SQVID

Marek Suppa

Sep 08, 2020

TABLE OF CONTENTS:

SQVID, the **Simple sQl Validator of varIous Datasources** is a framework for validating any type of data source that can be queried via SQL with SQLAlchemy.

It aims to be a simplified and extensible counterpart to validation of dbt models or data assertions of dataform that does not require you to use the full dbt or dataform and still ensure your data is automatically validated to be what you expect it to be. This allows SQVID to be used on all sorts of data sources: from CSVs and spreadsheets to massive databases.

You can easily use SQVID to serve as a "sanity check" of your processing pipeline or as a testing framework for your various ETL processes.

CHAPTER

ONE

INSTALLATION

```
pip install sqvid
```

1.1 Getting Started

Let us consider a database table called `suppliers` that would result from executing the following code snippet in a SQLite database called `test_sqvid_db`.

```
CREATE TABLE `suppliers` (
  `SupplierID` int NOT NULL,
  `SupplierName` varchar(255) DEFAULT NULL,
  `ContactName` varchar(255) DEFAULT NULL,
  `Address` varchar(255) DEFAULT NULL,
  `City` varchar(255) DEFAULT NULL,
  `PostalCode` varchar(255) DEFAULT NULL,
  `Country` varchar(255) DEFAULT NULL,
  `Phone` varchar(255) DEFAULT NULL
);

INSERT INTO `suppliers` (`SupplierID`, `SupplierName`, `ContactName`, `Address`,␣
↪`City`, `PostalCode`, `Country`, `Phone`) VALUES
(1, "Exotic Liquid", "Charlotte Cooper", "49 Gilbert St.", "Londona", "EC1 4SD", "UK",
↪ "(171) 555-2222"),
(2, "New Orleans Cajun Delights", "Shelley Burke", "P.O. Box 78934", "New Orleans",
↪"70117", "USA", "(100) 555-4822"),
(3, "Grandma Kelly's Homestead", "Regina Murphy", "707 Oxford Rd.", "Ann Arbor",
↪"48104", "USA", "(313) 555-5735"),
(4, "Tokyo Traders", "Yoshi Nagase", "9-8 Sekimai Musashino-shi", "Tokyo", "100",
↪"Japan", "(03) 3555-5011"),
(5, "Cooperativa de Quesos 'Las Cabras'", "Antonio del Valle Saavedra", "Calle del␣
↪Rosal 4", "Oviedo", "33007", "Spain", "(98) 598 76 54"),
```

In order to validate that this table contains the data we would expect it to, we can put together the following SQVID validation config:

```
[general]
sqla = "sqlite:///test_sqvid_db.sqlite"
db_name = 'test_sqvid_db'

[[test_sqvid_db.suppliers.SupplierID]]
```

(continues on next page)

(continued from previous page)

```
validator = 'unique'

[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
args = {min = 1, max = 256}
```

Note that the the validation config file is fromated using TOML *– you can find a very nice tutorial on this formatting language at* LearnXinYMinutes.

The `[general]` section specifies SQLAlchemy connection string in `sqla` and the name of the DB that is going to have its data validated in `db_name`.

The other sections specify the various validations performed on this DB. In particular the table `suppliers` and data in its column `SupplierID` is being validated via two validators: the `unique` validator ensures that each value in this column occurs only once and the `in_range` validator checks whether all data points in this column fall withing the `min` and `max` range specified via parameters of the same name in `args`.

Once we save a validation config like this one into a file (say `validate_suppliers.toml`), SQVID validation tests can be invoked in the following way:

```
sqvid --config ./validate_suppliers.toml
```

This should provide output close to the following:

```
PASSED: Validation on [test_sqvid_db] suppliers.SupplierID of unique
PASSED: Validation on [test_sqvid_db] suppliers.SupplierID of in_range({'min': 1, 'max
↪': 256})
```

Since all tests passed, `sqvid` would finish with exit code `0`.

1.1.1 Failing validations

What happens when a SQVID validation test fails? We can easily see that by slightly changing the config file from the above:

```
[general]
sqla = "sqlite:///test_sqvid_db.sqlite"
db_name = 'test_sqvid_db'

[[test_sqvid_db.suppliers.SupplierID]]
validator = 'unique'

[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
args = {min = 3, max = 256}
```

Note that the contents stayed the same, except for the final line where the `min` parameter has been set to `3`. If we now save this file (to say `./validate_suppliers_fail.toml`), we can again execute SQVID tests in a similar way:

```
sqvid --config ./validate_suppliers_fail.toml
```

The output should change to something like this:

```
PASSED: Validation on [test_sqvid_db] suppliers.SupplierID of unique
FAILED: Validation on [test_sqvid_db] suppliers.SupplierID of in_range({'min': 3, 'max
↪': 256})
Offending 2 rows:
+-------------+-----------------------------+-------------------+-----------------
↪+---------------+-------------+-----------+------------------+
|  SupplierID  |  SupplierName                |  ContactName       |  Address
↪|  City         |  PostalCode  |  Country  |  Phone           |
+-------------+-----------------------------+-------------------+-----------------
↪+---------------+-------------+-----------+------------------+
|           1  |  Exotic Liquid               |  Charlotte Cooper  |  49 Gilbert St.
↪|  Londona      |  EC1 4SD     |  UK       |  (171) 555-2222  |
|           2  |  New Orleans Cajun Delights  |  Shelley Burke     |  P.O. Box 78934
↪|  New Orleans  |  70117       |  USA      |  (100) 555-4822  |
+-------------+-----------------------------+-------------------+-----------------
↪+---------------+-------------+-----------+------------------+
```

As we would expect, the `unique` validation still passed while the `in_range` validation failed on the two rows which have their `SupplierID` outside of the `[3, 256]` range.

