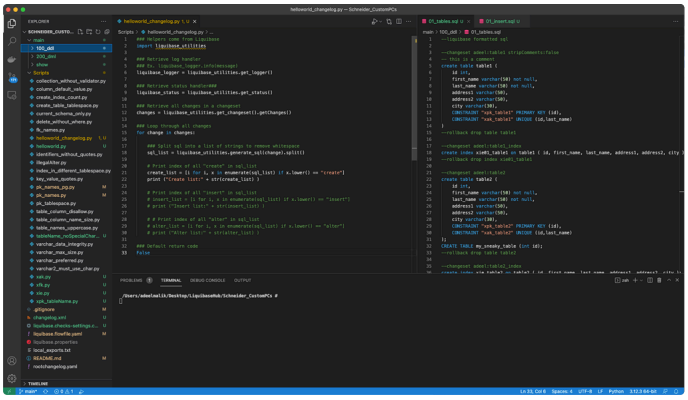
Python Custom Policy Checks Tutorial for Changelog Checks

Introduction

In this tutorial, we will explore authoring custom policy checks using Python.

We will write custom checks for changelog (i.e, `--checks-scope=changelog`) starting with an equivalent of a "hello world" example and then diving deeper.

All of the code editing in this tutorial was done using VS Code.



Prerequisites

- Install Liquibase 4.29.0+ (Release Liquibase v4.29.2 · liquibase/liquibase)
- Install Policy Checks extension (maven repository)
- Install Python 3.10.14 or higher
- Set this property: `checks-scripts-enabled=true`
- James' custom policy checks repo: https://github.com/liquibase/CustomPCs Connect your Github account

Verify that you have Python3 installed:

```
% python3 --version
Python 3.12.3
```

Private Repo

Clone `@James Bennett` 's repository (`git clone git@github.com:liquibase/CustomPCs.git`) and work in that repo because there are tons of sample Python scripts eavailable in `Scripts` directory

Public Repo

Clone @Adeel Malik 's repository (`git@github.com:adeelmalik78/python_policy_checks.git`) which contains examples shown in this tutorial.

Here is a starter `liquibase.properties` file:

```
changeLogFile: changelog.sql
url: jdbc:oracle:thin:@cs-oracledb.liquibase.net:1521/PP_DEV
username: liquibase_user
password: liquibase_user
liquibase.reports.enabled: false
checks-scripts-enabled=true
```

Changelog Setup

Setup changelog for what you want tested with Python custom policy checks.

