration/pgchameleon_migration_to_postgresql.sh` executes a step-by-step migration

ORACLE SPECIFIC

(sorry no image here)

ORA2PG

This tool is (still) preferred as migration analysis and migration from Oracle.
Produces a good report of objects to be migrated.
So far, one of the best tools to perform.
Written in perl.
Gilles Darold.

ora2pg@darold.net
https://github.com/darold/ora2pg
http://ora2pg.darold.net

FEATURES

- Migrates *a lot* of objects
  - Common objects (tables, views, index) with common types
  - Oracle-spatial to PostGIS
- Tries to migrate functionnal objects
  - Packages, functions, triggers
- Pre-migration report
- Types casting
- Object filtering (tables, columns)
- And many others

LOXODATA

MIGRATION WORKFLOW

1. Install ora2pg (including OCI + Perl Oracle DBD driver to be compiled)
2. Generate migration directory structure
3. Configure migration (access, schema, ...)
4. Migrate to files → PostgreSQL SQL files
5. Read report
6. Tweak SQL files and/or adjust configuration
7. Import SQL structure without constraints/fkey/triggers
8. Import DATA
9. Import triggers, fkeys and contraints
10. **Test your migration**
11. Enjoy (i.e. plug your application)

MAIN CONFIGURATION OPTIONS

`config/ora2pg.conf` is very documented but **a lot of options**.

Here are the main options:

- Access : `ORACLE_DSN`, `SCHEMA`, `ORACLE_{USER,PWD}`
- Exports : `TYPE`
- Modification : `ALLOW`, `EXCLUDE`, `MODIFY_STRUCT` (columns reordering, limiting), `REPLACE_TABLES`
- Content : `WHERE` including/excluding tables/columns; reodering columns
- Structure : `REPLACE_COLS`, `REPLACE_AS_BOOLEAN`, `BOOLEAN_VALUES`, `REPLACE_ZERO_DATE`
- Type : `DATA_TYPE` (mapping oracle_type:pg_type), `MODIFY_TYPE` (table:column:newtype)

LOXODATA

IN YOUR VM

- PostgreSQL database: `sakila_ora_ora2pg`
- Exercise migration walkthrough: `~/exercises/oracle/ora2pg/migrate_basic_and_better.sh`
- 2 preconfigured skeletons in `~attendee/sakila_ora_ora2pg` and `~attendee/sakila_ora_ora2pg_better`

IN YOUR VM

- For each skeleton
  - `config/ora2pg.conf` contains preconfigured `ORACLE_DSN`, `SCHEMA`, `ORACLE_{USER,PWD}`
  - `./export_schema.sh` was already generated and launched (long)
  - Give a look at report in `reports/report.html`
  - List objects to be imported (`-s -t TABLE, SEQUENCE, TYPE, VIEW, PACKAGE`)
  - Migrate structure objects one by one (useful switches : `-x -f -s`)
  - Correct errors (see VIEW, PACKAGE without type and schema)
  - Import data `-a` switch
  - Import triggers `-s -f -x -t TRIGGER`
  - Import constraints `-s -i`

CHECK YOUR MIGRATION

You can ask ora2pg to check your migration and be sure not to forget an object.

```
ora2pg -t TEST -c config/ora2pg.conf | tee dbtest.txt
```

BEWARE : TYPING

Take care of your destination types.

Inspect your data model

Example:

- Table: `customer`
- Columns: `active`
- Oracle type: `char(1)`
- PostgreSQL type: `boolean`
- Ora2pg.conf:

```
REPLACE_AS_BOOLEAN CUSTOMER:ACTIVE
BOOLEAN_VALUES           yes:no y:n 1:0 true:false enabled:disabled Y:N
```

BEWARE: FUNCTIONAL

Function migration seems to be ok but sometimes you'll have:

```
CREATE OR REPLACE FUNCTION public.trigger_fct_rental_before_trigger()
 RETURNS trigger
 LANGUAGE plpgsql
 SECURITY DEFINER
AS $function$
BEGIN
-- Outch!!!! you'll have to change this to default your columns
 IF (NEW.rental_id IS NULL) THEN
   SELECT nextval('rental_sequence') INTO STRICT NEW.rental_id
;
  END IF;
 NEW.last_update:=current_date;
RETURN NEW;
END
$function$
;
```

ORA_MIGRATOR

Cybertec tool based on oracle_fdw (part of a higher level tool called cybertec migrator -UI-)

This is a PostgreSQL extension.

TL;DR: quite complete, very promising.

Written in plpgsql.

Laurenz Albe (and Cybertec)

https://github.com/cybertec-postgresql/ora_migrator

FEATURES

Based on `ora_fdw`, adds *a lot* of migration functions and some high level migration functions.