Since some tests failed, `sqvid` would finish with exit code `1`.

1.2 The SQVID config file

Every SQVID config file has basically two components: the `[general]` section and all the other sections that describe the configuration of respective validations.

1.2.1 The `[general]` section

This section is generally used to set up parameters that affect all validations that are defined in a single SQVID config file.

A sample `[general]` section may look as follows:

```
[general]
sqla = "sqlite:///test_sqvid_db.sqlite"
db_name = 'test_sqvid_db'
```

`sqla` This parameter sets the so called Database URL that SQLAlchemy uses to connect to the database in which the data to be validated is located.

`db_name` The name of the database in which the data to be validated is located. Although some connection engines would already have this specified in the `sqla` parameter, having it specified separately allows us much greater flexibility when generating validations.

`limit` By default, `sqvid` will report all rows that did not conform to a specific validation. This may not be desirable in all cases (i.e. when the expected amount of non-conforming rows is large)

The limit can be specified on per validation config basis using the `limit` parameter, such as in the following:

```
[general]
sqla = "sqlite:///test_sqvid_db.sqlite"
db_name = 'test_sqvid_db'
limit = 50
```

1.2.2 Definition of validations

Using the `db_name` mentioned above, validations are defined as arrays of TOML tables in the format of `$db_name.$table_name.$column_name`. Here is a simple example:

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'unique'
```

As we can see the `$db_name` would in this case be `test_sqvid_db`, the `suppliers` would be the `$table_name` and `SupplierID` the `$column_name`.

Note: the double brackets (that is `[[` and `]]`) around `test_sqvid_db.suppliers.SupplierID` denote "arrays of tables". It means that `test_sqvid_db.suppliers.SupplierID` will not be a single table but rather an array. Among other things this allows us to define various validations for a single column. Here is a quick example:

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'unique'

[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
args = {min = 1, max = 256}
```

This example defines two validations on the `test_sqvid_db.suppliers.SupplierID` column: one with the `unique` validator and one with the `in_range` validator.

Once the database, table and column in which the data we want to validate is define, we can specify which validator to use and with what parameters. This can be done using he following set of parameters:

`validator` This (string) parameter needs to contain one of the validators described in *Validators*. For example:

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'unique'
```

`args` Arguments to be passed to the specified validator. They expressed as a TOML table and TOML supports various ways of describing those. All of the following definitions are functionally equal:

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
args = {min = 1, max = 256}

# is equivalent to

[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
args.min = 1
args.max = 256

# is again equivalen to

[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'

  [test_sqvid_db.suppliers.SupplierID.args]
  min = 1
  max = 256
```

For short arguments (and short values) the first way is preferable. In all other cases the second one should make a bit more sense, especially in terms of readability.

custom_column Validating a single columns is sometimes of little use as what we would really want to validate is result of combining multiple columns. That is exactly what `custom_column` is for: it allows for any custom SQL to be considered a "virtual" column onto which a validation can then be applied.

Suppose we would like to check whether the combination of the `SupplierID` and `SupplierName` is unique. We can easily do that using the `custom_column` parameter as in the example below:

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'unique'
custom_column = "SupplierID || '-' || SupplierName"
```

The fact that a custom column is used gets represented in the printed report as well. The validator definition above would have resulted in something like the following:

```
PASSED: Validation on [test_sqvid_db] suppliers.SupplierID (customized as
↪'SupplierID || '-' || SupplierName') of unique
```

severity A "severity" of a validation defines what happens when it fails. It can be set to either `error` (the default) or `warn`. When set to `warn`, the validation will still report the results (i.e. which rows did not meet the validation criteria) but the validation will otherwise behave as if it passed.

Here is a quick example. Provided that the IDs stored in SupplierID would not have been unique, the following validator would only cause a warning – the `sqvid` call would still have ended with exit code 0, as if the validation has passed.

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
severity = 'warn'
args = {min = 3, max = 256}
```