The following changelog is compatible with both Oracle and Postgres. Name this file `changelog.sql` . These few SQL commands create tables with columns, primary keys and unique constraints. We can design custom policy checks around these. *No need to deploy these changesets*.

```
--liquibase formatted sql

--changeset adeel:table1 stripComments:false
-- this is a comment
create table table1 (
    id int,
    first_name varchar(50) not null,
    last_name varchar(50) not null,
    address1 varchar(50),
    address2 varchar(50),
    city varchar(30),
    CONSTRAINT "xpk_table1" PRIMARY KEY (id),
    CONSTRAINT "xak_table1" UNIQUE (id,last_name)
)
--rollback drop table table1

--changeset adeel:table1_index
create index xie01_table1 on table1 ( id, first_name, last_name, address1, address2, city );
--rollback drop index xie01_table1

--changeset adeel:table2
create table table2 (
    id int,
    first_name varchar(50) not null,
    last_name varchar(50) not null,
    address1 varchar(50),
    address2 varchar(50),
    city varchar(30),
    CONSTRAINT "xpk_table2" PRIMARY KEY (id),
```

```
    CONSTRAINT "xak_table2" UNIQUE (id,last_name)
);
CREATE TABLE my_sneaky_table (int id);
--rollback drop table table2

--changeset adeel:table2_index
create index xie_table2 on table2 ( id, first_name, last_name, address1, address2, city );
--rollback drop index xie_table2

--changeset adeel:person
CREATE TABLE person (
    id int,
    first_name varchar(50) NOT NULL,
    last_name varchar(50) NOT NULL,
    CONSTRAINT "pk_person" PRIMARY KEY (id),
    CONSTRAINT "ak_person" UNIQUE (id,last_name)
)
--rollback drop table person

--changeset adeel:person_index
create index xie03_person on person ( id, first_name, last_name );
--rollback drop index xie03_person

--changeset adeel:fk_table1
alter table table1 add constraint fk_table1 foreign key (id) references table2 (id);
--rollback alter table table1 drop constraint fk_table1;

--changeset adeel:xfk_table2
alter table table2 add constraint xfk_table2 foreign key (id) references table1 (id);
--rollback alter table table2 drop constraint xfk_table2;

--changeset adeel:employee
CREATE TABLE "employee+" (
    id int,
    first_name varchar(50) NOT NULL,
    last_name varchar(50) NOT NULL,
    CONSTRAINT "xpk_employee" PRIMARY KEY (id),
    CONSTRAINT "xak_employee" UNIQUE (id,last_name)
)
--rollback drop table "employee+"

--changeset amalik:insert_adeel
INSERT INTO person (id,first_name,last_name)
    VALUES (1,'Adeel','Ashraf');
--rollback DELETE FROM person WHERE id=1;

--changeset amalik:insert_amy
INSERT INTO person (id,first_name,last_name)
    VALUES (2,'Amy','Smith');
--rollback DELETE FROM person WHERE id=2;

--changeset amalik:insert_roderick
-- Let's create multiple INSERTs into this changeset!S
INSERT INTO person (id,first_name,last_name) VALUES (3,'Roderick','Bowser');
INSERT INTO person (id,first_name,last_name) VALUES (4,'Don','DeArmond III');
--rollback DELETE FROM person WHERE id=3;

--changeset amalik:update_adeel
```

```
UPDATE person
    SET first_name='Ryan', last_name='Campbell'
    WHERE id=1;
--rollback UPDATE person SET first_name='Adeel', last_name='Malik' WHERE id=1;
```

Python Basic Code

Create the following Python code and save it in `Scripts` directory. Name this script as `helloworld_changelog.py`.

This script executes once for each changeset.

- For each changeset, the SQL is split into list of strings, e.g, `["create", "table", "table1" ... ]`, and saved into “sql_list”
- Lines 21-22: Locate any CREATE statements by printing its index in sql_list. We use enumerate() to find multiple locations if there are more than one “CREATE” in the changeset.
- Lines 25-26: Locate any INSERT statements by printing its index in sql_list. We use enumerate() to find multiple locations if there are more than one “INSERT” in the changeset.
- Lines 29-30: Locate any ALTER statements by printing its index in sql_list. We use enumerate() to find multiple locations if there are more than one “ALTER” in the changeset.

```
### Helpers come from Liquibase
import liquibase_utilities

### Retrieve log handler
### Ex. liquibase_logger.info(message)
liquibase_logger = liquibase_utilities.get_logger()

### Retrieve status handler###
liquibase_status = liquibase_utilities.get_status()

### Retrieve all changes in a changeset
changes = liquibase_utilities.get_changeset().getChanges()

### Loop through all changes
for change in changes:

    ### Split sql into a list of strings to remove whitespace
    sql_list = liquibase_utilities.generate_sql(change).split()

    # Print index of all "create" in sql_list
    create_list = [i for i, x in enumerate(sql_list) if x.lower() == "create"]
    print ("Create list:" + str(create_list) )

    # Print index of all "insert" in sql_list
    # insert_list = [i for i, x in enumerate(sql_list) if x.lower() == "insert"]
    # print ("Insert list:" + str(insert_list) )

    # # Print index of all "alter" in sql_list
    # alter_list = [i for i, x in enumerate(sql_list) if x.lower() == "alter"]
    # print ("Alter list:" + str(alter_list) )

### Default return code
False
```

Disable All Checks

Let's create our `liquibase.checks-settings.conf` file:

```
% liquibase checks show
Press [1] to create a new checks settings file

% liquibase checks bulk-set --disable
Confirm: YES

% liquibase checks show --check-status=enabled
+------------+-------+--------+----------+---------------+-------------+
| Short Name | Scope | Status | Severity | Customization | Description |
+------------+-------+--------+----------+---------------+-------------+
```

First Custom Policy Check

Let's use the `CustomCheckTemplate` to write our first Python-based custom quality check:

```
% liquibase checks customize --check-name=CustomCheckTemplate

Short name:         HelloWorldchangelog
Severity:           0-4
Script description: List all objects
Script scope:       changelog
Script message:     This check will never trigger. This is a first custom policy check.
Script type:        python
Script path:        Scripts/helloworld_changelog.py
Script_Args:        <empty>
Requires snapshot:  false

% liquibase checks show --check-status=enabled
+---------------------+-----------+---------+----------+--------------------------------+---------------------
-----------+
| Short Name          | Scope     | Status  | Severity | Customization                  | Description
|
+---------------------+-----------+---------+----------+--------------------------------+---------------------
-----------+
| HelloWorldchangelog | changelog | enabled | 4        | SCRIPT_DESCRIPTION = List all  | Executes a custom
check        |
|                     |           |         |          | objects                        | script.
|
|                     |           |         |          | SCRIPT_SCOPE = CHANGELOG       |
|
|                     |           |         |          | SCRIPT_MESSAGE = The message   |
|
|                     |           |         |          | to display when the check is   |
|
|                     |           |         |          | triggered                      |
|
|                     |           |         |          | SCRIPT_TYPE = PYTHON           |
|
|                     |           |         |          | SCRIPT_PATH =                  |
|
|                     |           |         |          | Scripts/helloworld_changelog.p |
|
```

```
|                     |           |         |          | y                              |
|
|                     |           |         |          | SCRIPT_ARGS = null             |
|
|                     |           |         |          | REQUIRES_SNAPSHOT = false      |
|
+---------------------+-----------+---------+----------+--------------------------------+---------------------
-----------+
```

Run Your First Python Custom Policy Check

Let's run our policy check and examine the output. First run only displays location of CREATE statements. You can uncomment lines from python script to display location of INSERT and ALTER statements.

If you use the abovementioned `changelog.sql` script, you will notice that one of the changesets has multiple CREATE statements and we captured it in the output (see line 6 below). Also note that many changesets do not have CREATE statements and those show as empty `[]`.

```
% liquibase checks run --checks-scope=changelog

INFO: Checks executed against changes generated by PostgreSQL at jdbc:postgresql://localhost:5432/postgres.
Create list:[5]
Create list:[0]
Create list:[0, 30]
Create list:[0]
Create list:[0]
Create list:[0]
Create list:[]
Create list:[]
Create list:[0]
Create list:[]
Create list:[]
Create list:[]
Create list:[]
Checks-settings File:      liquibase.checks-settings.conf
================================================================================
Changesets Validated: in main/200_dml/01_insert.sql
  ID: insert_adeel; Author: amalik
  ID: insert_amy; Author: amalik
  ID: insert_roderick; Author: amalik
  ID: update_adeel; Author: amalik

Changesets Validated: in main/100_ddl/01_tables.sql
  ID: table1; Author: adeel
  ID: table1_index; Author: adeel
  ID: table2; Author: adeel
  ID: table2_index; Author: adeel
  ID: person; Author: adeel
  ID: person_index; Author: adeel
  ID: fk_table1; Author: adeel
  ID: xfk_table2; Author: adeel
  ID: employee; Author: adeel

Checks run against each changeset:
  Custom Check Template (Short names: HelloWorldchangelog)
```

Try rerunning the check by uncommenting lines that locate INSERT (25-26) and ALTER (lines 29-30) statements.

Task 1: (changelog) Create a Check for Table Name Must Not Have Special Characters ".*[*#+-].*"

Let's create a new python script in the `Scripts` directory and name it `tableName_noSpecialCharacters_changelog.py`.

We will need to catch two scenarios: `CREATE TABLE` statements and `RENAME TABLE` statements.

In this script, we will build on our earlier `helloworld_changelog.py` script.

Let's first adapt the script for `CREATE TABLE` statements.

- We commented out the `print` statement in line 24.
- Print out all tables found (line 24).
- We updated the code for finding the table names (lines 27- 32).
- Compare if the table name contain `[*#+-]` (line 37).
- If the table name does not match naming convention then we trigger the check (line 39).
- Print out relevant message to the user (lines 42-43).
- Exit from the check (line 45).