- This is a migration framework
- Creates 2 schemata
  - Edit type mapping for each individual columns
  - Does not provide a convenient type transform functions (i.e. `char(1)` to `boolean`)
- Does not automatically migrate functionnal items: functions, triggers, packages
  - But offers a convenient way to do so
- Does not attach sequences to tables

MIGRATION WORKFLOW

- Easy :

```
SELECT oracle_migrate(server => 'oradb', only_schemas => '{SAKILA}');
```

- But sometimes things doesn't work :

```
WARNING:  Error creating view sakila.customer_list
DETAIL:  function decode(character, integer, unknown, unknown) does not exist:
```

- Did you really think it's automagical? How do you check your types mapping?
- If you want control, you can perform a **step-by-step migration worflow**.

MIGRATION WORKFLOW (DETAILS)

- Create ora_fdw extension, a foreign server and a user mapping
- Call `select oracle_migrate_prepare(...);`
  - → Oracle *read-only* staging schema with foreign tables: `ora_stage`
  - → PostgreSQL *read-write* staging schema with meta: `pgsql_stage`

```
sakila_ora_ora_migrator=> \dE ora_stage.
                List of relations
  Schema   |     Name      |     Type      |  Owner
-----------+---------------+---------------+----------
 ora_stage | checks        | foreign table | attendee
 ora_stage | column_privs  | foreign table | attendee
 ora_stage | columns       | foreign table | attendee
 ora_stage | foreign_keys  | foreign table | attendee
 ora_stage | func_src      | foreign table | attendee
 ora_stage | index_exp     | foreign table | attendee
 ora_stage | keys          | foreign table | attendee
 ora_stage | pack_src      | foreign table | attendee
 ora_stage | schemas       | foreign table | attendee
 ora_stage | segments      | foreign table | attendee
 ora_stage | sequences     | foreign table | attendee
 ora_stage | table_privs   | foreign table | attendee
 ora_stage | tables        | foreign table | attendee
 ora_stage | trig          | foreign table | attendee
 ora_stage | views         | foreign table | attendee
(15 rows)
```

MIGRATION WORKFLOW PGSQL_STAGE

```
sakila_ora_ora_migrator=> \dt pgsql_stage.*
                 List of relations
   Schema    |       Name        | Type  |  Owner
-------------+-------------------+-------+----------
 pgsql_stage | checks            | table | attendee
 pgsql_stage | column_privs      | table | attendee
 pgsql_stage | columns           | table | attendee
 pgsql_stage | foreign_keys      | table | attendee
 pgsql_stage | functions         | table | attendee
 pgsql_stage | index_columns     | table | attendee
 pgsql_stage | keys              | table | attendee
 pgsql_stage | migrate_log       | table | attendee
 pgsql_stage | packages          | table | attendee
 pgsql_stage | schemas           | table | attendee
 pgsql_stage | sequences         | table | attendee
 pgsql_stage | table_privs       | table | attendee
 pgsql_stage | tables            | table | attendee
 pgsql_stage | test_error        | table | attendee
 pgsql_stage | test_error_stats  | table | attendee
 pgsql_stage | triggers          | table | attendee
 pgsql_stage | views             | table | attendee
(17 rows)
```

MIGRATION WORKFLOW

- Just after `oracle_migration_prepare(...)`
  - Modify column data types → `pgsql_stage.columns`;
  - Check what should be migrated: `migrate` column in `tables`, `checks`, `triggers`, `functions`, `views`, `foreign_keys` and `keys`.
  - If Oracle definitions changes → refresh PostgreSQL staging schema with `oracle_migrate_refresh()`.

MIGRATION WORKFLOW

- Evaluate migration effort with `oracle_code_count()`
  - Oracle object type (views, triggers, packages and functions)
  - Object count
  - Object code lines
  - Object definition byte count

```
select oracle_code_count();
   oracle_code_count
-----------------------
 (function,0,0,0)
 (trigger,30,143,2851)
 (package,2,269,6030)
 (view,5,59,2328)
(4 rows)
```

MIGRATION WORKFLOW

- Call `oracle_migrate_mkforeign` to create the PostgreSQL schemas and sequences and foreign tables.
- Call `oracle_migrate_tables` to replace the foreign tables with real tables and migrate the data.
  - You may not want to migrate data? → `with_data => 'FALSE'`

MIGRATION WORKFLOW

- Review triggers/function/packages 👁 `migrate` field
  - Packages are not migrated (but listed) → move to `functions`
  - Call `oracle_migrate_functions` for functions → `pgsql_stage.functions`
  - Call `oracle_migrate_triggers` for triggers → `pgsql_stage.triggers`
- Call `oracle_migrate_views` for views
- Call `oracle_migrate_constraints` for sontraints & indexes
- Call `oracle_migrate_finish` to remove the staging schemas and complete the migration

IN YOUR VM WITH SAKILA

- PostgreSQL database: `sakila_ora_ora_migrator`
- Database preconfigured with `oracle_fdw`, `ora_migrator` extensions and foreign server
- Exercise: `~/exercises/oracle/ora_migrator/00_oramigrator.sql`
- Walkthrough migration without type finalization
- Commented.