Severity set to `warn` will also be presented in `sqvid`'s output – the sample output from the validation defined above can be found below:

```
FAILED (WARN ONLY): Validation on [test_sqvid_db] suppliers.SupplierID of in_
↪range({'min': 3, 'max': 256})
Offending 2 rows:
+--------------+------------------------------+--------------------+--------------
↪----+---------------+-------------+-----------+------------------+
|  SupplierID  |  SupplierName                |  ContactName       |  Address     ␣
↪    |  City         |  PostalCode  |  Country  |  Phone           |
+--------------+------------------------------+--------------------+--------------
↪----+---------------+-------------+-----------+------------------+
|           1  |  Exotic Liquid               |  Charlotte Cooper  |  49 Gilbert␣
↪St.  |  Londona      |  EC1 4SD     |  UK       |  (171) 555-2222  |
|           2  |  New Orleans Cajun Delights  |  Shelley Burke     |  P.O. Box␣
↪78934  |  New Orleans  |  70117       |  USA      |  (100) 555-4822  |
+--------------+------------------------------+--------------------+--------------
↪----+---------------+-------------+-----------+------------------+
```

report_columns With tables that have a large number of columns, it becomes quite difficult to just print out all the columns relevant to a specific validation that failed.

All columns are printed (or reported) by default. Using the `report_columns` parameter a subset of the column list can be selected.

For instance the following validation declaration

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
args = {min = 3, max = 256}
```

would result in the following output:

```
FAILED: Validation on [test_sqvid_db] suppliers.SupplierID of in_range({'min': 3,
↪'max': 256})
Offending 2 rows:
+--------------+------------------------------+--------------------+--------------
↪----+--------------+-------------+-----------+------------------+
|  SupplierID  |  SupplierName                |  ContactName       |  Address     ␣
↪    |  City        |  PostalCode  |  Country  |  Phone           |
+--------------+------------------------------+--------------------+--------------
↪----+--------------+-------------+-----------+------------------+
|           1  |  Exotic Liquid               |  Charlotte Cooper  |  49 Gilbert␣
↪St.  |  Londona     |  EC1 4SD     |  UK       |  (171) 555-2222  |
|           2  |  New Orleans Cajun Delights  |  Shelley Burke     |  P.O. Box␣
↪78934  |  New Orleans  |  70117       |  USA      |  (100) 555-4822  |
+--------------+------------------------------+--------------------+--------------
↪----+--------------+-------------+-----------+------------------+
```

Whereas the following validation

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
report_columns = [
  'SupplierID',
  'SupplierName'
]
args = {min = 3, max = 256}
```

would then output the following:

```
FAILED: Validation on [test_sqvid_db] suppliers.SupplierID of in_range({'min': 3,
↪'max': 256})
Offending 2 rows:
+--------------+------------------------------+
|  SupplierID  |  SupplierName                |
+--------------+------------------------------+
|           1  |  Exotic Liquid               |
|           2  |  New Orleans Cajun Delights  |
+--------------+------------------------------+
```

`limit` The limit of rows to report can also be specified per particular validation. This value defaults to the `limit` set on the validation config level.

For instance

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
report_columns = [
  'SupplierID',
  'SupplierName'
]
args = {min = 3, max = 256}
limit = 1
```

would report only a single offending row.

This configuration option also allows one to selectively turn off the `limit` set on the validation config level. For instance

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
report_columns = [
  'SupplierID',
  'SupplierName'
]
args = {min = 3, max = 256}
limit = false
```

would report all offending rows, regardless of what the setting of `limit` in the `general` section was.

1.3 Validators

Validators are the corner stone of SQVID. They basically take a look at a specific column and check whether the values in it are "valid". If they are not, the validation fails and the rows that do not pass the validation requirement are returned.

1.3.1 Available validators

sqvid.validators.**accepted_values**(*table*, *column*, *args=None*)

Check that a column contains only specified values.

Parameters **vals** (*list*) – a list of values

Example

```
[[test_sqvid_db.suppliers.Country]]
validator = 'accepted_values'
args.vals  = [
  'USA',
  'UK',
  'Spain',
  'Japan',
  'Germany',
  'Australia',
  'Sweden',
  'Finland',
  'Italy',
  'Brazil',
  'Singapore',
  'Norway',
  'Canada',
  'France',
  'Denmark',
  'Netherlands'
]
```

sqvid.validators.**custom_sql**(*table*, *column*, *args=None*)

Execute a custom (optionally Jinja-formatted) SQL query and fail if non-zero number of rows is returned.

Either `query` or `query_file` parameter needs to be provided. All the other arguments are passed as Jinja variables and can be used to build the query.

Parameters

- **query** (*str*) – query to be executed (optional).
- **query_file** (*str*) – path to the file in which the query to be executed can be found (optional)

Example

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'custom_sql'
args.query_file = './tests/queries/tables_equal_rows.sql'
args.other_table = 'suppliers_copy'
```

sqvid.validators.**in_range**(*table*, *column*, *args=None*)

Check whether values in a column fall within a specific range.

Parameters

- **min** (*int*) – the minimum value of the range
- **max** (*int*) – the maximum value of the range

Example

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'in_range'
args = {min = 1, max = 256}
```

sqvid.validators.**not_null**(*table*, *column*, *args=None*)

Check that a column contains only non-*NULL* values.

Example

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'not_null'
```

sqvid.validators.**unique**(*table*, *column*, *args=None*)

Check whether values in a column are unique.

Example

```
[[test_sqvid_db.suppliers.SupplierID]]
validator = 'unique'
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX