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.
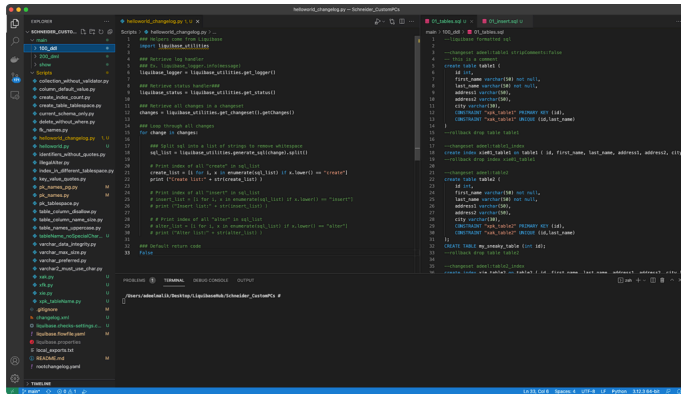
Prerequisites

- 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:

```
1  % python3 --version
2  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:

```
1  changeLogFile: changelog.sql
2  url: jdbc:oracle:thin:@cs-oracledb.liquibase.net:1521/PP_DEV
3  username: liquibase_user
4  password: liquibase_user
5  liquibase.reports.enabled: false
6  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_.

```
1  --liquibase formatted sql
2
3  --changeset adeel:table1 stripComments:false
4  -- this is a comment
5  create table liquibase_user.table1 (
6      id int,
7      first_name varchar(50) not null,
8      last_name varchar(50) not null,
9      address1 varchar(50),
10     address2 varchar(50),
11     city varchar(30),
12     CONSTRAINT "xpk_table1" PRIMARY KEY (id),
13     CONSTRAINT "xak_table1" UNIQUE (id,last_name)
14 )
15 --rollback drop table table1
16
17 --changeset adeel:table1_index
18 create index xie01_table1 on liquibase_user.table1 ( id, first_name, last_name, address1, address2, city );
19 --rollback drop index xie01_table1
20
21 --changeset adeel:table2
22 create table table2 (
23     id int,
24     first_name varchar(50) not null,
25     last_name varchar(50) not null,
26     address1 varchar(50),
27     address2 varchar(50),
28     city varchar(30),
29     CONSTRAINT "xpk_table2" PRIMARY KEY (id),
```

```
30        CONSTRAINT "xak_table2" UNIQUE (id,last_name)
31   );
32   CREATE TABLE my_sneaky_table (id int);
33   --rollback drop table table2;
34   --rollback drop table my_sneaky_table;
35
36   --changeset adeel:table2_index
37   create index xie_table2 on table2 ( id, first_name, last_name, address1, address2, city );
38   --rollback drop index xie_table2
39
40   --changeset adeel:person
41   CREATE TABLE public.person (
42        id int,
43        first_name varchar(50) NOT NULL,
44        last_name varchar(50) NOT NULL,
45        CONSTRAINT "pk_person" PRIMARY KEY (id),
46        CONSTRAINT "ak_person" UNIQUE (id,last_name)
47   )
48   --rollback drop table public.person
49
50   --changeset adeel:person_index
51   create index xie03_person on person ( id, first_name, last_name );
52   --rollback drop index xie03_person
53
54   --changeset adeel:fk_table1
55   alter table table1 add constraint fk_table1 foreign key (id) references table2 (id);
56   --rollback alter table table1 drop constraint fk_table1;
57
58   --changeset adeel:xfk_table2
59   alter table table2 add constraint xfk_table2 foreign key (id) references table1 (id);
60   --rollback alter table table2 drop constraint xfk_table2;
61
62   --changeset adeel:employee
63   CREATE TABLE "employee+" (
64        id int,
65        first_name varchar(50) NOT NULL,
66        last_name varchar(50) NOT NULL,
67        CONSTRAINT "xpk_employee" PRIMARY KEY (id),
68        CONSTRAINT "xak_employee" UNIQUE (id,last_name)
69   )
70   --rollback drop table "employee+"
71
72   --changeset amalik:insert_adeel
73   INSERT INTO person (id,first_name,last_name)
74        VALUES (1,'Adeel','Ashraf');
75   --rollback DELETE FROM person WHERE id=1;
76
77   --changeset amalik:insert_amy
78   INSERT INTO person (id,first_name,last_name)
79        VALUES (2,'Amy','Smith');
80   --rollback DELETE FROM person WHERE id=2;
81
82   --changeset amalik:insert_roderick
83   -- Let's create multiple INSERTs into this changeset!S
84   INSERT INTO person (id,first_name,last_name) VALUES (3,'Roderick','Bowser');
85   INSERT INTO person (id,first_name,last_name) VALUES (4,'Don','DeArmond III');
86   --rollback DELETE FROM person WHERE id=3;
87
```

```
88  --changeset amalik:update_adeel
89  UPDATE person
90      SET first_name='Ryan', last_name='Campbell'
91      WHERE id=1;
92  --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.

