pg_chameleon Documentation

Release v2.0.19

Federico Campoli

Mar 26, 2023

Contents

pg_chameleon is a replication tool from MySQL to PostgreSQL developed in Python 3.5+ The system use the library mysql-replication to pull the row images from MySQL which are transformed into a jsonb object. A pl/pgsql function decodes the jsonb and replays the changes into the PostgreSQL database.

The tool requires an initial replica setup which pulls the data from MySQL in read only mode.

pg_chameleon can pull the data from a cascading replica when the MySQL slave is configured with log-slave-updates.

Documentation available at pgchameleon.org

Release available via pypi

CHAPTER 1

FEATURES

- Replicates multiple MySQL schemas within the same MySQL cluster into a target PostgreSQL database. The source and target schema names can be different.
- Conservative approach to the replica. Tables which generate errors are automatically excluded from the replica.
- Daemonised init_replica,refresh_schema,sync_tables processes.
- Daemonised replica process with two separated subprocess, one for the read and one for the replay.
- Soft replica initialisation. The tables are locked when needed and stored with their log coordinates. The replica damon will put the database in a consistent status gradually.
- Rollbar integration for a simpler error detection and alerting.

CHAPTER 2

CHANGELOG

2.1 changelog

2.1.1 2.0.19 - XX March 2023

- Merge pull request #144, mysql-replication support for PyMySQL>0.10.0 was introduced in v0.22
- add support for fillfactor when running init_replica
- improve logging on discarded rows
- add distinct on group concat when collecting foreign keys
- use mysql-replication>=0.31, fix for crash when replicating from MariaDB

2.1.2 2.0.18 - 31 March 2022

- Support the ON DELETE and ON UPDATE clause when creating the foreign keys in PostgreSQL
- change logic for index and foreign key names by managing only duplicates within same schema
- use mysql-replication<0.27 as new versions crash when receiving queries
- add copy_schema method for copying only the schema without data (EXPERIMENTAL)
- change type for identifiers in replica schema to varchar(64)

2.1.3 2.0.17 - 30 January 2022

- Remove argparse from the requirements
- Add the collect for unique constraints when keep_existing_schema is **Yes**
- Fix wrong order in copy data/create indices when keep_existing_schema is **No**
- Remove check for log_bin we are replicating from Aurora MySQL

- Manage different the different behaviour in pyyaml to allow pg_chameleon to be installed as rpm in centos 7

2.1.4 2.0.16 - 23 September 2020

- Fix for issue #126 init_replica failure with tables on transactional engine and invalid data

2.1.5 2.0.15 - 20 September 2020

- Support for reduced lock if MySQL engine is transactional, thanks to @rascalDan
- setup.py now requires python-mysql-replication to version 0.22 which adds support for PyMySQL >=0.10.0
- removed PyMySQL requirement <0.10.0 from setup.py
- prevent pg_chameleon to run as root

2.1.6 2.0.14 - 26 July 2020

- Add support for spatial data types (requires postgis installed on the target database)
- When `keep_existing_schema` is set to `yes` now drops and recreates indices, and constraints during the `init_replica` process
- Fix for issue #115 thanks to @porshkevich
- setup.py now forces PyMySQL to version <0.10.0 because it breaks the python-mysql-replication library (issue #117)

2.1.7 2.0.13 - 05 July 2020

- **EXPERIMENTAL** support for Point datatype - @jovankricka-everon
- Add `keep_existing_schema` in MySQL source type to keep the existing scema in place instead of rebuilding it from the mysql source
- Change tabs to spaces in code

2.1.8 2.0.12 - 11 Dec 2019

- Fixes for issue #96 thanks to @daniel-qcode
- Change for configuration and SQL files location
- Package can build now as source and wheel
- The minimum python requirements now is 3.5

2.1.9 2.0.11 - 25 Oct 2019

- Fix wrong formatting for yaml example files. @rebtoor
- Make start_replica run in foreground when log_file == stdout . @clifff
- Travis seems to break down constantly, Disable the CI until a fix is found. Evaluate to use a different CI.
- Add the add loader to yaml.load as required by the new PyYAML version.

2.1.10 2.0.10 - 01 Sep 2018

- Fix regression in new replay function with PostgreSQL 10
- Convert to string the dictionary entries pulled from a json field
- Let `enable_replica` to disable any leftover maintenance flag
- Add capture in CHANGE for tables in the form schema.table

2.1.11 2.0.9 - 19 Aug 2018

- Fix wrong check for the next auto maintenance run if the maintenance wasn't run before
- Improve the replay function's speed
- Remove blocking from the GTID operational mode

2.1.12 2.0.8 - 14 Jul 2018

- Add support for skip events as requested in issue #76. Is now possible to skip events (insert,delete,update) for single tables or for entire schemas.
- **EXPERIMENTAL** support for the GTID. When configured on MySQL or Percona server pg_chameleon will use the GTID to auto position the replica stream. Mariadb is not supported by this change.
- ALTER TABLE RENAME is now correctly parsed and executed
- Add horrible hack to ALTER TABLE MODIFY. Previously modify with default values would parse wrongly and fail when translating to PostgreSQL dialect
- Disable erroring the source when running with `--debug` switch enabled
- Add cleanup for logged events when refreshing schema and syncing tables. previously spurious logged events could lead to primary key violations when syncing single tables or refreshing single schemas.

2.1.13 2.0.7 - 19 May 2018

- Fix for issue #71, make the multiprocess logging safe. Now each replica process logs in a separate file
- Fix the `--full` option to store true instead of false. Previously the option had no effect.
- Add *auto_maintenance* optional parameter to trigger a vacuum over the log tables after a specific timeout
- Fix for issue #75, avoid the wrong conversion to string for None keys when cleaning up malformed rows during the init replica and replica process
- Fix for issue #73, fix for wrong data type tokenisation when an alter table adds a column with options (e.g. `ADD COLUMN foo DEFAULT NULL`)
- Fix wrong TRUNCATE TABLE tokenisation if the statement specifies the table with the schema.

2.1.14 2.0.6 - 29 April 2018

- fix for issue #69 add source's optional parameter `on_error_read:` to allow the read process to continue in case of connection issues with the source database (e.g. MySQL in maintenance)
- remove the detach partition during the maintenance process as this proved to be a very fragile approach

- add switch `--full` to run a `VACUUM FULL` during the maintenance
- when running the maintentenance execute a `VACUUM` instead of a `VACUUM FULL`
- fix for issue #68. fallback to `binlog_row_image=FULL` if the parameter is missing in mysql 5.5.
- add cleanup for default value `NOW()` when adding a new column with `ALTER TABLE`
- allow `enable_replica` to reset the source status in the case of a catalogue version mismatch

2.1.15 2.0.5 - 25 March 2018

- fix wrong exclusion when running sync_tables with limit_tables set
- add *run_maintenance* command to perform a VACUUM FULL on the source's log tables
- add *stop_all_replicas* command to stop all the running sources within the target postgresql database

2.1.16 2.0.4 - 04 March 2018

- Fix regression added in 2.0.3 when handling MODIFY DDL
- Improved handling of dropped columns during the replica

2.1.17 2.0.3 - 11 February 2018

- fix regression added by commit 8c09ccb. when ALTER TABLE ADD COLUMN is in the form datatype DEFAULT (NOT) NULL the parser captures two words instead of one
- Improve the speed of the cleanup on startup deleting only for the source's log tables instead of the parent table
- fix for issue #63. change the field i_binlog_position to bigint in order to avoid an integer overflow error when the binlog is largher than 2 GB.
- change to psycopg2-binary in install_requires. This change will ensure the psycopg2 will install using the wheel package when available.
- add upgrade_catalogue_v20 for minor schema upgrades

2.1.18 2.0.2 - 21 January 2018

- Fix for issue #61, missing post replay cleanup for processed batches.
- add private method `_swap_enums` to the class `pg_engine` which moves the enumerated types from the loading to the destination schema.

2.1.19 2.0.1 - 14 January 2018

- Fix for issue #58. Improve the read replica performance by filtering the row images when `limit_tables/skip_tables` are set.
- Make the `read_replica_stream` method private.
- Fix read replica crash if in alter table a column was defined as `character varying`

2.1.20 2.0.0 - 01 January 2018

- Add option `--rollbar-level` to set the maximum level for the messages to be sent to rollbar. Accepted values: "critical", "error", "warning", "info". The Default is "info".
- Add command `enable_replica` used to reset the replica status in case of error or unespected crash
- Add script alias `chameleon` along with `chameleon.py`

2.1.21 2.0.0.rc1 - 24 December 2017

- Fix for issue #52, When adding a unique key the table's creation fails because of the NULLable field
- Add check for the MySQL configuration when initialising or refreshing replicated entities
- Add class rollbar_notifier for simpler message management
- Add end of init_replica,refresh_schema,sync_tables notification to rollbar
- Allow `--tables disabled` when syncing the tables to re synchronise all the tables excluded from the replica

2.1.22 2.0.0.beta1 - 10 December 2017

- fix a race condition where an unrelated DDL can cause the collected binlog rows to be added several times to the log_table
- fix regression in write ddl caused by the change of private method
- fix wrong ddl parsing when a column definition is surrounded by parentheses e.g. `ALTER TABLE foo ADD COLUMN(bar varchar(30));`
- error handling for wrong table names, wrong schema names, wrong source name and wrong commands
- init_replica for source pgsql now can read from an hot standby but the copy is not consistent
- init_replica for source pgsql adds "replicated tables" for better show_status display
- check if the source is registered when running commands that require a source name

2.1.23 2.0.0.alpha3 - 03 December 2017

- Remove limit_tables from binlogreader initialisation, as we can read from multiple schemas we should only exclude the tables not limit
- Fix wrong formatting for default value when altering a field
- Add upgrade procedure from version 1.8.2 to 2.0
- Improve error logging and table exclusion in replay function
- Add stack trace capture to the rollbar and log message when one of the replica daemon crash
- Add `on_error_replay` to set whether the replay process should skip the tables or exit on error
- Add init_replica support for source type pgsql (EXPERIMENTAL)

2.1.24 2.0.0.alpha2 - 18 November 2017

- Fix wrong position when determining the destination schema in read_replica_stream
- Fix wrong log position stored in the source's high watermark
- Fix wrong table inclusion/exclusion in read_replica_steam
- Add source parameter `replay_max_rows` to set the amount of rows to replay. Previously the value was set by `replica_batch_size`
- Fix crash when an alter table affected a table not replicated
- Fixed issue with alter table during the drop/set default for the column (thanks to psycopg2's sql.Identifier)
- add type display to source status
- Add fix for issue #33 cleanup NUL markers from the rows before trying to insert them in PostgreSQL
- Fix broken save_discarded_row
- Add more detail to show_status when specifying the source with –source
- Changed some methods to private
- ensure the match for the alter table's commands are enclosed by word boundaries
- add if exists when trying to drop the table in swap tables. previously adding a new table failed because the table wasn't there
- fix wrong drop enum type when adding a new field
- add log error for storing the errors generated during the replay
- add not functional class pgsql_source for source type pgsql
- allow `type_override` to be empty
- add show_status command for displaying the log error entries
- add separate logs for per source
- change log line formatting inspired by the super clean look in pgbackrest (thanks you guys)

2.1.25 2.0.0.alpha1 - 11 November 2017

- Python 3 only development
- Add support for reading from multiple MySQL schemas and restore them it into a target PostgreSQL database. The source and target schema names can be different.
- Conservative approach to the replica. Tables which generate errors are automatically excluded from the replica.
- Daemonised init_replica process.
- Daemonised replica process with two separated subprocess, one for the read and one for the replay.
- Soft replica initialisation. The tables are locked when needed and stored with their log coordinates. The replica damon will put the database in a consistent status gradually.
- Rollbar integration for a simpler error detection.

CHAPTER 3

RELEASE NOTES

3.1 RELEASE NOTES

3.1.1 2.0.19

This maintenance release adds the following bugfix and improvements.

Merge pull request #144 adding mysql-replication support for PyMySQL>0.10.0 was introduced in v0.22 Adds support for fillfactor when running init_replica, it's now possible to specify the fillfactor for the tables when running init_replica. Useful to mitigate bloat in advance when replicating/migrating from MySQL.

Improve logging on discarded rows, now the discarded row image is displayed in the log.

Add distinct on group concat when collecting foreign keys metadata to avoid duplicate fields in the foreign key definition.

Use mysql-replication>=0.31, this fix the crash when replicating from MariaDB introduced in mysql-replication 0.27

3.1.2 2.0.18

This maintenance release adds the following bugfix and improvements.

Adds a new method *copy_schema* to copy only the schema without the data (EXPERIMENTAL).

Adds the support for the **ON DELETE** and **ON UPDATE** clause when creating the foreign keys in PostgreSQL with *detach_replica* and *copy_schema*.

When running *init_replica* or *copy_schema* the names for the indices and foreign keys are preserved. Only if there is any duplicate name then pg_chameleon will ensure that the names on PostgreSQL are unique within the same schema.

Adds a workaround for a regression introduced in **mysql-replication** by forcing the version to be lesser than 0.27.

Change the data type for the identifiers stored into the replica schema to varchar(64)

This release requires a replica catalogue upgrade, therefore is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to use screen or tmux for the upgrade
- Stop all the replica processes with `chameleon stop_all_replicas --config <your_config>`
- Take a backup of the schema `sch_chameleon` with pg_dump as a good measure.
- Install the upgrade with `pip install pg_chameleon --upgrade`
- Check if the version is upgraded with `chameleon --version`
- Upgrade the replica schema with the command `chameleon upgrade_replica_schema --config <your_config>`
- Start all the replicas.

3.1.3 2.0.17

This maintenance release adds the following bugfix.

Fix the wrong order in copy data/create indices when keep_existing_schema is **No**.

Previously the indices were created before the data was loaded into the target schema with great performance degradation.

This fix applies only if the parameter keep_existing_schema is set to **No**.

Add the collect for unique constraints when keep_existing_schema is **Yes**.

Previously the unique constraint were not collected or dropped if defined as constraints instead of indices.

This fix applies only if the parameter keep_existing_schema is set to **Yes**.

This release adds the following changes:

- Remove argparse from the requirements as now it's part of the python3 core dist
- Remove check for log_bin when we replicate from Aurora MySQL
- Manage different the different behaviour in pyyaml to allow pg_chameleon to be installed as rpm in centos 7 via pgdg repository

This release works with Aurora MySQL. However Aurora MySQL 5.6 segfaults when FLUSH TABLES WITH READ LOCK is issued.

The replica is tested on Aurora MySQL 5.7.

This release requires a replica catalogue upgrade, therefore is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to use screen or tmux for the upgrade
- Stop all the replica processes with `chameleon stop_all_replicas --config <your_config>`
- Take a backup of the schema `sch_chameleon` with pg_dump as a good measure.
- Install the upgrade with `pip install pg_chameleon --upgrade`
- Check if the version is upgraded with `chameleon --version`
- Upgrade the replica schema with the command `chameleon upgrade_replica_schema --config <your_config>`
- Start all the replicas.

3.1.4 2.0.16

This maintenance release fix a crash in init_replica caused by an early disconnection during the fallback on insert. This caused the end of transaction to crash aborting the init_replica entirely.

3.1.5 2.0.15

This maintenance release adds the support for reduced lock if MySQL engine is transactional, thanks to @rascalDan.

The init_replica process checks whether the engine for the table is transactional and runs the initial copy within a transaction. The process still requires a FLUSH TABLES WITH READ LOCK but the lock is released as soon as the transaction snapshot is acquired. This improvement allows pg_chameleon to run agains primary databases with minimal impact during the init_replica process.

The python-mysql-replication requirement is now changed to version >=0.22. This release adds support for PyMySQL >=0.10.0. The requirement for PyMySQL to version <0.10.0 is therefore removed from setup.py.

From this version pg_chameleon refuse to run as root.

3.1.6 2.0.14

This maintenance release improves the support for spatial datatypes. When postgis is installed on the target database then the spatial data types `point`,‘‘geometry‘‘,‘‘linestring‘‘,‘‘polygon‘‘, `multipoint`, `multilinestring`, `geometrycollection` are converted to geometry and the data is replicated using the Well-Known Binary (WKB) Format. As the MySQL implementation for WKB is not standard pg_chameleon removes the first 4 bytes from the decoded binary data before sending it to PostgreSQL.

When `keep_existing_schema` is set to `yes` now drops and recreates indices, and primary keys during the `init_replica` process. The foreign keys are dropped as well and recreated when the replica reaches the consistent status. This way the `init_replica` may complete successfully even when there are foreign keys in place and with the same speed of the usual `init_replica`.

The setup.py now forces PyMySQL to version <0.10.0 because it breaks the python-mysql-replication library (issue #117).

Thanks to @porshkevich which fixed issue #115 by trim the space from PK index name.

This release requires a replica catalogue upgrade, therefore is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to use screen or tmux for the upgrade
- Stop all the replica processes with `chameleon stop_all_replicas --config <your_config>`
- Take a backup of the schema `sch_chameleon` with pg_dump as a good measure.
- Install the upgrade with `pip install pg_chameleon --upgrade`
- Check if the version is upgraded with `chameleon --version`
- Upgrade the replica schema with the command `chameleon upgrade_replica_schema --config <your_config>`
- Start all the replicas.

If the upgrade procedure can’t upgrade the replica catalogue because of running or errored replicas is it possible to reset the statuses by using the command `chameleon enable_replica --source <source_name>`.

If the catalogue upgrade is still not possible then you can downgrade pgchameleon to the previous version. Please note that you may need to install manually PyMySQL to fix the issue with the version 0.10.0.

```
pip install pg_chameleon==2.0.13
pip install "PyMySQL<0.10.0"
```

3.1.7 2.0.13

This maintenance release adds the **EXPERIMENTAL** support for Point datatype thanks to the contribution by @jovankricka-everon.

The support is currently limited to only the POINT datatype with hardcoded stuff to keep the init_replica and the replica working. However as this feature is related with PostGIS, the next point release will rewrite this part of code using a more general approach.

The release adds the `keep_existing_schema` parameter in the MySQL source type. When set to `Yes` init_replica,refresh_schema and sync_tables do not recreate the affected tables using the data from the MySQL source. Instead the existing tables are truncated and the data is reloaded.

A REINDEX TABLE is executed in order to have the indices in good shape after the reload. The next point release will very likely improve the approach on the reload and reindexing.

When `keep_existing_schema` is set to Yes the parameter `grant_select_to` have no effect.

From this release the codebase switched from tabs to spaces, following the guidelines in PEP-8.

3.1.8 2.0.12

This maintenance release fixes the issue #96 where the replica initialisation failed on MySQL 8 because of the wrong field names pulled out from the information_schema. Thanks to @daniel-qcode for contributing with his fix.

The configuration and SQL files are now moved inside into the directory pg_chameleon. This change simplifies the setup.py file and allow pg_chameleon to be built as source and wheel package.

As python 3.4 has now reached its end-of-life and has been retired the minimum requirement for pg_chameleon has been updated to Python 3.5.

3.1.9 2.0.11

This maintenance release fixes few things. As reported in #95 the yaml filles were not completely valid. @rebtoor fixed them.

@clifff made a pull request to have the start_replica running in foreground when log_file set to *stdout*. Previously the process remained in background with the log set to *stdout*.

As Travis seems to break down constantly the CI configuration is disabled until a fix or a different CI is found .

Finally the method which loads the yaml file is now using an explicit loader as required by the new PyYAML version.

Previously with newer version of PyYAML there was a warning emitted by the library because the default loader is unsafe. If you have

3.1.10 2.0.10

This maintenance release fixes a regression caused by the new replay function with PostgreSQL 10. The unnested primary key was put in cartesian product with the json elements generating NULL identifiers which made the subsequent format function to fail.

This release fixes adds a workaround for decoding the keys in the mysql's json fields. This allows the sytem to replicate the json data type as well.

The command `enable_replica` fixes a race condition when the maintenance flag is not returned to false (e.g. an application crash during the maintenance run) allowing the replica to start again.

The tokeniser for the `CHANGE` statement now parses the tables in the form of `schema.table`. However the tokenised schema is not used to determine the query's schema because the `__read_replica_stream` method uses the schema name pulled out from the mysql's binlog.

As this change requires a replica catalogue upgrade is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to use screen or tmux for the upgrade
- Stop all the replica processes with `chameleon stop_all_replicas --config <your_config>`
- Take a backup of the schema `sch_chameleon` with pg_dump for good measure.
- Install the upgrade with `pip install pg_chameleon --upgrade`
- Check if the version is upgraded with `chameleon --version`
- Upgrade the replica schema with the command `chameleon upgrade_replica_schema --config <your_config>`
- Start all the replicas.

If the upgrade procedure refuses to upgrade the catalogue because of running or errored replicas is possible to reset the statuses using the command `chameleon enable_replica --source <source_name>`.

If the catalogue upgrade is still not possible downgrading pgchameleon to the previous version. E.g. `pip install pg_chameleon==2.0.9` will make the replica startable again.

3.1.11 2.0.9

This maintenance release fixes a wrong check for the next auto maintenance run if the maintenance wasn't run before. Previously when changing the value of `auto_maintenance` from disabled to an interval, the process didn't run the automatic maintenance unless a manual maintenance was executed before.

This release adds improvements on the replay function's speed. The new version is now replaying the data without accessing the parent log partition and the decoding logic has been simplified. Not autoritative tests has shown a cpu gain of at least 10% and a better memory allocation. However your mileage may vary.

The GTID operational mode has been improved removing the blocking mode which caused increased lag in systems with larger binlog size.

