Hands-on Lab: Verifying Data Quality for a Data Warehouse

Estimated time needed: 30 minutes

Purpose of the Lab:

The primary purpose of this lab is to instruct participants on the process of conducting thorough data quality checks in a data warehousing environment. It focuses on using a Python-based framework within a PostgreSQL database to validate data integrity. Key areas of emphasis include identifying null values, duplicates, and invalid entries, as well as verifying data ranges. The lab aims to equip learners with the necessary skills to set up and utilize a testing framework for data validation, ensuring data accuracy and consistency.

Benefits of Learning the Lab:

Engaging in this lab offers several benefits, particularly in enhancing one's capabilities in data management and quality assurance. Learners will gain hands-on experience in implementing automated data quality checks, a skill crucial for maintaining the reliability of data in real-world applications. This proficiency is especially beneficial for professionals working with large datasets, as it ensures the integrity of data used for analysis and decision-making. Moreover, understanding these concepts is essential for anyone aspiring to specialize in data science, database administration, or any field that relies heavily on accurate and reliable data.

Objectives

In this lab, you will:

- Check Null values
- Check Duplicate values
- Check Min Max
- Check Invalid values
- Generate a report on data quality

About Skills Network Cloud IDE

Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands on labs for course and project related labs. Theia is an open source IDE (Integrated Development Environment), that can be run on desktop or on the cloud. To complete this lab, we will be using the Cloud IDE based on Theia running in a Docker container.

Important Notice about this lab environment

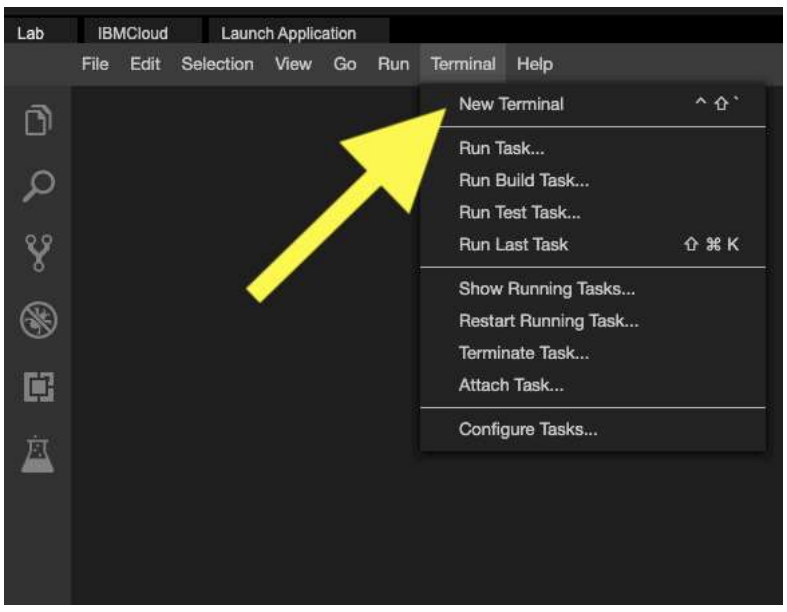
Please be aware that sessions for this lab environment are not persistent. A new environment is created for you every time you connect to this lab. Any data you may have saved in an earlier session will get lost. To avoid losing your data, please plan to complete these labs in a single session.

Exercise 1 - Getting the environment ready

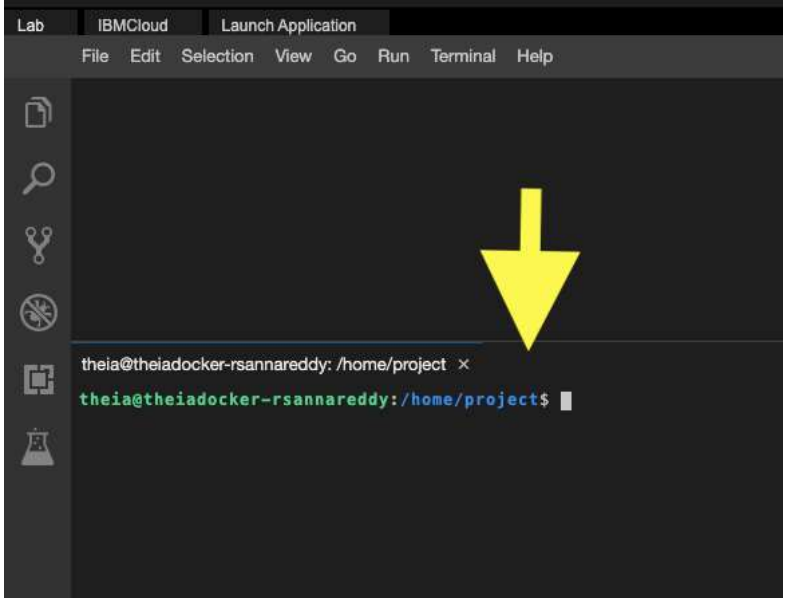
In this exercise, you will get the environment ready so that we can perform the data quality checks.

Step 1: Start the postgresql server.

Open a new terminal, by clicking on the menu bar and selecting **Terminal->New Terminal**, as shown in the image below.



This will open a new terminal at the bottom of the screen.



Run the commands below on the newly opened terminal. (You can copy the code by clicking on the little copy button on the bottom right of the codeblock below and then paste it wherever you wish.)