```python
1   ### Helpers come from Liquibase
2   import liquibase_utilities
3
4   ### Retrieve log handler
5   ### Ex. liquibase_logger.info(message)
6   liquibase_logger = liquibase_utilities.get_logger()
7
8   ### Retrieve status handler###
9   liquibase_status = liquibase_utilities.get_status()
10
11  ### Retrieve all changes in a changeset
12  changes = liquibase_utilities.get_changeset().getChanges()
13
14  ### Loop through all changes
15  for change in changes:
16
17      ### Split sql into a list of strings to remove whitespace
18      sql_list = liquibase_utilities.generate_sql(change).split()
19
20      # Print index of all "create" in sql_list
21      create_list = [i for i, x in enumerate(sql_list) if x.lower() == "create"]
22      print ("Create list:" + str(create_list) )
23
24      # Print index of all "insert" in sql_list
25      # insert_list = [i for i, x in enumerate(sql_list) if x.lower() == "insert"]
26      # print ("Insert list:" + str(insert_list) )
27
28      # # Print index of all "alter" in sql_list
29      # alter_list = [i for i, x in enumerate(sql_list) if x.lower() == "alter"]
30      # print ("Alter list:" + str(alter_list) )
31
32  ### Default return code
33  False
```

Disable All Checks

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

```
 1  % liquibase checks show
 2  Press [1] to create a new checks settings file
 3
 4  % liquibase checks bulk-set --disable
 5  Confirm: YES
 6
 7  % liquibase checks show --check-status=enabled
 8  +------------+-------+--------+----------+--------------+-------------+
 9  | Short Name | Scope | Status | Severity | Customization | Description |
10  +------------+-------+--------+----------+--------------+-------------+
```

First Custom Policy Check

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

```
 1  % liquibase checks customize --check-name=CustomCheckTemplate
 2
 3  Short name:         HelloWorldchangelog
 4  Severity:           0-4
 5  Script description: List all objects
 6  Script scope:       changelog
 7  Script message:     This check will never trigger. This is a first custom policy check.
 8  Script type:        python
 9  Script path:        Scripts/helloworld_changelog.py
10  Script_Args:        <empty>
11  Requires snapshot:  false
12
13  % liquibase checks show --check-status=enabled
14  +--------------------+-----------+---------+----------+------------------------------+-------------------
     -----------+
15  | Short Name         | Scope     | Status  | Severity | Customization                | Description
     |
16  +--------------------+-----------+---------+----------+------------------------------+-------------------
     -----------+
17  | HelloWorldchangelog | changelog | enabled | 4        | SCRIPT_DESCRIPTION = List all | Executes a custom
     check        |
18  |                     |           |         |          | objects                       | script.
     |
19  |                     |           |         |          | SCRIPT_SCOPE = CHANGELOG      |
     |
20  |                     |           |         |          | SCRIPT_MESSAGE = The message  |
     |
21  |                     |           |         |          | to display when the check is  |
     |
22  |                     |           |         |          | triggered                     |
     |
23  |                     |           |         |          | SCRIPT_TYPE = PYTHON          |
     |
24  |                     |           |         |          | SCRIPT_PATH =                 |
     |
25  |                     |           |         |          | Scripts/helloworld_changelog.p |
     |
```

```
26  |                    |          |         |          |  y                            |
    |
27  |                    |          |         |          |  SCRIPT_ARGS = null           |
    |
28  |                    |          |         |          |  REQUIRES_SNAPSHOT = false    |
    |
29  +--------------------+----------+---------+----------+-------------------------------+--------------------
    -----------+
```

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 `[]`.