As this change requires a replica catalogue upgrade is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to use screen or tmux for the upgrade
- Stop all the replica processes with `chameleon stop_all_replicas --config <your_config>`
- Take a backup of the schema `sch_chameleon` with pg_dump for good measure.
- Install the upgrade with `pip install pg_chameleon --upgrade`
- Check if the version is upgraded with `chameleon --version`
- Upgrade the replica schema with the command `chameleon upgrade_replica_schema --config <your_config>`
- Start all the replicas.

If the upgrade procedure refuses to upgrade the catalogue because of running or errored replicas is possible to reset the statuses using the command `chameleon enable_replica --source <source_name>`.

If the catalogue upgrade is still not possible downgrading pgchameleon to the previous version. E.g. `pip install pg_chameleon==2.0.8` will make the replica startable again.

3.1.12 2.0.8

This maintenance release adds the support for skip events. Is now is possible to skip events (insert,delete,update) for single tables or for entire schemas.

A new optional source parameter `skip_events:` is available for the sources with type mysql. Under skip events there are three keys one per each DML operation. Is possible to list an entire schema or single tables in the form of `schema.table`. The example snippet disables the inserts on the table `delphis_mediterranea.foo` and the deletes on the entire schema `delphis_mediterranea`.

```
skip_events:
  insert:
    - delphis_mediterranea.foo #skips inserts on the table delphis_mediterranea.foo
  delete:
    - delphis_mediterranea #skips deletes on schema delphis_mediterranea
  update:
```

The release 2.0.8 adds the **EXPERIMENTAL** support for the GTID for MySQL or Percona server. The GTID in MariaDb is currently not supported. A new optional parameter `gtid_enable:` which defaults to `No` is available for the source type mysql.

When MySQL is configured with the GTID and the parameter `gtid_enable:` is set to Yes, pg_chameleon will use the GTID to auto position the replica stream. This allows pg_chameleon to reconfigure the source within the MySQL replicas without the need to run init_replica.

This feature has been extensively tested but as it's new has to be considered **EXPERIMENTAL**.

ALTER TABLE RENAME is now correctly parsed and executed. ALTER TABLE MODIFY is now parsed correctly when the field have a default value. Previously modify with default values would parse wrongly and fail when translating to PostgreSQL dialect

The source no longer gets an error state when running with `--debug`.

The logged events are now cleaned when refreshing schema and syncing tables. Previously spurious logged events could lead to primary key violations when syncing single tables or refreshing single schemas.

As this change requires a replica catalogue upgrade is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to use screen or tmux for the upgrade
- Stop all the replica processes with `chameleon stop_all_replicas --config <your_config>`
- Take a backup of the schema `sch_chameleon` with pg_dump for good measure.
- Install the upgrade with `pip install pg_chameleon --upgrade`
- Check if the version is upgraded with `chameleon --version`
- Upgrade the replica schema with the command `chameleon upgrade_replica_schema --config <your_config>`
- Start all the replicas.

If the upgrade procedure refuses to upgrade the catalogue because of running or errored replicas is possible to reset the statuses using the command `chameleon enable_replica --source <source_name>`.

If the catalogue upgrade is still not possible downgrading pgchameleon to the previous version. E.g. `pip install pg_chameleon==2.0.7` will make the replica startable again.

3.1.13 2.0.7

This maintenance release makes the multiprocess logging safe. Now each replica process logs in a separate file.

The `--full` option now is working. Previously the option had no effect causing the maintenance to run always a conventional vacuum.

This release fixes the issues reported in ticket #73 and #75 by pg_chameleon's users.

The bug reported in ticket #73 caused a wrong data type tokenisation when an alter table adds a column with options (e.g. `ADD COLUMN foo DEFAULT NULL`)

The bug reported in ticket #75 , caused a wrong conversion to string for the row keys with None value during the cleanup of malformed rows for the init replica and the replica process.

A fix for the TRUNCATE TABLE tokenisation is implemented as well. Now if the statement specifies the table with the schema the truncate works properly.

A new optional source's parameter is added. `auto_maintenance` trigger a vacuum on the log tables after a specific timeout. The timeout shall be expressed like a PostgreSQL interval (e.g. "1 day"). The special value "disabled" disables the auto maintenance. If the parameter is omitted the auto maintenance is disabled.

3.1.14 2.0.6

The maintenance release 2.0.6 fixes a crash occurring when a new column is added on the source database with the default value `NOW()`.

The maintenance introduced in the version 2.0.5 is now less aggressive. In particular the `run_maintenance` command now executes a conventional `VACUUM` on the source's log tables, unless the switch `--full` is specified. In that case a `VACUUM FULL` is executed. The detach has been disabled and may be completely removed in the future releases because very fragile and prone to errors.

However running VACUUM FULL on the log tables can cause the other sources to be blocked during the maintenance run.

This release adds an optional parameter `on_error_read:` on the mysql type's sources which allow the read process to stay up if the mysql database is refusing connections (e.g. MySQL down for maintenance). Following the principle of least astonishment the parameter if omitted doesn't cause any change of behaviour. If added with the value continue (e.g. `on_error_read: continue`) will prevent the replica process to stop in the case of connection issues from the MySQL database with a warning is emitted on the replica log .

This release adds the support for mysql 5.5 which doesn't have the parameter `binlog_row_image`.

`enable_replica` now can reset the replica status to `stopped` even if the catalogue version is mismatched. This simplifies the upgrade procedure in case of errored or wrongly running replicas.

As this change requires a replica catalogue upgrade is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to open a screen session
- Before upgrading pg_chameleon **stop all the replica processes.**
- Upgrade the pg_chameleon package with *pip install pg_chameleon –upgrade*

- Upgrade the replica schema with the command *chameleon upgrade_replica_schema –config <your_config>*
- Start the replica processes

If the upgrade procedure refuses to upgrade the catalogue because of running or errored replicas is possible to reset the statuses with the `enable_replica` command.

If the catalogue upgrade is still not possible downgrading pgchameleon to the version 2.0.5 with `pip install pg_chameleon==2.0.5` should make the replicas startable again.

3.1.15 2.0.5

The maintenance release 2.0.5 fixes a regression which prevented some tables to be synced with *sync_tables* when the parameter *limit_tables* was set. Previously having two or more schemas mapped with only one schema listed in *limit_tables* prevented the other schema's tables to be synchronised with *sync_tables*.

This release add two new commands to improve the general performance and the management.

The command *stop_all_replicas* stops all the running sources within the target postgresql database.

The command *run_maintenance* performs a VACUUM FULL on the specified source's log tables. In order to limit the impact on other sources eventually configured the command performs the following steps.

- The read and replay processes for the given source are paused
- The log tables are detached from the parent table *sch_chameleon.t_log_replica* with the command *NO INHERIT*
- The log tables are vacuumed with *VACUUM FULL*
- The log tables are attached to the parent table *sch_chameleon.t_log_replica* with the command *INHERIT*
- The read and replay processes are resumed

Currently the process is manual but it will become eventually automated if it's proven to be sufficiently robust.

The pause for the replica processes creates the infrastructure necessary to have a self healing replica. This functionality will appear in future releases of the branch 2.0.

As this change requires a replica catalogue upgrade is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to open a screen session
- Before the upgrade stop all the replica processes.
- Upgrade pg_chameleon with *pip install pg_chameleon –upgrade*
- Run the upgrade command *chameleon upgrade_replica_schema –config <your_config>*
- Start the replica processes

3.1.16 2.0.4

The maintenance release 2.0.4 fix the wrong handling of the `ALTER TABLE` when generating the `MODIFY` translation. The regression was added in the version 2.0.3 and can result in a broken replica process.

This version improves the way to handle the replica from tables with dropped columns in the future. The python-mysql-replication library with this commit adds a way to manage the replica with the tables having columns dropped before the read replica is started.

Previously the auto generated column name caused the replica process to crash as the type map dictionary didn't had the corresponding key.

The version 2.0.4 handles the `KeyError` exception and allow the row to be stored on the PostgreSQL target database. However this will very likely cause the table to be removed from the replica in the replay step. A debug log message is emitted when this happens in order to when the issue occurs.

3.1.17 2.0.3

The bugfix release 2.0.3 fixes the issue #63 changeing all the fields *i_binlog_position* to bigint. Previously binlog files larger than 2GB would cause an integer overflow during the phase of write rows in the PostgreSQL database. The issue can affect also MySQL databases with smaller *max_binlog_size* as it seems that this value is a soft limit.

As this change requires a replica catalogue upgrade is very important to follow the upgrade instructions provided below.

- If working via ssh is suggested to open a screen session
- Before the upgrade stop all the replica processes.
- Upgrade pg_chameleon with *pip install pg_chameleon –upgrade*
- Run the upgrade command *chameleon upgrade_replica_schema –config <your_config>*
- Start the replica processes

Please note that because the upgrade command will alter the data types with subsequent table rewrite. The process can take long time, in particular if the log tables are large. If working over a remote machine the best way to proceed is to run the command in a screen session.

This release fixes a regression introduced with the release 2.0.1. When an alter table comes in the form of *ALTER TABLE ADD COLUMN is in the form datatype DEFAULT (NOT) NULL* the parser captures two words instead of one, causing the replica process crash.

The speed of the initial cleanup, when the replica starts has been improved as now the delete runs only on the sources log tables instead of the parent table. This improvement is more effective when many sources are configured all togheter.

From this version the setup.py switches the psycopg2 requirement to using the psycopg2-binary which ensures that psycopg2 will install using the wheel package when available.

3.1.18 2.0.2

This bugfix relase adds a missing functionality which wasn't added during the application development and fixes a bug in the `sync_tables` command.

Previously the parameter `batch_retention` was ignored making the replayed batches to accumulate in the table `sch_chameleon.t_replica_batch` with the conseguent performance degradation over time.

This release solves the issue re enabling the batch_retention. Please note that after upgrading there will be an initial replay lag building. This is normal as the first cleanup will have to remove a lot of rows. After the cleanup is complete the replay will resume as usual.

The new private method `_swap_enums` added to the class `pg_engine` moves the enumerated types from the loading schema to the destination schema when the method `swap_tables` is executed by the command `sync_tables`.

Previously when running `sync_tables` tables with enum fields were created on PostgreSQL without the corresponding enumerated types. This happened because the custom enumerated type were not moved into the destination schema and therefore dropped along with the loading schema when the procedure performed the final cleanup.

3.1.19 2.0.1

The first maintenance release of pg_chameleon v2 adds a performance improvement in the read replica process when the variables limit_tables or skip_tables are set.

Previously all the rows were read from the replica stream as the `BinLogStreamReader` do not allow the usage of the tables in the form of `schema_name.table_name`. This caused a large amount of useless data hitting the replica log tables as reported in the issue #58.

The private method __`store_binlog_event` now evaluates the row schema and table and returns a boolean value on whether the row or query should be stored or not into the log table.

The release fixes also a crash in read replica if an alter table added a column was of type `character varying`.