Copy/paste the following python code:

```
### Helpers come from Liquibase
import sys
import liquibase_utilities
import re

### Retrieve log handler
### Ex. liquibase_logger.info(message)
liquibase_logger = liquibase_utilities.get_logger()

### Retrieve status handler###
liquibase_status = liquibase_utilities.get_status()

### Retrieve all changes in a changeset
changes = liquibase_utilities.get_changeset().getChanges()

### Loop through all changes
for change in changes:

    ### Split sql into a list of strings to remove whitespace
    sql_list = liquibase_utilities.generate_sql(change).split()

    # Get index of all "create" in sql_list. Result will look like this [0] or [0,30]
    create_list = [i for i, x in enumerate(sql_list) if x.lower() == "create"]
    # print ("Create list:" + str(create_list) )

    # If there is a CREATE statement in the changeset ...
    if len(create_list) > 0:
        for create_loc in create_list:
            # If we find TABLE immediately after CREATE ... as in CREATE TABLE ...
            #    then very next index will contain the table name.
            if sql_list[create_loc + 1].lower() == "table":
                table_name_current = sql_list[create_loc + 2].lower()
                table_name_expected = "[*#+-]"

                # print ("CREATE table name is " + table_name_current)
```

```
36
37                 if re.search(table_name_expected, table_name_current):
38                     # if table_name_expected not in table_name_current:
39                     liquibase_status.fired = True       # indicates the custom check has been triggered
40
41                     # set the message of the custom check, which liquibase will return
42                     status_message = "CREATE table name " + f"{table_name_current}" + " contains one of these
   special characters " + f"{table_name_expected}"
43                     liquibase_status.message = status_message
44
45                     sys.exit(1)     # exit from the check
46
47 ### Default return code
48 False
```

Let's now adapt the script for `RENAME TABLE`:

Here is what the complete script would look like (new lines 50-69 added):

```
1  ### Helpers come from Liquibase
2  import sys
3  import liquibase_utilities
4  import re
5
6  ### Retrieve log handler
7  ### Ex. liquibase_logger.info(message)
8  liquibase_logger = liquibase_utilities.get_logger()
9
10 ### Retrieve status handler###
11 liquibase_status = liquibase_utilities.get_status()
12
13 ### Retrieve all changes in a changeset
14 changes = liquibase_utilities.get_changeset().getChanges()
15
16 ### Loop through all changes
17 for change in changes:
18
19     ### Split sql into a list of strings to remove whitespace
20     sql_list = liquibase_utilities.generate_sql(change).split()
21
22     # Get index of all "create" in sql_list. Result will look like this [0] or [0,30]
23     create_list = [i for i, x in enumerate(sql_list) if x.lower() == "create"]
24     # print ("Create list:" + str(create_list) )
25
26     # Get index of all "rename" in sql_list. Result will look like this [0] or [0,30]
27     rename_list = [i for i, x in enumerate(sql_list) if x.lower() == "rename"]
28
29     # If there is a CREATE statement in the changeset ...
30     if len(create_list) > 0:
31         for create_loc in create_list:
32             # If we find TABLE immediately after CREATE ... as in CREATE TABLE ...
33             #    then very next index will contain the table name.
34             if sql_list[create_loc + 1].lower() == "table":
35                 table_name_current = sql_list[create_loc + 2].lower()
36                 table_name_expected = "[*#+-]"
37
38                 print ("CREATE table name is " + table_name_current)
39
40                 if re.search(table_name_expected, table_name_current):
```

```
                    # if table_name_expected not in table_name_current:
                    liquibase_status.fired = True       # indicates the custom check has been triggered

                    # set the message of the custom check, which liquibase will return
                    status_message = "CREATE table name " + f"{table_name_current}" + " contains one of these special characters " + f"{table_name_expected}"
                    liquibase_status.message = status_message

                    sys.exit(1)     # exit from the check

   # If there is a RENAME statement in the changeset ...
    if len(rename_list) > 0:
        for rename_loc in rename_list:
            # If we find TABLE immediately after RENAME ... as in RENAME TABLE ...
            #    then very next index will contain the table name.
            if sql_list[rename_loc + 1].lower() == "table":
                table_name_current = sql_list[rename_loc + 2].lower()
                table_name_expected = "[*#+-]"

                print ("RENAME table name is " + table_name_current)

                if re.search(table_name_expected, table_name_current):
                    # if table_name_expected not in table_name_current:
                    liquibase_status.fired = True       # indicates the custom check has been triggered

                    # set the message of the custom check, which liquibase will return
                    status_message = "RENAME table name " + f"{table_name_current}" + " contains one of these special characters " + f"{table_name_expected}"
                    liquibase_status.message = status_message

                    sys.exit(1)     # exit from the check

### Default return code
False
```