LOXODATA

WRAP-UP

LOXODATA

IF WE ARE HERE

That means we all are skilled.

Reviewing so much tools and workflow in only half a day is great!

Congrats!

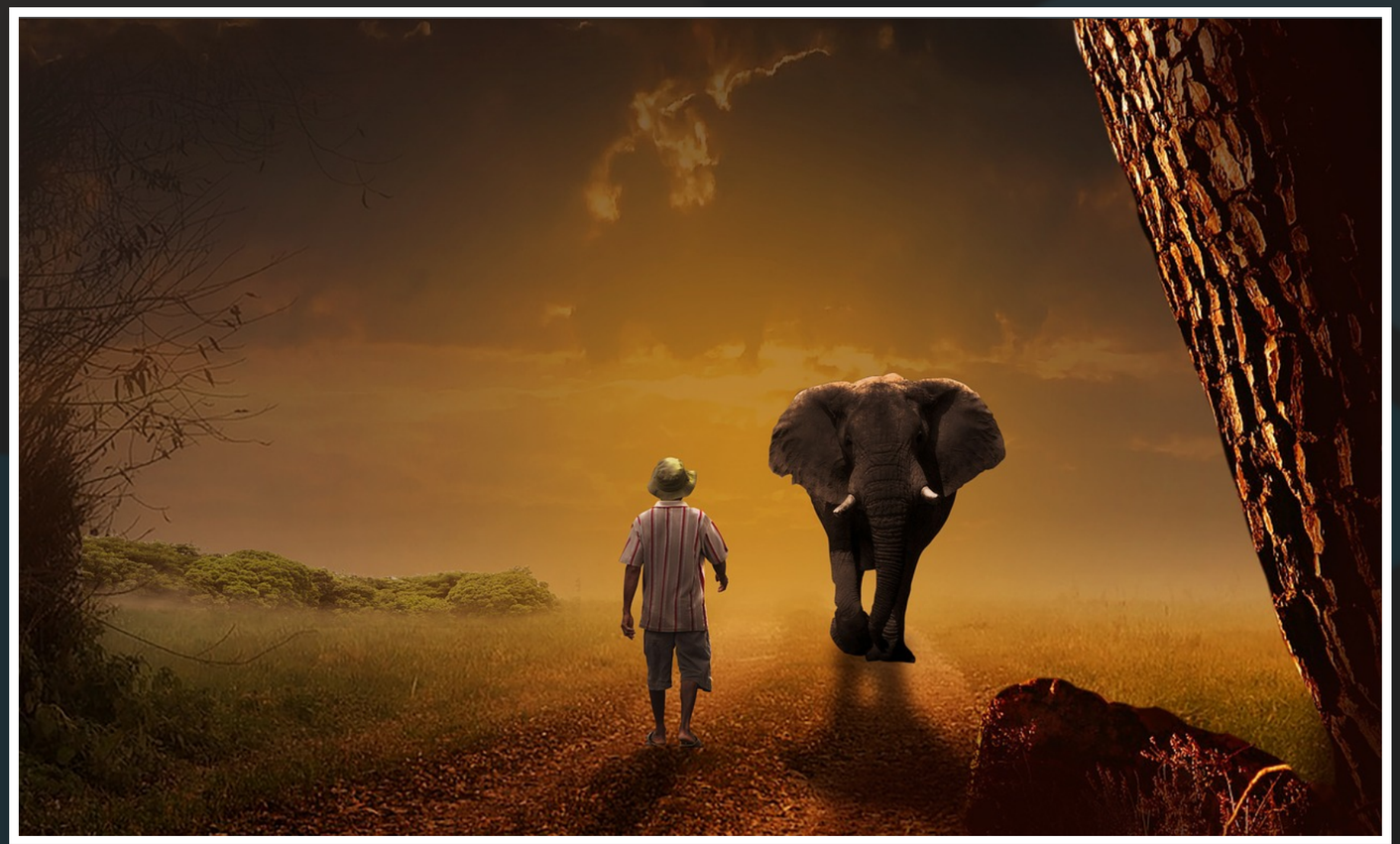
HOW SYNTHETIC ?