3.1.20 2.0.0

This stable release consists of the same code of the RC1 with few usability improvements.

A new option is now available to set to set the maximum level for the messages to be sent to rollbar. This is quite useful if we configure a periodical init_replica (e.g. pgsql source type refreshed every hour) and we don't want to fill rollbar with noise. For example `chameleon init_replica --source pgsql --rollbar-level critical` will send to rollbar only messages marked as critical.

There is now a command line alias `chameleon` which is a wrapper for `chameleon.py`.

A new command `enable_replica` is now available to enable the source's replica if the source is not stopped clean.

3.1.21 2.0.0.rc1

This release candidate comes with few bug fixes and few usability improvements.

Previously when adding a table with a replicated DDL having an unique key, the table's creation failed because of the fields were set as NULLable . Now the command works properly.

The system now checks if the MySQL configuration allows the replica when initialising or refreshing replicated entities.

A new class `rollbar_notifier` was added in order to simplyfi the message management within the source and engine classes.

Now the commands `init_replica,refresh_schema,sync_tables` send an info notification to rollbar when they complete successfully or an error if they don't.

The command `sync_tables` now allows the special name `--tables disabled` to have all the tables with replica disabled re synchronised at once.

3.1.22 2.0.0.beta1

The first beta for the milestone 2.0 adds fixes a long standing bug to the replica process and adds more features to the postgresql support.

The race condition fixed was caused by a not tokenised DDL preceeded by row images, causing the collected binlog rows to be added several times to the log_table. It was quite hard to debug as the only visible effect was a primary key violation on random tables.

The issue is caused if a set of rows lesser than the `replica_batch_size` are followed by a DDL that is not tokenised (e.g. ``CREATE TEMPORARY TABLE `foo`;``) which coincides with the end of read from the binary log. In that case the batch is not closed and the next read replica attempt will restart from the previous position reading and storing again the same set of rows. When the batch is closed the replay function will eventually fail because of a primary/unique key violation.

The tokeniser now works properly when an `ALTER TABLE ADD COLUMN`'s definition is surrounded by parentheses e.g. `ALTER TABLE foo ADD COLUMN(bar varchar(30));` There are now error handlers when wrong table names, wrong schema names, wrong source name and wrong commands are specified to `chameleon.py` When running commands that require a source name tye system checks if the source is registered.

The `init_replica` for source pgsql now can read from an hot standby but the copy is not consistent as it's not possible to export a snapshot from the hot standbys. Also the `* init_replica` for source pgsql adds the copied tables as fake "replicated tables" for better show_status display.

For the source type `pgsql` the following restrictions apply.

- There is no support for real time replica
- The data copy happens always with file method
- The copy_max_memory doesn't apply
- The type override doesn't apply
- Only `init_replica` is currently supported
- The source connection string requires a database name

3.1.23 2.0.0.alpha3

please note this is a not production release. do not use it in production

The third and final alpha3 for the milestone 2.0 fixes some issues and add more features to the system.

As there are changes in the replica catalog if upgrading from the alpha1 there will be need to do a `drop_replica_schema` followed by a `create_replica_schema`. This **will drop any existing replica** and will require re adding the sources and re initialise them with `init_replica`.

The system now supports a source type `pgsql` with the following limitations.

- There is no support for real time replica
- The data copy happens always with file method
- The copy_max_memory doesn't apply
- The type override doesn't apply
- Only `init_replica` is currently supported
- The source connection string requires a database name
- In the `show_status` detailed command the replicated tables counters are always zero

A stack trace capture is now added on the log and the rollbar message for better debugging. A new parameter `on_error_replay` is available for the sources to set whether the replay process should skip the tables or exit on error.

This release adds the command `upgrade_replica_schema` for upgrading the replica schema from the version 1.8 to the 2.0.

The upgrade procedure is described in the documentation.

Please read it carefully before any upgrade and backup the schema sch_chameleon before attempting any upgrade.

3.1.24 2.0.0.alpha2

please note this is a not production release. do not use it in production

The second alpha of the milestone 2.0 comes after a week of full debugging. This release is more usable and stable than the alpha1. As there are changes in the replica catalog if upgrading from the alpha1 there will be need to do a `drop_replica_schema` followed by a `create_replica_schema`. This **will drop any existing replica** and will require re adding the sources and re initialise them with `init_replica`.

The full list of changes is in the CHANGELOG file. However there are few notable remarks.

There is a detailed display of the `show_status` command when a source is specified. In particular the number of replicated and not replicated tables is displayed. Also if any table as been pulled out from the replica it appears on the bottom.

From this release there is an error log which saves the exception's data during the replay phase. The error log can be queried with the new command `show_errors`.

A new source parameter `replay_max_rows` has been added to set the amount of rows to replay. Previously the value was set by the parameter `replica_batch_size`. If upgrading from alpha1 you may need to add this parameter to your existing configuration.

Finally there is a new class called `pgsql_source`, not yet functional though. This class will add a very basic support for the postgres source type. More details will come in the alpha3.

3.1.25 2.0.0.alpha1

please note this is a not production release. do not use it in production

This is the first alpha of the milestone 2.0. The project has been restructured in many ways thanks to the user's feedback. Hopefully this will make the system much simple to use.

The main changes in the version 2 are the following.

The system is Python 3 only compatible. Python 3 is the future and there is no reason why to keep developing thing in 2.7.

The system now can read from multiple MySQL schemas in the same database and replicate them it into a target PostgreSQL database. The source and target schema names can be different.

The system now use a conservative approach to the replica. The tables which generate errors during the replay are automatically excluded from the replica.

The init_replica process runs in background unless the logging is on the standard output or the debug option is passed to the command line.

The replica process now runs in background with two separated subprocess, one for the read and one for the replay. If the logging is on the standard output or the debug option is passed to the command line the main process stays in foreground though.

The system now use a soft approach when initialising the replica . The tables are locked only when copied. Their log coordinates will be used by the replica damon to put the database in a consistent status gradually.

The system can now use the rollbark key and environment to setup the Rollbar integration, for a better error detection.

CHAPTER 4

Upgrade procedure

4.1 Maintenance release upgrade

Upgrading a maintenance release is in general very simple but requires some attention.

Always check the release notes. If there is no schema upgrade the procedure is straightforward

- Stop all the replica processes with `chameleon stop_all_replicas --config <your_config>`
- Install the upgrade with `pip install pg_chameleon --upgrade`
- Check if the version is upgraded with `chameleon --version`
- Start all the replicas.

  If the release comes with a schema upgrade, **after stopping the replicas** take a backup of the schema `sch_chameleon` with pg_dump for good measure.

```
pg_dump -h <your_database_server > -n sch_chameleon -Fc -f sch_chameleon.dmp
-U <your_username >-d <your_database>
```

- If working via ssh is suggested to use screen or tmux for the upgrade
- Upgrade the pg_chameleon package with `pip install pg_chameleon --upgrade`
- Upgrade the replica schema with the command `chameleon upgrade_replica_schema --config <your_config>`
- Start the replica processes

If the upgrade procedure refuses to upgrade the catalogue because of running or errored replicas is possible to reset the statuses using the command `chameleon enable_replica --source <source_name>`.

If the catalogue upgrade is still not possible downgrading pgchameleon to the previous version. E.g. `pip install pg_chameleon==2.0.7`.

4.2 Version 1.8 to 2.0 upgrade

pg_chameleon 2.0 can upgrade an existing 1.8 replica catalogue using the command `upgrade_replica_schema`. As the new version supports different schema mappings within the same source the parameter `schema_mappings` must match all the pairs `my_database destination_schema` for the source database that we are configuring. Any discrepancy will abort the upgrade procedure.

4.2.1 Preparation

- **Check the pg_chameleon version you are upgrading is 1.8.2. If not upgrade it and start the replica for each source present**
  This step is required in order to have the destination and target schema's updated from the configuration files.
- Check the replica catalogue version is 1.7 with `SELECT * FROM sch_chameleon.v_version;`.
- Check the field t_source_schema have a schema name set `SELECT t_source_schema FROM sch_chameleon.t_sources;`
- Take a backup of the existing schema sch_chameleon with `pg_dump`
- Install pg_chameleon version 2 and create the configuration files executing `chameleon set_configuration_files`.
- cd in `~/.pg_chameleon/configuration/` and copy the file `config-example.yml` in a different file `e.g. cp config-example.yml upgraded.yml`
- Edit the file and set the target and source's database connections. You may want to change the source name as well

For each configuration file in the old setup `~/.pg_chameleon/config/` using the MySQL database configured in the source you should get the values stored in `my_database` and `destination_schema` and add it to the new source's schema_mappings.

For example, if there are two sources `source_01.yaml` and `source_02.yaml` with the following configuration:

Both sources are pointing the same MySQL database

```
mysql_conn:
    host: my_host.foo.bar
    port: 3306
    user: my_replica
passwd: foo_bar
```

source_01.yaml have the following schema setup

```
my_database: my_schema_01
destination_schema: db_schema_01
```

source_02.yaml have the following schema setup

```
my_database: my_schema_02
destination_schema: db_schema_02
```

The new source's database configuration should be

```
mysql:
    db_conn:
        host: "my_host.foo.bar"
```

(continues on next page)

(continued from previous page)

```
        port: "3306"
        user: "my_replica"
        password: "foo_bar"
        charset: 'utf8'
        connect_timeout: 10
    schema_mappings:
            my_schema_01: db_schema_01
            my_schema_02: db_schema_02
```

4.2.2 Upgrade

Execute the following command `chameleon upgrade_replica_schema --config upgraded`

The procedure checks if the start catalogue version is 1.7 and fails if the value is different. After answering YES the procedure executes the following steps.

- Replays any exising batches present in the catalogue 1.7
- Checks if the schema_mappings are compatible with the values stored in the schema `sch_chameleon`
- Renames the schema `sch_chameleon` to `_sch_chameleon_version1`
- Installs a new 2.0 schema in `sch_chameleon`
- Stores a new source using the schema mappings
- Migrates the existing tables into the new catalogue using the replica batch data to store the tables start of consistent point.
- Determines maximum and minimum point for the binlog coordinates and use them for writing the new batch start point and the source's consistent point

If the migration is successful, before starting the replica process is better to check that all tables are correctly mapped with

`chameleon show_status --source upgraded`

4.2.3 Rollback

If something goes wrong during the upgrade procedure, then the changes are rolled back. The schema `sch_chameleon` is renamed to `_sch_chameleon_version2` and the previous version's schema `_sch_chameleon_version1` is put batck to `sch_chameleon`. If this happens the procedure 1.8.2 will continue to work as usual. The schema `_sch_chameleon_version2` can be used to check what went wrong.

Before attempting a new upgrade schema `_sch_chameleon_version2` should be dropped or renamed in order to avoid a schema conflict in the case of another failure.