```
 1  % liquibase checks run --checks-scope=changelog
 2
 3  INFO: Checks executed against changes generated by PostgreSQL at jdbc:postgresql://localhost:5432/postgres.
 4  Create list:[5]
 5  Create list:[0]
 6  Create list:[0, 30]
 7  Create list:[0]
 8  Create list:[0]
 9  Create list:[0]
10  Create list:[]
11  Create list:[]
12  Create list:[0]
13  Create list:[]
14  Create list:[]
15  Create list:[]
16  Create list:[]
17  Checks-settings File:      liquibase.checks-settings.conf
18  ================================================================================
19  Changesets Validated: in main/200_dml/01_insert.sql
20    ID: insert_adeel; Author: amalik
21    ID: insert_amy; Author: amalik
22    ID: insert_roderick; Author: amalik
23    ID: update_adeel; Author: amalik
24
25  Changesets Validated: in main/100_ddl/01_tables.sql
26    ID: table1; Author: adeel
27    ID: table1_index; Author: adeel
28    ID: table2; Author: adeel
29    ID: table2_index; Author: adeel
30    ID: person; Author: adeel
31    ID: person_index; Author: adeel
32    ID: fk_table1; Author: adeel
33    ID: xfk_table2; Author: adeel
34    ID: employee; Author: adeel
35
36  Checks run against each changeset:
37    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:

```python
### 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):
```

```
41                    # if table_name_expected not in table_name_current:
42                    liquibase_status.fired = True       # indicates the custom check has been triggered
43
44                    # set the message of the custom check, which liquibase will return
45                    status_message = "CREATE table name " + f"{table_name_current}" + " contains one of these
   special characters " + f"{table_name_expected}"
46                    liquibase_status.message = status_message
47
48                    sys.exit(1)     # exit from the check
49
50     # If there is a RENAME statement in the changeset ...
51     if len(rename_list) > 0:
52         for rename_loc in rename_list:
53             # If we find TABLE immediately after RENAME ... as in RENAME TABLE ...
54             #    then very next index will contain the table name.
55             if sql_list[rename_loc + 1].lower() == "table":
56                 table_name_current = sql_list[rename_loc + 2].lower()
57                 table_name_expected = "[*#+-]"
58
59                 print ("RENAME table name is " + table_name_current)
60
61                 if re.search(table_name_expected, table_name_current):
62                     # if table_name_expected not in table_name_current:
63                     liquibase_status.fired = True       # indicates the custom check has been triggered
64
65                     # set the message of the custom check, which liquibase will return
66                     status_message = "RENAME table name " + f"{table_name_current}" + " contains one of these
   special characters " + f"{table_name_expected}"
67                     liquibase_status.message = status_message
68
69                     sys.exit(1)     # exit from the check
70
71 ### Default return code
72 False
```

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

```
 1 % liquibase checks customize --check-name=CustomCheckTemplate
 2
 3 Short name:          TableNameNoSpecialCharactersChangelog
 4 Severity:            0-4
 5 Script description: Table name
 6 Script scope:        changelog
 7 Script message:     <empty>
 8 Script type:         python
 9 Script path:         Scripts/tableName_noSpecialCharacters_changelog.py
10 Script_Args:         <empty>
11 Requires snapshot:  false
12
13 % liquibase checks show --check-status=enabled
14 +---------------------------------------+-----------+---------+----------+-----------------------------+---
   ----------------------------+
15 | Short Name                            | Scope     | Status  | Severity | Customization                |
   Description                   |
16 +---------------------------------------+-----------+---------+----------+-----------------------------+---
   ----------------------------+
17 | 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
```

```
31    ID: table2; Author: adeel
32    ID: table2_index; Author: adeel
33    ID: person; Author: adeel
34    ID: person_index; Author: adeel
35    ID: fk_table1; Author: adeel
36    ID: xfk_table2; Author: adeel
37    ID: employee; Author: adeel
38    ID: employee_rename; Author: adeel
39    ID: insert_adeel; Author: amalik
40    ID: insert_amy; Author: amalik
41    ID: insert_roderick; Author: amalik
42    ID: update_adeel; Author: amalik
43
44  Checks run against each changeset:
45    Custom Check Template (Short names: TableNameNoSpecialCharactersChangelog)
```

Task 2: (changelog) Create a Check to Disallow Schema Prefix (except "public")

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

We will need to catch for the use of schema prefix which is always used in the form `schemaName.objectName` notation. This means that we can look for the presence of a `.` in the string.

In this script:

- Create a Python function called `find_string_index` which returns the index of the schema character (".") when the changeset is split into a list of strings (lines 6-12).
- Iterate of each change in the changeset (to catch more than one SQL statement in a changeset) (line 25)
- Define schema character as `schemaString = "."` (line 30).
- Invoke the `find_string_index` function and return the index value in the list containing the `schemaString` (line 32)
- (Optional) Print out all tables found (line 33).
- Check if we detected the index, `index != 0` (line 35)
- If the schema name is "public", we ignore this line 38-39)
- Trigger the check (line 41)
- Print out relevant message to the user (lines 44-45).
- Exit from the check (line 47).