Tool	DBMS	S	C	F/P	T	V/MV	U	G	Te
FDW	All	✓	✗	✗	✗	✗	✗	✗	✗
Pgloader	All but Oracle	✓	✓	✗	✓	✓	✗	✗	✗
PgChameleon (*)	MySQL	✓	✓	✗	✗	✗	✗	✗	✗
Ora2pg	Oracle	✓	✓	✓	✓	✓	✓	✓	✓
ora_migrator	Oracle	✓	✓	✓	✓	✓	✗	✓	✓

S: Schema, C: Constraints, F/P: Functions/Procedures, T: Triggers, V/MV: Views/materialized views, U: users, G: Grants, Te: Migration testing

(*) The only one to stream data allowing very small downtime while migrating

THAT'S ALL!



(for today's session)

SLIDES AND SOURCES

- Slides will be available on the pgconf.eu website by the end of the conference.
- Sources are available at : https://gitlab.com/loxo-trainings/migrate-to-pg

LOXODATA

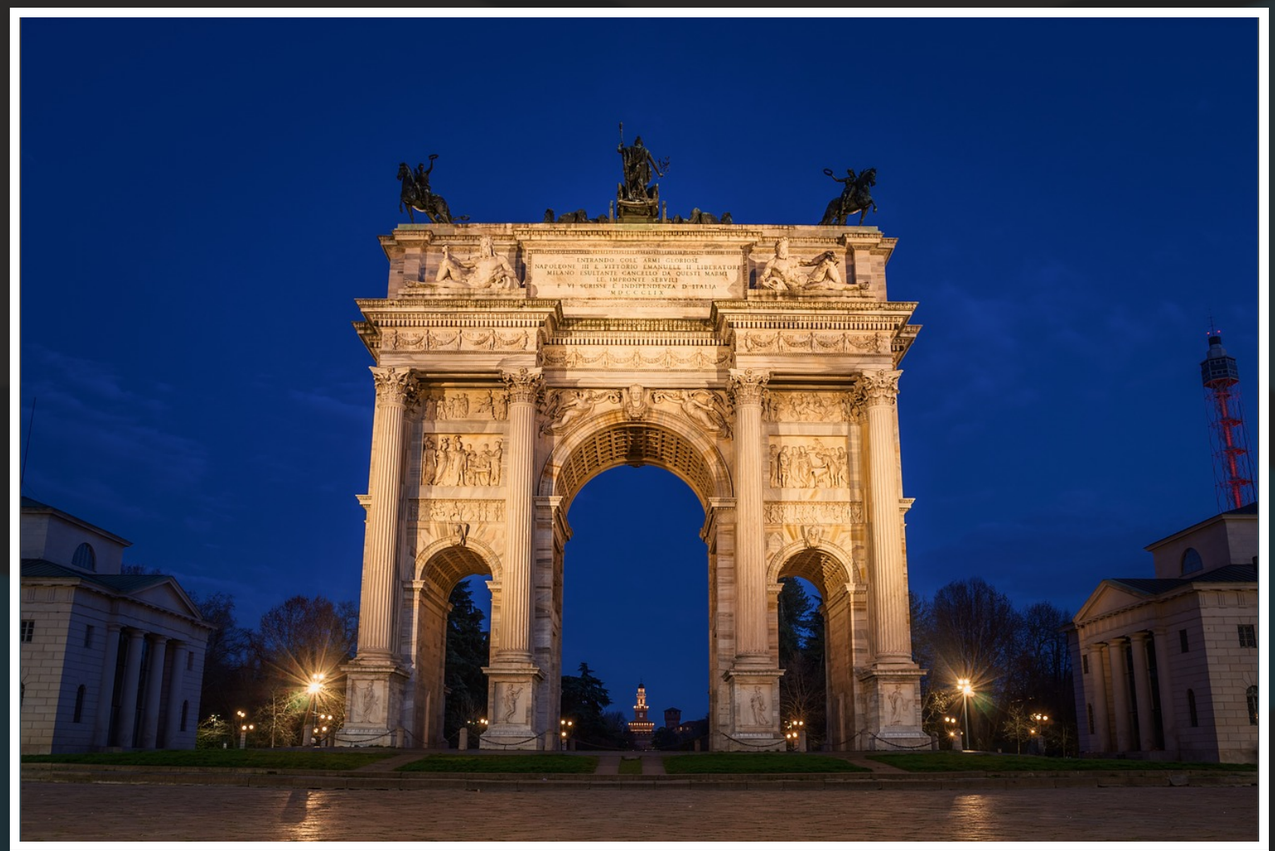
DON'T FORGET

Feedbacks (training+conference) :

https://2019.pgconf.eu/f

LOXODATA

THANK YOU!



For any further question:

n.lutic@loxodata.com or jc.arnu@loxodata.com