CHAPTER 5

README

pg_chameleon is a MySQL to PostgreSQL replica system written in Python 3. The system use the library mysql-replication to pull the row images from MySQL which are stored into PostgreSQL as JSONB. A pl/pgsql function decodes the jsonb values and replays the changes against the PostgreSQL database.

pg_chameleon 2.0 is available on pypi

The documentation is available on pgchameleon.org

Please submit your bug reports on GitHub.

5.1 Requirements

5.1.1 Replica host

Operating system: Linux, FreeBSD, OpenBSD Python: CPython 3.5+

- PyMySQL
- mysql-replication
- psycopg2
- PyYAML
- tabulate

- rollbar
- daemonize

Optionals for building documentation

- sphinx
- sphinx-autobuild

5.1.2 Origin database

MySQL 5.5+

Aurora MySQL 5.7+

5.1.3 Destination database

PostgreSQL 9.5+

5.1.4 Example scenarios

- Analytics
- Migrations
- Data aggregation from multiple MySQL databases

5.1.5 Features

- Read from multiple MySQL schemas and restore them it into a target PostgreSQL database. The source and target schema names can be different.
- Setup PostgreSQL to act as a MySQL replica.
- Support for enumerated and binary data types.
- Basic DDL Support (CREATE/DROP/ALTER TABLE, DROP PRIMARY KEY/TRUNCATE, RENAME).
- Discard of rubbish data coming from the replica.
- Conservative approach to the replica. Tables which generate errors are automatically excluded from the replica.
- Possibilty to refresh single tables or single schemas.
- Basic replica monitoring.
- Detach replica from MySQL for migration support.
- Data type override (e.g. tinyint(1) to boolean)
- Daemonised init_replica process.
- Daemonised replica process with two separated subprocess, one for the read and one for the replay.
- Rollbar integration
- Support for geometrical data. **Requires PostGIS on the target database.**
- Minimal locking during init_replica for transactional engines (e.g. innodb)

5.1.6 Caveats

The replica requires the tables to have a primary or unique key. Tables without primary/unique key are initialised during the init_replica process but not replicated.

The copy_max_memory is just an estimate. The average rows size is extracted from mysql's informations schema and can be outdated. If the copy process fails for memory error check the failing table's row length and the number of rows for each slice.

Python 3 is supported only from version 3.5 as required by mysql-replication .

The lag is determined using the last received event timestamp and the postgresql timestamp. If the mysql is read only the lag will increase because no replica event is coming in.

The detach replica process resets the sequences in postgres to let the database work standalone. The foreign keys from the source MySQL schema are extracted and created initially as NOT VALID. The foreign keys are created without the ON DELETE or ON UPDATE clauses. A second run tries to validate the foreign keys. If an error occurs it gets logged out according to the source configuration.

5.2 Setup

5.2.1 RPM PGDG

pg_chameleon is included in the PGDG RMP repository thanks to Devrim.

Please follow the instructions on https://www.postgresql.org/download/linux/redhat/

5.2.2 openSUSE Build Service

pg_chameleon is available on the openSUSE build Service

Currently all releases are supported except SLE_12_SP5 because of unresolved dependencies.

5.2.3 Virtual env setup

- Create a virtual environment (e.g. python3 -m venv venv)
- Activate the virtual environment (e.g. source venv/bin/activate)
- Upgrade pip with **pip install pip –upgrade**
- Install pg_chameleon with **pip install pg_chameleon**.
- Create a user on mysql for the replica (e.g. usr_replica)
- Grant access to usr on the replicated database (e.g. GRANT ALL ON sakila.* TO 'usr_replica';)
- Grant RELOAD privilege to the user (e.g. GRANT RELOAD ON *.* to 'usr_replica';)
- Grant REPLICATION CLIENT privilege to the user (e.g. GRANT REPLICATION CLIENT ON *.* to 'usr_replica';)
- Grant REPLICATION SLAVE privilege to the user (e.g. GRANT REPLICATION SLAVE ON *.* to 'usr_replica';)

5.3 Configuration directory

The system wide install is now supported correctly.

The configuration is set with the command `chameleon set_configuration_files` in $HOME/.pg_chameleon . Inside the directory there are three subdirectories.

- configuration is where the configuration files are stored.
- pid is where the replica pid file is created. it can be changed in the configuration file
- logs is where the replica logs are saved if log_dest is file. It can be changed in the configuration file

You should use config-example.yaml as template for the other configuration files. Check the configuration file reference for an overview.

CHAPTER 6

The configuration file

6.1 The configuration file

The file config-example.yaml is stored in **~/.pg_chameleon/configuration** and should be used as template for the other configuration files. The configuration consists of three configuration groups.

6.1.1 Global settings

```yaml
# global settings
pid_dir: '~/.pg_chameleon/pid/'
log_dir: '~/.pg_chameleon/logs/'
log_dest: file
log_level: info
log_days_keep: 10
rollbar_key: ''
rollbar_env: ''
```

- pid_dir directory where the process pids are saved.
- log_dir directory where the logs are stored.
- log_dest log destination. stdout for debugging purposes, file for the normal activity.
- log_level logging verbosity. allowed values are debug, info, warning, error.
- log_days_keep configure the retention in days for the daily rotate replica logs.
- rollbar_key: the optional rollbar key
- rollbar_env: the optional rollbar environment

If both rollbar_key and rollbar_env are configured some messages are sent to the rollbar conf

6.1.2 type override

```
# type_override allows the user to override the default type conversion
# into a different one.

type_override:
  "tinyint(1)":
    override_to: boolean
```

The type_override allows the user to override the default type conversion into a different one. Each type key should be named exactly like the mysql type to override including the dimensions. Each type key needs two subkeys.

- override_to specifies the destination type which must be a postgresql type and the type cast should be possible
- override_tables is a yaml list which specifies to which tables the override applies. If the first list item is set to "*" then the override is applied to all tables in the replicated schemas.

The override is applied when running the init_replica,refresh_schema andsync_tables process. The override is also applied for each matching DDL (create table/alter table) if the table name matches the override_tables values.

6.1.3 PostgreSQL target connection

```
# postgres  destination connection
pg_conn:
  host: "localhost"
  port: "5432"
  user: "usr_replica"
  password: "never_commit_password"
  database: "db_replica"
  charset: "utf8"
```

The pg_conn key maps the target database connection string.

6.1.4 sources configuration

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
```

(continues on next page)

(continued from previous page)

```
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No

  pgsql:
    db_conn:
      host: "localhost"
      port: "5432"
      user: "usr_replica"
      password: "never_commit_passwords"
      database: "db_replica"
      charset: 'utf8'
      connect_timeout: 10
      schema_mappings:
        loxodonta_africana: elephas_maximus
      limit_tables:
        - loxodonta_africana.foo
      skip_tables:
        - loxodonta_africana.bar
      copy_max_memory: "300M"
      grant_select_to:
        - usr_readonly
      lock_timeout: "10s"
      my_server_id: 100
      replica_batch_size: 3000
      replay_max_rows: 10000
      sleep_loop: 5
      batch_retention: '1 day'
      copy_mode: 'file'
      out_dir: /tmp
      type: pgsql
```

The key sources allow to setup multiple replica sources writing on the same postgresql database. The key name myst be unique within the replica configuration.

The following remarks apply only to the mysql source type.

For the postgresql source type. See the last section for the description and the limitations.

Database connection

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No
```

The db_conn key maps the target database connection string. Within the connection is possible to configure the connect_timeout which is 10 seconds by default. Larger values could help the tool working better on slow networks. Low values can cause the connection to fail before any action is performed.

Schema mappings

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
```

(continues on next page)

(continued from previous page)

```
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No
```

The key schema mappings is a dictionary. Each key is a MySQL database that needs to be replicated in PostgreSQL. Each value is the destination schema in the PostgreSQL database. In the example provided the MySQL database `delphis_mediterranea` is replicated into the schema `loxodonta_africana` stored in the database specified in the pg_conn key (db_replica).

Limit and skip tables

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
```

(continues on next page)

(continued from previous page)

```yaml
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No
```

- limit_tables list with the tables to replicate. If the list is empty then the entire mysql database is replicated.
- skip_tables list with the tables to exclude from the replica.

The table's names should be in the form SCHEMA_NAME.TABLE_NAME.

Grant select to option

```yaml
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
```

(continues on next page)

(continued from previous page)

```
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No
```

This key allows to specify a list of database roles which will get select access on the replicate tables.

Source configuration parameters

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No
```

- lock_timeout the max time in seconds that the target postgresql connections should wait for acquiring a lock.

This parameter applies to init_replica,refresh_schema and sync_tables when performing the relation's swap.

- my_server_id the server id for the mysql replica. must be unique within the replica cluster
- replica_batch_size the max number of rows that are pulled from the mysql replica before a write on the postgresql database is performed. See caveats in README for a complete explanation.
- batch_retention the max retention for the replayed batches rows in t_replica_batch. The field accepts any valid interval accepted by PostgreSQL
- copy_max_memory the max amount of memory to use when copying the table in PostgreSQL. Is possible to specify the value in (k)ilobytes, (M)egabytes, (G)igabytes adding the suffix (e.g. 300M).
- copy_mode the allowed values are 'file' and 'direct'. With direct the copy happens on the fly. With file the table is first dumped in a csv file then reloaded in PostgreSQL.
- out_dir the directory where the csv files are dumped during the init_replica process if the copy mode is file.
- sleep_loop seconds between a two replica batches.
- on_error_replay specifies whether the replay process should `exit` or `continue` if any error during the replay happens. If `continue` is specified the offending tables are removed from the replica.
- on_error_read specifies whether the read process should `exit` or `continue` if a connection error during the read process happens. If `continue` is specified the process emits a warning and waits for the connection to come back. If the parameter is omitted the default is `exit` which cause the replica process to stop with error.
- auto_maintenance specifies the timeout after an automatic maintenance is triggered. The parameter accepts values valid for the PostgreSQL interval data type (e.g. `1 day`). If the value is set to `disabled` the automatic maintenance doesn't run. If the parameter is omitted the default is `disabled`.
- gtid_enable **(EXPERIMENTAL)** Specifies whether to use the gtid to auto position the replica stream. This parameter have effect only on MySQL and only if the server is configured with the GTID.
- type specifies the source database type. The system supports `mysql` or `pgsql`. See below for the pgsql limitations.

Skip events configuration

```yaml
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
```

(continues on next page)

(continued from previous page)

```
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No
```

The `skip_events` variable allows to tell pg_chameleon to skip events for tables or entire schemas. The example provided with configuration-example.ym disables the inserts on the table `delphis_mediterranea.foo` and disables the deletes on the entire schema `delphis_mediterranea`.

Keep existing schema