Copy/paste the following python code:

```
1   ### Helpers come from Liquibase
2   import sys
3   import liquibase_utilities
4   import re
5
6   def find_string_index(lst, schemaChar):
7       result = 0
8       for i, string in enumerate(lst):
9           if schemaChar in string:
10              result = i
11              # result.append((i, string.index(schemaChar)))
12      return result
13
14  ### Retrieve log handler
15  ### Ex. liquibase_logger.info(message)
```

```
16  liquibase_logger = liquibase_utilities.get_logger()

17

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

20

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

23

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

26

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

29

30      schemaString = "."

31

32      index = find_string_index(sql_list, schemaString)
33      # print ("Schema found at index " + str(index) + ": " + str(sql_list[index]) )

34

35      if (index != 0):
36          # Found schema prefix
37          # Now check if the schema is "public". If not public, then trigger the check
38          publicString = "public"
39          if not re.search(publicString, sql_list[index]):

40

41              liquibase_status.fired = True       # indicates the custom check has been triggered

42

43              # set the message of the custom check, which liquibase will return
44              status_message = "No schema prefix allowed. Found schema prefix: [ " + f"{sql_list[index]}" + " ]
    in the changeset."
45              liquibase_status.message = status_message

46

47              sys.exit(1)      # exit from the check

48

49  ### Default return code
50  False
```

Customize another `CustomCheckTemplate` policy check for disallowing schema in changelog:

```
1   % liquibase checks customize --check-name=CustomCheckTemplate
2
3   Short name:          DisallowSchemaInChangelog
4   Severity:            0-4
5   Script description:
6   Script scope:        changelog
7   Script message:      <empty>
8   Script type:         python
9   Script path:         Scripts/disallow_schema_changelog.py
10  Script_Args:         <empty>
11  Requires snapshot:   false
12
13  % liquibase checks show --check-status=enabled
14  +--------------------------+-----------+---------+----------+-------------------------------+---------------
    -----------------+
15  | Short Name               | Scope     | Status  | Severity | Customization                 | Description
    |
16  +--------------------------+-----------+---------+----------+-------------------------------+---------------
    -----------------+
```

```
17 | DisallowSchemaInChangelog | changelog | enabled | 3         | SCRIPT_DESCRIPTION = Custom   | Executes a
   custom check        |
18 |                             |           |         |          | check                         | script.
   |
19 |                             |           |         |          | SCRIPT_SCOPE = CHANGELOG       |
   |
20 |                             |           |         |          | SCRIPT_MESSAGE = The message   |
   |
21 |                             |           |         |          | to display when the check is   |
   |
22 |                             |           |         |          | triggered                     |
   |
23 |                             |           |         |          | SCRIPT_TYPE = PYTHON          |
   |
24 |                             |           |         |          | SCRIPT_PATH =                 |
   |
25 |                             |           |         |          | Scripts/disallow_schema_change |
   |
26 |                             |           |         |          | log.py                        |
   |
27 |                             |           |         |          | SCRIPT_ARGS = null            |
   |
28 |                             |           |         |          | REQUIRES_SNAPSHOT = false     |
   |
29 +-------------------------+-----------+---------+----------+-------------------------------+----------------
   -----------------+
```

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

```
 1  INFO: Checks executed against changes generated by Oracle at jdbc:oracle:thin:@cs-
    oracledb.liquibase.net:1521/PP_DEV.
 2  Checks-settings File:      liquibase.checks-settings.conf
 3  ================================================================================
 4  CHANGELOG CHECKS
 5  ----------------
 6  Checks completed validation of the changelog and found the following issues:
 7
 8  Check Name:         Custom Check Template (DisallowSchemaInChangelog)
 9  Changeset ID:       table1
10  Changeset Filepath: changelog.sql
11  Check Severity:     CRITICAL (Return code: 3)
12  Message:            No schema prefix allowed. Found schema prefix: [ liquibase_user.table1 ] in the changeset.
13
14  Check Name:         Custom Check Template (DisallowSchemaInChangelog)
15  Changeset ID:       table1_index
16  Changeset Filepath: changelog.sql
17  Check Severity:     CRITICAL (Return code: 3)
18  Message:            No schema prefix allowed. Found schema prefix: [ liquibase_user.table1 ] in the changeset.
19
20  Check Name:         Custom Check Template (DisallowSchemaInChangelog)
21  Changeset ID:       table2
22  Changeset Filepath: changelog.sql
23  Check Severity:     CRITICAL (Return code: 3)
24  Message:            No schema prefix allowed. Found schema prefix: [ "LIQUIBASE_USER".table2 ] in the
    changeset.
25
26  Changesets Validated: in changelog.sql
```

```
27    ID: table1; Author: adeel
28    ID: table1_index; Author: adeel
29    ID: table2; Author: adeel
30    ID: table2_index; Author: adeel
31    ID: person; Author: adeel
32    ID: person_index; Author: adeel
33    ID: fk_table1; Author: adeel
34    ID: xfk_table2; Author: adeel
35    ID: employee; Author: adeel
36    ID: insert_adeel; Author: amalik
37    ID: insert_amy; Author: amalik
38    ID: insert_roderick; Author: amalik
39    ID: update_adeel; Author: amalik
40
41  Checks run against each changeset:
42    Custom Check Template (Short names: DisallowSchemaInChangelog)
```