Customize another `CustomCheckTemplate` policy check for foreign key naming convention:

```
% liquibase checks customize --check-name=CustomCheckTemplate

Short name:         TableNameNoSpecialCharactersChangelog
Severity:           0-4
Script description: Table name
Script scope:       changelog
Script message:     <empty>
Script type:        python
Script path:        Scripts/tableName_noSpecialCharacters_changelog.py
Script_Args:        <empty>
Requires snapshot:  false

% liquibase checks show --check-status=enabled
+---------------------------------------+-----------+---------+----------+--------------------------------+-----------------------------+
| Short Name                            | Scope     | Status  | Severity | Customization                  | Description                 |
+---------------------------------------+-----------+---------+----------+--------------------------------+-----------------------------+
| TableNameNoSpecialCharactersChangelog | changelog | enabled | 4        | SCRIPT_DESCRIPTION = Table     | Executes a custom check     |
```

```
|                                     |           |         |          | name                            |
script.                        |
|                                     |           |         |          | SCRIPT_SCOPE = CHANGELOG        |
|
|                                     |           |         |          | SCRIPT_MESSAGE = The message    |
|
|                                     |           |         |          | to display when the check is    |
|
|                                     |           |         |          | triggered                       |
|
|                                     |           |         |          | SCRIPT_TYPE = PYTHON            |
|
|                                     |           |         |          | SCRIPT_PATH =                   |
|
|                                     |           |         |          | tableName_noSpecialCharacters_  |
|
|                                     |           |         |          | changelog.py                    |
|
|                                     |           |         |          | SCRIPT_ARGS = null              |
|
|                                     |           |         |          | REQUIRES_SNAPSHOT = false       |
|
+-------------------------------------+-----------+---------+----------+---------------------------------+---
----------------------------+
```

Let's run this check. If you ran against the database populated with the `changelog.sql` provided earlier, you would see a single check triggered.

```
liquibase checks run --checks-scope=database

INFO: Checks executed against changes generated by PostgreSQL at jdbc:postgresql://localhost:5432/postgres.
CREATE table name is table1
CREATE table name is table2
CREATE table name is my_sneaky_table
CREATE table name is person
CREATE table name is "employee+"
RENAME table name is "employee+"
Checks-settings File:      liquibase.checks-settings.conf
================================================================================
CHANGELOG CHECKS
----------------
Checks completed validation of the changelog and found the following issues:

Check Name:         Custom Check Template (TableNameNoSpecialCharactersChangelog)
Changeset ID:       employee
Changeset Filepath: changelog.sql
Check Severity:     BLOCKER (Return code: 4)
Message:            CREATE table name "employee+" contains one of these special characters [*#+-]

Check Name:         Custom Check Template (TableNameNoSpecialCharactersChangelog)
Changeset ID:       employee_rename
Changeset Filepath: changelog.sql
Check Severity:     BLOCKER (Return code: 4)
Message:            RENAME table name "employee+" contains one of these special characters [*#+-]

Changesets Validated: in changelog.sql
  ID: table1; Author: adeel
  ID: table1_index; Author: adeel
```

```
  ID: table2; Author: adeel
  ID: table2_index; Author: adeel
  ID: person; Author: adeel
  ID: person_index; Author: adeel
  ID: fk_table1; Author: adeel
  ID: xfk_table2; Author: adeel
  ID: employee; Author: adeel
  ID: employee_rename; Author: adeel
  ID: insert_adeel; Author: amalik
  ID: insert_amy; Author: amalik
  ID: insert_roderick; Author: amalik
  ID: update_adeel; Author: amalik

Checks run against each changeset:
  Custom Check Template (Short names: TableNameNoSpecialCharactersChangelog)
```