```
sources:
  mysql:
    db_conn:
      host: "localhost"
      port: "3306"
      user: "usr_replica"
      password: "never_commit_passwords"
      charset: 'utf8'
      connect_timeout: 10
    schema_mappings:
      delphis_mediterranea: loxodonta_africana
    limit_tables:
      - delphis_mediterranea.foo
    skip_tables:
      - delphis_mediterranea.bar
    grant_select_to:
      - usr_readonly
    lock_timeout: "120s"
    my_server_id: 100
    replica_batch_size: 10000
    replay_max_rows: 10000
    batch_retention: '1 day'
    copy_max_memory: "300M"
    copy_mode: 'file'
    out_dir: /tmp
    sleep_loop: 1
    on_error_replay: continue
    on_error_read: continue
    auto_maintenance: "disabled"
    gtid_enable: false
    type: mysql
```

(continues on next page)

(continued from previous page)

```
    skip_events:
      insert:
        - delphis_mediterranea.foo  # skips inserts on delphis_mediterranea.foo
      delete:
        - delphis_mediterranea  # skips deletes on schema delphis_mediterranea
      update:
    keep_existing_schema: No
```

When set to `Yes` init_replica,refresh_schema and sync_tables do not recreate the affected tables using the data from the MySQL source.

Instead the existing tables are truncated and the data is reloaded. A REINDEX TABLE is executed in order to have the indices in good shape after the reload.

When `keep_existing_schema` is set to Yes the parameter `grant_select_to` have no effect.

PostgreSQL source type (EXPERIMENTAL)

pg_chameleon 2.0 has an experimental support for the postgresql source type. When set to `pgsql` the system expects a postgresql source database rather a mysql. The following limitations apply.

- There is no support for real time replica
- The data copy happens always with file method
- The copy_max_memory doesn't apply
- The type override doesn't apply
- Only `init_replica` is currently supported
- The source connection string requires a database name
- In the `show_status` detailed command the replicated tables counters are always zero

```
  pgsql:
    db_conn:
      host: "localhost"
      port: "5432"
      user: "usr_replica"
      password: "never_commit_passwords"
      database: "db_replica"
      charset: 'utf8'
      connect_timeout: 10
      schema_mappings:
        loxodonta_africana: elephas_maximus
      limit_tables:
        - loxodonta_africana.foo
      skip_tables:
        - loxodonta_africana.bar
      copy_max_memory: "300M"
      grant_select_to:
        - usr_readonly
      lock_timeout: "10s"
      my_server_id: 100
      replica_batch_size: 3000
      replay_max_rows: 10000
      sleep_loop: 5
```

(continues on next page)

(continued from previous page)

```
      batch_retention: '1 day'
      copy_mode: 'file'
      out_dir: /tmp
      type: pgsql
```

Fillfactor

The dictionary fillfactor is used to set the fillfactor for tables that are expected to work with large updates. The key name defines the fillfactor level (The allowed values range is 10 to 100). If key name is set to "*" then the fillfactor applies to all tables in the replicated schema. If the table appears multiple times, then only the last matched value will be applied

```
fillfactor:
  "30":
    tables:
      - "foo.bar"
  "40":
    tables:
      - "foo.foobar"
```

CHAPTER 7

Usage instructions

7.1 Usage

7.1.1 Command line reference

```
chameleon command [ [ --config ] [ --source ] [ --schema ]  [ --tables ] [--logid] [ -
↪-debug ] [ --rollbar-level ] ] [ --version ] [ --full ]
```

Table 1: Options

Option	Description	Default	Example
--config	Specifies the configuration to use in ~.pg_chameleon/configuration/. The configuration name should be the file without the extension .yml	default	--config foo will use the file ~.pg_chameleon/configuration/foo.yml
--source	Specifies the source within a configuration file.	N/A	--source bar
--schema	Specifies a schema configured within a source.	N/A	--schema schema_foo
--tables	Specifies one or more tables configured in a schema. Multiple tables can be specified separated by comma. The table must have the schema.	N/A	--tables schema_foo.table_bar
--logid	Specifies the log id entry for displaying the error details	N/A	--logid 30
--debug	When added to the command line the debug option disables any daemonisation and outputs all the logging to the console. The keybord interrupt signal is trapped correctly.	N/A	--debug
--version	Displays the package version.	N/A	--version
--rollbar-level	Sets the maximum level for the messages to be sent to rolllbar. Accepted values: “critical” “error” “warning” “info”	info	--rollbar-level error
--full	Runs a VACUUM FULL on the log tables when the run_maintenance is executed	N/A	--full

Table 2: Command list reference

Command	Description	Options
set_configuration_files	Setup the example configuration files and directories in ~/.pg_chameleon	
show_config	Displays the configuration for the configuration	--config
show_sources	Displays the sourcches configured for the configuration	--config
show_status	Displays an overview of the status of the sources configured within the configuration. Specifying the source gives more details about that source	--config --source
show_errors	Displays the errors logged by the replay function. If a log id is specified then the log entry is displayed entirely	--config --logid
create_replica_schema	Creates a new replication schema into the config's destination database	--config
drop_replica_schema	Drops an existing replication schema from the config's destination database	--config
upgrade_replica_schema	Upgrades the replica schema from a an older version	--config
add_source	Adds a new source to the replica catalogue	--config --source
drop_source	Remove an existing source from the replica catalogue	--config --source
init_replica	Initialise the replica for an existing source	--config --source
copy_schema	Copy only the schema from mysql to PostgreSQL.	--config --source
update_schema_mappings	Update the schema mappings stored in the replica catalogue using the data from the configuration file.	--config --source
refresh_schema	Synchronise all the tables for a given schema within an already initialised source.	--config --source --schema
sync_tables	Synchronise one or more tables within an already initialised source. The switch `--tables` accepts the special name `disabled` to resync all the tables with replica disabled.	--config --source --tables
start_replica	Starts the replica process daemon	--config --source
stop_replica	Stops the replica process daemon	--config --source
detach_replica	Detaches a replica from the mysql master configuring the postgres schemas to work as a standalone system. Useful for migrations.	--config --source
enable_replica	Enables the replica for the given source changing the source status to stopped. It's useful if the replica crashes.	--config --source
run_maintenance	Runs a VACUUM on the log tables for the given source. If is specified then the maintenance runs a VACUUM FULL	--config --source --full
stop_all_replicas	Stops all the running sources within the target postgresql database.	--config

7.1.2 Example

Create a virtualenv and activate it

```
python3 -m venv venv
source venv/bin/activate
```

Install pg_chameleon

```
pip install pip --upgrade
pip install pg_chameleon
```

Run the set_configuration_files command in order to create the configuration directory.

```
chameleon set_configuration_files
```

cd in ~/.pg_chameleon/configuration and copy the file config-example.yml` to ``default.yml.

In MySQL create a user for the replica.

```
CREATE USER usr_replica ;
SET PASSWORD FOR usr_replica=PASSWORD('replica');
GRANT ALL ON sakila.* TO 'usr_replica';
GRANT RELOAD ON *.* to 'usr_replica';
GRANT REPLICATION CLIENT ON *.* to 'usr_replica';
GRANT REPLICATION SLAVE ON *.* to 'usr_replica';
FLUSH PRIVILEGES;
```

Add the configuration for the replica to my.cnf. It requires a MySQL restart.

```
binlog_format= ROW
binlog_row_image=FULL
log-bin = mysql-bin
server-id = 1
expire_logs_days = 10
```

In PostgreSQL create a user for the replica and a database owned by the user

```
CREATE USER usr_replica WITH PASSWORD 'replica';
CREATE DATABASE db_replica WITH OWNER usr_replica;
```

Check you can connect to both databases from the machine where pg_chameleon is installed.

For MySQL

```
mysql -p -h derpy -u usr_replica sakila
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 116
Server version: 5.6.30-log Source distribution

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

For PostgreSQL

```
psql  -h derpy -U usr_replica db_replica
Password for user usr_replica:
psql (9.5.5)
Type "help" for help.
db_replica=>
```

Check the docs for the configuration file reference. It will help you to configure correctly the connections.

Initialise the replica

```
chameleon create_replica_schema --debug
chameleon add_source --config default  --debug
chameleon init_replica --config default --debug
```

Start the replica with

```
chameleon start_replica --config default --source example
```

Check the source status

```
chameleon show_status --source example
```

Check the error log

```
chameleon show_errors
```

```
chameleon start_replica --config default --source example
```

To stop the replica

```
chameleon stop_replica --config default --source example
```

To detach the replica

```
chameleon detach_replica --config default --source example
```

CHAPTER 8

Module reference

8.1 global_lib api documentation

8.2 mysql_lib api documentation

8.3 pg_lib api documentation

class pg_lib.**pg_encoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Bases: `json.encoder.JSONEncoder`

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

encode(*o*)

Return a JSON string representation of a Python data structure.

```
>>> from json.encoder import JSONEncoder
>>> JSONEncoder().encode({"foo": ["bar", "baz"]})
'{"foo": ["bar", "baz"]}'
```

item_separator = ', '

iterencode(*o*, *_one_shot=False*)

Encode the given object and yield each string representation as available.

For example:

```
for chunk in JSONEncoder().iterencode(bigobject):
    mysocket.write(chunk)
```

key_separator = ': '

class pg_lib.**pg_engine**

Bases: object

add_source()

The method adds a new source to the replication catalog. The method calls the function fn_refresh_parts() which generates the log tables used by the replica. If the source is already present a warning is issued and no other action is performed.

build_alter_table(*schema*, *token*)

The method builds the alter table statement from the token data. The function currently supports the following statements. DROP TABLE ADD COLUMN CHANGE MODIFY

The change and modify are potential source of breakage for the replica because of the mysql implicit fallback data types. For better understanding please have a look to

http://www.cybertec.at/why-favor-postgresql-over-mariadb-mysql/

Parameters

- **schema** – The schema where the affected table is stored on postgres.
- **token** – A dictionary with the tokenised sql statement

Returns query the DDL query in the PostgreSQL dialect

Return type string

build_create_index(*schema*, *table*, *index_data*)

The method loops over the list index_data and builds a new list with the statements for the indices.

Parameters

- **destination_schema** – the schema where the table belongs
- **table_name** – the table name
- **index_data** – the index dictionary used to build the create index statements

Returns a list with the alter and create index for the given table

Return type list

build_enum_ddl(*schema*, *enm_dic*)

The method builds the enum DDL using the token data. The postgresql system catalog is queried to determine whether the enum exists and needs to be altered. The alter is not written in the replica log table but executed as single statement as PostgreSQL do not allow the alter being part of a multi command SQL.

Parameters

- **schema** – the schema where the enumeration is present
- **enm_dic** – a dictionary with the enumeration details

Returns a dictionary with the pre_alter and post_alter statements (e.g. pre alter create type , post alter drop type)

Return type dictionary

check_auto_maintenance()

This method checks if the the maintenance for the given source is required. The SQL compares the last maintenance stored in the replica catalogue with the NOW() function. If the value is bigger than the configuration parameter auto_maintenance then it returns true. Otherwise returns false.

Returns flag which tells if the maintenance should run or not

Return type boolean

check_postgis()

The method checks whether postgis is present or not on the

check_replica_schema()

The method checks if the sch_chameleon exists

Returns count from information_schema.schemata

Return type integer

check_schema_mappings(*exclude_current_source=False*)

The default is false.

The method checks if there is already a destination schema in the stored schema mappings. As each schema should be managed by one mapping only, if the method returns None then the source can be store safely. Otherwise the action. The method doesn't take any decision leaving this to the calling methods. The method assumes there is a database connection active. The method returns a list or none. If the list is returned then contains the count and the destination schema name that are already present in the replica catalogue.

Parameters **exclude_current_source** – If set to true the check excludes the current source name from the check.

Returns the schema already mapped in the replica catalogue.

Return type list

check_source()

The method checks if the source name stored in the class variable self.source is already present. As this method is used in both add and drop source it just retuns the count of the sources. Any decision about the source is left to the calling method. The method assumes there is a database connection active.

check_source_consistent()

This method checks if the database is consistent using the source's high watermark and the source's flab b_consistent. If the batch data is larger than the source's high watermark then the source is marked consistent and all the log data stored witth the source's tables are set to null in order to ensure all the tables are replicated.

clean_batch_data()

This method removes all the batch data for the source id stored in the class varible self.i_id_source.

The method assumes there is a database connection active.

clean_not_processed_batches()

The method cleans up the not processed batches rows from the table sch_chameleon.t_log_replica. The method should be executed only before starting a replica process. The method assumes there is a database connection active.

cleanup_idx_cons(*schema*, *table*)
The method cleansup the constraint and indices for the given table using the statements collected in collect_idx_cons. :param schema: the table's schema :param table: the table's name

cleanup_replayed_batches()
The method cleanup the replayed batches for the given source accordingly with the source's parameter batch_retention

cleanup_source_tables()
The method cleans up the tables for active source in sch_chameleon.t_replica_tables.

cleanup_table_events()
The method cleans up the log events in the source's log tables for the given tables

collect_idx_cons(*schema*, *table*)
The method collects indices and primary keys for the given table from the views v_idx_pkeys,v_fkeys. :param schema: the table's schema :param table: the table's name

connect_db()
Connects to PostgreSQL using the parameters stored in self.dest_conn. The dictionary is built using the parameters set via adding the key dbname to the self.pg_conn dictionary. This method's connection and cursors are widely used in the procedure except for the replay process which uses a dedicated connection and cursor.

copy_data(*csv_file*, *schema*, *table*, *column_list*)
The method copy the data into postgresql using psycopg2's copy_expert. The csv_file is a file like object which can be either a csv file or a string io object, accordingly with the configuration parameter copy_mode. The method assumes there is a database connection active.

Parameters

- **csv_file** – file like object with the table's data stored in CSV format
- **schema** – the schema used in the COPY FROM command
- **table** – the table name used in the COPY FROM command
- **column_list** – A string with the list of columns to use in the COPY FROM command already quoted and comma separated

create_database_schema(*schema_name*)
The method creates a database schema. The create schema is issued with the clause IF NOT EXISTS. Should the schema be already present the create is skipped.

Parameters **schema_name** – The schema name to be created.

create_foreign_keys()
The method creates and validates the foreign keys if we are not keeping the existing schema.

create_idx_cons(*schema*, *table*)
The method creates the constraint and indices for the given table using the statements collected in collect_idx_cons. The foreign keys are not created at this stage as they may be left inconsistent during the initial replay phase. The foreign key creation is managed by __create_foreign_keys() which is executed when the replica reaches the consistent status. :param schema: the table's schema :param table: the table's name

create_indices(*schema*, *table*, *index_data*)
The method loops over the list index_data and creates the indices on the table specified with schema and table parameters. The method assumes there is a database connection active.

Parameters

- **schema** – the schema name where table belongs
- **table** – the table name where the data should be inserted
- **index_data** – a list of dictionaries with the index metadata for the given table.

Returns a list with the eventual column(s) used as primary key

Return type list

create_replica_schema()

The method installs the replica schema sch_chameleon if not already present.

create_table(*table_metadata*, *table_name*, *schema*, *metadata_type*)

Executes the create table returned by __build_create_table (mysql or pgsql) on the destination_schema.

Parameters

- **table_metadata** – the column dictionary extracted from the source's information_schema or builty by the sql_parser class
- **table_name** – the table name
- **destination_schema** – the schema where the table belongs
- **metadata_type** – the metadata type, currently supported mysql and pgsql

detach_replica()

The method detach the replica from mysql, resets all the sequences and creates the foreign keys using the dictionary extracted from mysql. The result is a stand alone set of schemas ready to work.

The foreign keys are first created invalid then validated in a second time.

disconnect_db()

The method disconnects the postgres connection if there is any active. Otherwise ignore it.

drop_database_schema(*schema_name*, *cascade*)

The method drops a database schema. The drop can be either schema is issued with the clause IF NOT EXISTS. Should the schema be already present the create is skipped.

Parameters

- **schema_name** – The schema name to be created.
- **schema_name** – If true the schema is dropped with the clause cascade.

drop_replica_schema()

The method removes the service schema discarding all the replica references. The replicated tables are kept in place though.

drop_source()

The method deletes the source from the replication catalogue. The log tables are dropped as well, discarding any replica reference for the source.

end_maintenance()

The method sets the flag b_maintenance to false for the given source

generate_default_statements(*schema*, *table*, *column*, *create_column=None*)

The method gets the default value associated with the table and column removing the cast. :param schema: The schema name :param table: The table name :param column: The column name :return: the statements for dropping and creating default value on the affected table :rtype: dictionary

get_active_sources()
The method counts all the sources with state not in 'ready' or 'stopped'. The method assumes there is a database connection active.

get_batch_data()
The method updates the batch status to started for the given source_id and returns the batch informations.

Returns psycopg2 fetchall results without any manipulation

Return type psycopg2 tuple

get_catalog_version()
The method returns if the replica schema's version

Returns the version string selected from sch_chameleon.v_version

Return type text

get_data_type(*column*, *schema*, *table*)
The method determines whether the specified type has to be overridden or not.

Parameters

- **column** – the column dictionary extracted from the information_schema or built in the sql_parser class
- **schema** – the schema name
- **table** – the table name

Returns the postgresql converted column type

Return type string

get_existing_pkey(*schema*, *table*)
The method gets the primary key of an existing table and returns the field(s) composing the PKEY as a list. :param schema: the schema name where table belongs :param table: the table name where the data should be inserted :return: a list with the eventual column(s) used as primary key :rtype: list

get_inconsistent_tables()
The method collects the tables in not consistent state. The informations are stored in a dictionary which key is the table's name. The dictionary is used in the read replica loop to determine wheter the table's modifications should be ignored because in not consistent state.

Returns a dictionary with the tables in inconsistent state and their snapshot coordinates.

Return type dictionary

get_log_data(*log_id*)
The method gets the error log entries, if any, from the replica schema. :param log_id: the log id for filtering the row by identifier :return: a dictionary with the errors logged :rtype: dictionary

get_replica_paused()
The method returns the status of the replica. This value is used in both read/replay replica methods for updating the corresponding flags. :return: the b_paused flag for the current source :rtype: boolean

get_replica_status()
The method gets the replica status for the given source. The method assumes there is a database connection active.

get_schema_list()
The method gets the list of source schemas for the given source. The list is generated using the mapping in sch_chameleon.t_sources. Any change in the configuration file is ignored The method assumes there is a database connection active.

get_schema_mappings()

The method gets the schema mappings for the given source. The list is the one stored in the table sch_chameleon.t_sources. Any change in the configuration file is ignored The method assumes there is a database connection active. :return: the schema mappings extracted from the replica catalogue :rtype: dictionary

get_status()

The method gets the status for all sources configured in the target database. :return: a list with the status details :rtype: list

get_table_pkey(*schema*, *table*)

The method queries the table sch_chameleon.t_replica_tables and gets the primary key associated with the table, if any. If there is no primary key the method returns None

Parameters

- **schema** – The table schema
- **table** – The table name

Returns the primary key associated with the table

Return type list

get_tables_disabled(*format='csv'*)

The method returns a CSV or a python list of tables excluded from the replica. The origin's schema is determined from the source's schema mappings jsonb.

Returns CSV list of tables excluded from the replica

Return type text

grant_select()

The method grants the select permissions on all the tables on the replicated schemas to the database roles listed in the source's variable grant_select_to. In the case a role doesn't exist the method emits an error message and skips the missing user.

insert_batch(*group_insert*)

Fallback method for the batch insert. Each row event is processed individually and any problematic row is discarded into the table t_discarded_rows. The row is encoded in base64 in order to prevent any encoding or type issue.

Parameters **group_insert** – the event data built in mysql_engine

insert_data(*schema*, *table*, *insert_data*, *column_list*)

The method is a fallback procedure for when the copy method fails. The procedure performs a row by row insert, very slow but capable to skip the rows with problematic data (e.g. encoding issues).

Parameters

- **schema** – the schema name where table belongs
- **table** – the table name where the data should be inserted
- **insert_data** – a list of records extracted from the database using the unbuffered cursor
- **column_list** – the list of column names quoted for the inserts

insert_source_timings()

The method inserts the source timings in the tables t_last_received and t_last_replayed. On conflict sets the replay/receive timestamps to null. The method assumes there is a database connection active.

reindex_table(*schema*, *table*)
The method run a REINDEX TABLE on the table defined by schema and name. :param schema: the table's schema :param table: the table's name

replay_replica()
The method replays the row images in the target database using the function fn_replay_mysql. The function returns a composite type. The first element is a boolean flag which is true if the batch still require replay. it's false if it doesn't. In that case the while loop ends. The second element is a, optional list of table names. If any table cause error during the replay the problem is captured and the table is removed from the replica. Then the name is returned by the function. As the function can find multiple tables with errors during a single replay run, the table names are stored in a list (Actually is a postgres array, see the create_schema.sql file for more details).

> Each batch which is looped trough can also find multiple tables so we return a list of lists to the replica_engine's calling method.

rollback_upgrade_v1()
The procedure rollsback the upgrade dropping the schema sch_chameleon and renaming the version 1 to the

run_maintenance()
The method runs the maintenance for the given source. After the replica daemons are paused the procedure detach the log tables from the parent log table and performs a VACUUM FULL againts the tables. If any error occurs the tables are attached to the parent table and the replica daemons resumed.

save_discarded_row(*row_data*)
The method saves the discarded row in the table t_discarded_row along with the id_batch. The row is encoded in base64 as the t_row_data is a text field.

Parameters **row_data** – the row data dictionary

save_master_status(*master_status*)
This method saves the master data determining which log table should be used in the next batch. The method assumes there is a database connection active.

Parameters **master_status** – the master data with the binlogfile and the log position

Returns the batch id or none if no batch has been created

Return type integer

set_application_name(*action="*)
The method sets the application name in the replica using the variable self.pg_conn.global_conf.source_name, Making simpler to find the replication processes. If the source name is not set then a generic PGCHAMELEON name is used.

set_autocommit_db(*auto_commit*)
The method sets the auto_commit flag for the class connection self.pgsql_conn. In general the connection is always autocommit but in some operations (e.g. update_schema_mappings) is better to run the process in a single transaction in order to avoid inconsistencies.

Parameters **autocommit** – boolean flag which sets autocommit on or off.

set_batch_processed(*id_batch*)
The method updates the flag b_processed and sets the processed timestamp for the given batch id. The event ids are aggregated into the table t_batch_events used by the replay function.

Parameters **id_batch** – the id batch to set as processed

set_consistent_table(*table*, *schema*)
The method set to NULL the binlog name and position for the given table. When the table is marked consistent the read replica loop reads and saves the table's row images.

Parameters **`table`** – the table name

`set_lock_timeout`()

The method sets the lock timeout using the value stored in the class attribute lock_timeout.

`set_read_paused`(*read_paused*)

The method sets the read proces flag b_paused to true for the given source. The update is performed for the given source and for the negation of b_paused. This approach will prevent unnecessary updates on the table t_last_received.

Parameters **`read_paused`** – the flag to set for the read replica process.

`set_replay_paused`(*read_paused*)

The method sets the read proces flag b_paused to true for the given source. The update is performed for the given source and for the negation of b_paused. This approach will prevent unnecessary updates on the table t_last_received.

Parameters **`read_paused`** – the flag to set for the read replica process.

`set_source_highwatermark`(*master_status*, *consistent*)

This method saves the master data within the source. The values are used to determine whether the database has reached the consistent point.

Parameters **`master_status`** – the master data with the binlogfile and the log position

`set_source_id`()

The method sets the class attribute i_id_source for the self.source. The method assumes there is a database connection active.

`set_source_status`(*source_status*)

The method updates the source status for the source_name and sets the class attribute i_id_source. The method assumes there is a database connection active.

Parameters **`source_status`** – The source status to be set.

`store_table`(*schema*, *table*, *table_pkey*, *master_status*)

The method saves the table name along with the primary key definition in the table t_replica_tables. This is required in order to let the replay procedure which primary key to use replaying the update and delete. If the table is without primary key is not stored. A table without primary key is copied and the indices are create like any other table. However the replica doesn't work for the tables without primary key.

If the class variable master status is set then the master's coordinates are saved along with the table. This happens in general when a table is added to the replica or the data is refreshed with sync_tables.

Parameters

- **`schema`** – the schema name to store in the table t_replica_tables
- **`table`** – the table name to store in the table t_replica_tables
- **`table_pkey`** – a list with the primary key's columns. empty if there's no pkey
- **`master_status`** – the master status data .

`swap_schemas`()

The method loops over the schema_loading class dictionary and swaps the loading with the destination schemas performing a double rename. The method assumes there is a database connection active.

`swap_source_log_table`()

The method swaps the sources's log table and returns the next log table stored in the v_log_table array. The method expects an active database connection.

Returns The t_log_replica's active subpartition

Return type text

swap_tables()
The method loops over the tables stored in the class

truncate_table(*schema*, *table*)
The method truncates the table defined by schema and name. :param schema: the table's schema :param table: the table's name

unregister_table(*schema*, *table*)
This method is used to remove a table from the replica catalogue. The table is just deleted from the table sch_chameleon.t_replica_tables.

Parameters

- **schema** – the schema name where the table is stored
- **table** – the table name to remove from t_replica_tables

unset_lock_timeout()
The method sets the lock timeout using the value stored in the class attribute lock_timeout.

update_schema_mappings()
The method updates the schema mappings for the given source. Before executing the updates the method checks for the need to run an update and for any mapping already present in the replica catalogue. If everything is fine the database connection is set autocommit=false. The method updates the schemas in the table t_replica_tables and then updates the mappings in the table t_sources. After the final update the commit is issued to make the updates permanent.

Todo The method should run only at replica stopped for the given source. The method should also replay all the logged rows for the given source before updating the schema mappings to avoid to get an inconsistent replica.

upgrade_catalogue_v1()
The method upgrade a replica catalogue from version 1 to version 2. The original catalogue is not altered but just renamed. All the existing data are transferred into the new catalogue loaded using the create_schema.sql file.

upgrade_catalogue_v20()
The method applies the migration scripts to the replica catalogue version 2.0. The method checks that all sources are in stopped or ready state.

write_batch(*group_insert*)
Main method for adding the batch data in the log tables. The row data from group_insert are mogrified in CSV format and stored in the string like object csv_file.

psycopg2's copy expert is used to store the event data in PostgreSQL.

Should any error occur the procedure fallsback to insert_batch.

Parameters **group_insert** – the event data built in mysql_engine

write_ddl(*token*, *query_data*, *destination_schema*)
The method writes the DDL built from the tokenised sql into PostgreSQL.

Parameters

- **token** – the tokenised query
- **query_data** – query's metadata (schema,binlog, etc.)
- **destination_schema** – the postgresql destination schema determined using the schema mappings.

class pg_lib.**pgsql_source**

Bases: object

init_replica()

The method performs a full init replica for the given source

8.4 sql_util api documentation

class sql_util.**sql_token**

Bases: object

The class tokenises the sql statements captured by mysql_engine. Several regular expressions analyse and build the elements of the token. The DDL support is purposely limited to the following.

DROP PRIMARY KEY CREATE (UNIQUE) INDEX/KEY CREATE TABLE ALTER TABLE

The regular expression m_fkeys is used to remove any foreign key definition from the sql statement as we don't enforce any foreign key on the PostgreSQL replication.

build_column_dic(*inner_stat*)

The method builds a list of dictionaries with the column definitions.

The regular expression m_fields is used to find all the column occurrences and, for each occurrence, the method parse_column is called. If parse_column returns a dictionary, this is appended to the list col_parse.

Parameters **inner_stat** – The statement within the round brackets in CREATE TABLE

Returns cols_parse the list of dictionary with the column definitions

Return type list

build_key_dic(*inner_stat*, *table_name*)

The method matches and tokenise the primary key and index/key definitions in the create table's inner statement.

As the primary key can be defined as column or table constraint there is an initial match attempt with the regexp m_inline_pkeys. If the match is successful then the primary key dictionary is built from the match data. Otherwise the primary key dictionary is built using the eventual table key definition.

The method search for primary keys keys and indices defined in the inner_stat. The index name PRIMARY is used to tell pg_engine we are building a primary key. Otherwise the index name is built using the format (uk)idx_tablename[0:20] + counter. If there's a match for a primary key the composing columns are saved into pkey_cols.

The tablename limitation is required as PostgreSQL enforces a strict limit for the identifier name's lenght.

Each key dictionary have three keys. index_name, the index name or PRIMARY index_columns, a list with the column names non_unique, follows the MySQL's information schema convention and marks an index if is unique or not.

When the dictionary is built is appended to idx_list and finally returned to the calling method parse_create_table.s

Parameters

- **inner_stat** – The statement within the round brackets in CREATE TABLE
- **table_name** – The table name

Returns idx_list the list of dictionary with the index definitions

Return type list

parse_alter_table(*malter_table*)

The method parses the alter table match. As alter table can be composed of multiple commands the original statement (group 0 of the match object) is searched with the regexp m_alter_list. For each element in returned by findall the first word is evaluated as command. The parse alter table manages the following commands. DROP,ADD,CHANGE,MODIFY.

Each command build a dictionary alter_dic with at leaset the keys command and name defined. Those keys are respectively the commant itself and the attribute name affected by the command.

ADD defines the keys type and dimension. If type is enum then the dimension key stores the enumeration list.

CHANGE defines the key command and then runs a match with m_alter_change. If the match is successful the following keys are defined.

old is the old previous field name new is the new field name type is the new data type dimension the field's dimensions or the enum list if type is enum

MODIFY works similarly to CHANGE except that the field is not renamed. In that case we have only the keys type and dimension defined along with name and command.s

The class's regular expression self.m_ignore_keywords is used to skip the CONSTRAINT,INDEX and PRIMARY and FOREIGN KEY KEYWORDS in the alter command.

Parameters **malter_table** – The match object returned by the match method against tha alter table statement.

Returns stat_dic the alter table dictionary tokenised from the match object.

Return type dictionary

parse_column(*col_def*)

This method parses the column definition searching for the name, the data type and the dimensions. If there's a match the dictionary is built with the keys column_name, the column name data_type, the column's data type is nullable, the value is set always to yes except if the column is primary key (column name present in key_cols) enum_list,character_maximum_length,numeric_precision are the dimensions associated with the data type. The auto increment is set if there's a match for the auto increment specification.s

Parameters **col_def** – The column definition

Returns col_dic the column dictionary

Return type dictionary

parse_create_table(*sql_create*, *table_name*)

The method parse and generates a dictionary from the CREATE TABLE statement. The regular expression m_inner is used to match the statement within the round brackets.

This inner_stat is then cleaned from the primary keys, keys indices and foreign keys in order to get the column list. The indices are stored in the dictionary key "indices" using the method build_key_dic. The regular expression m_pars is used for finding and replacing all the commas with the | symbol within the round brackets present in the columns list. At the column list is also appended a comma as required by the regepx used in build_column_dic. The build_column_dic method is then executed and the return value is stored in the dictionary key "columns"

Parameters

- **sql_create** – The sql string with the CREATE TABLE statement
- **table_name** – The table name

Returns table_dic the table dictionary tokenised from the CREATE TABLE

Return type dictionary

`parse_rename_table`(*rename_statement*)

The method parses the rename statements storing in a list the old and the new table name.

Parameters **`rename_statement`** – The statement string without the RENAME TABLE

Returns rename_list, a list with the old/new table names inside

Return type list

`parse_sql`(*sql_string*)

The method cleans and parses the sql string A regular expression replaces all the default value definitions with a space. Then the statements are split in a list using the statement separator;

For each statement a set of regular expressions remove the comments, single and multi line. Parenthesis are surrounded by spaces and commas are rewritten in order to get at least one space after the comma. The statement is then put on a single line and stripped.

Different match are performed on the statement. RENAME TABLE CREATE TABLE DROP TABLE ALTER TABLE ALTER INDEX DROP PRIMARY KEY TRUNCATE TABLE

The match which is successful determines the parsing of the rest of the statement. Each parse builds a dictionary with at least two keys "name" and "command".

Each statement parse comes with specific addictional keys.

When the token dictionary is complete is added to the class list tokenised

Parameters **`sql_string`** – The sql string with the sql statements.

`quote_cols`(*cols*)

The method adds the " quotes to the column names. The string is converted to a list using the split method with the comma separator. The columns are then stripped and quoted with the "". Finally the list elements are rejoined in a string which is returned. The method is used in build_key_dic to sanitise the column names.

Parameters **`cols`** – The columns string

Returns The columns quoted between ".

Return type text

`reset_lists`()

The method resets the lists to empty lists after a successful tokenisation.

Python Module Index

Index