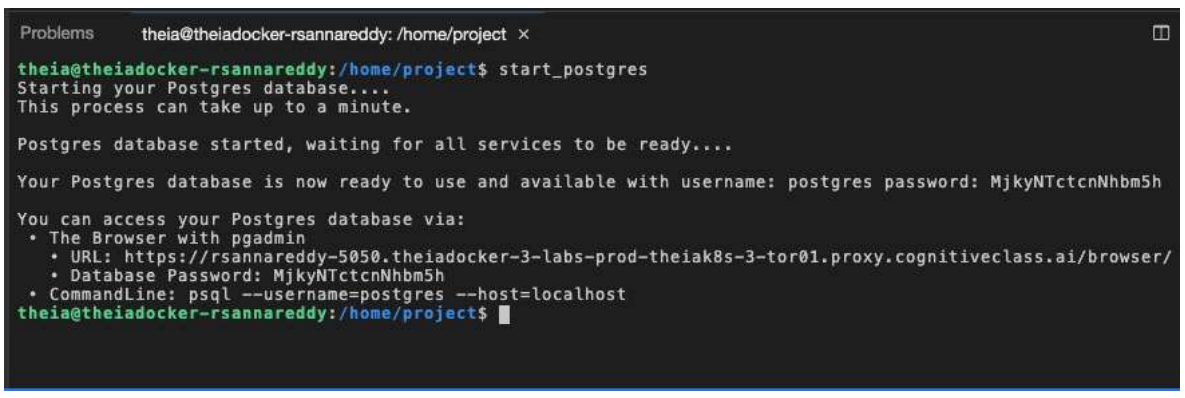
Start the PostgreSQL server, by running the command below.

```
1. 1
```

```
1. start_postgres
```

Copied! Executed!

You should see an output similar to the one below.



Step 2: Download the staging area setup script.

Run the command below to download the staging area setup script.

```
1. 1
```

```
1. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0260EN-SkillsNetwork/labs/Verifying%20Data%20Quality%20for%20a%20Data%20Warehouse/setup_staging_area.sh
```

Copied! Executed!

Step 3: Run the setup script.

Run the command below to execute the staging area setup script.

```
1. 1
```

```
1. bash setup_staging_area.sh
```

Copied! Executed!

When you see a message `Successfully setup the staging area` you are ready to perform data quality checks.

Exercise 2 - Getting the testing framework ready

You can perform most of the data quality checks by manually running sql queries on the data warehouse.

It is a good idea to automate these checks using custom programs or tools. Automation helps you to easily

- create new tests,
- run tests,
- and schedule tests.

We will be using a python based framework to run the data quality tests.

Step 1: Download the framework.

Run the commands below to download the framework

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
```

```
1. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0260EN-SkillsNetwork/labs/Verifying%20Data%20Quality%20for%20a%20Data%20Warehouse/dataqualitychecks.py
2.
3. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0260EN-SkillsNetwork/labs/Verifying%20Data%20Quality%20for%20a%20Data%20Warehouse/dbconnect.py
4.
5. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0260EN-SkillsNetwork/labs/Verifying%20Data%20Quality%20for%20a%20Data%20Warehouse/mytests.py
6.
7. wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0260EN-SkillsNetwork/labs/Verifying%20Data%20Quality%20for%20a%20Data%20Warehouse/generate-data-quality-report.py
8.
9. ls
```

Copied! Executed!

Step 2: Install the python driver for Postgresql.

Run the command below to install the python driver for Postgresql database

```
1. 1
```

```
1. python3 -m pip install psycopg2
```

Copied! Executed!

Step 3: Test database connectivity.

Now we need to check

- if the Postgresql python driver is installed properly.
- if Postgresql server is up and running.
- if our micro framework can connect to the database.

The command below to check all the above cases.

```
1. 1
```

```
1. python3 dbconnect.py
```

Copied! Executed!

If all goes well, you should a message `Successfully connected to database`.

The command also disconnects from the server with a message `Connection closed`.

Exercise 3 - Create a sample data quality report

Run the command below to install pandas.

```
1. 1
1. python3 -m pip install pandas tabulate
```

Copied! Executed!

Run the command below to generate a sample data quality report.

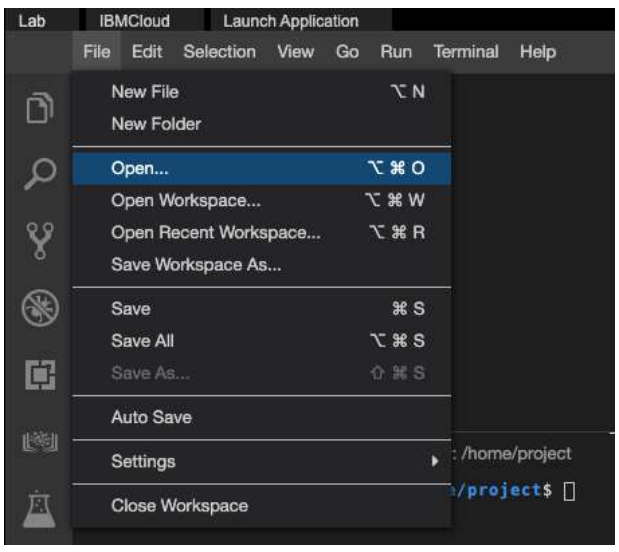
```
1. 1
1. python3 generate-data-quality-report.py
```

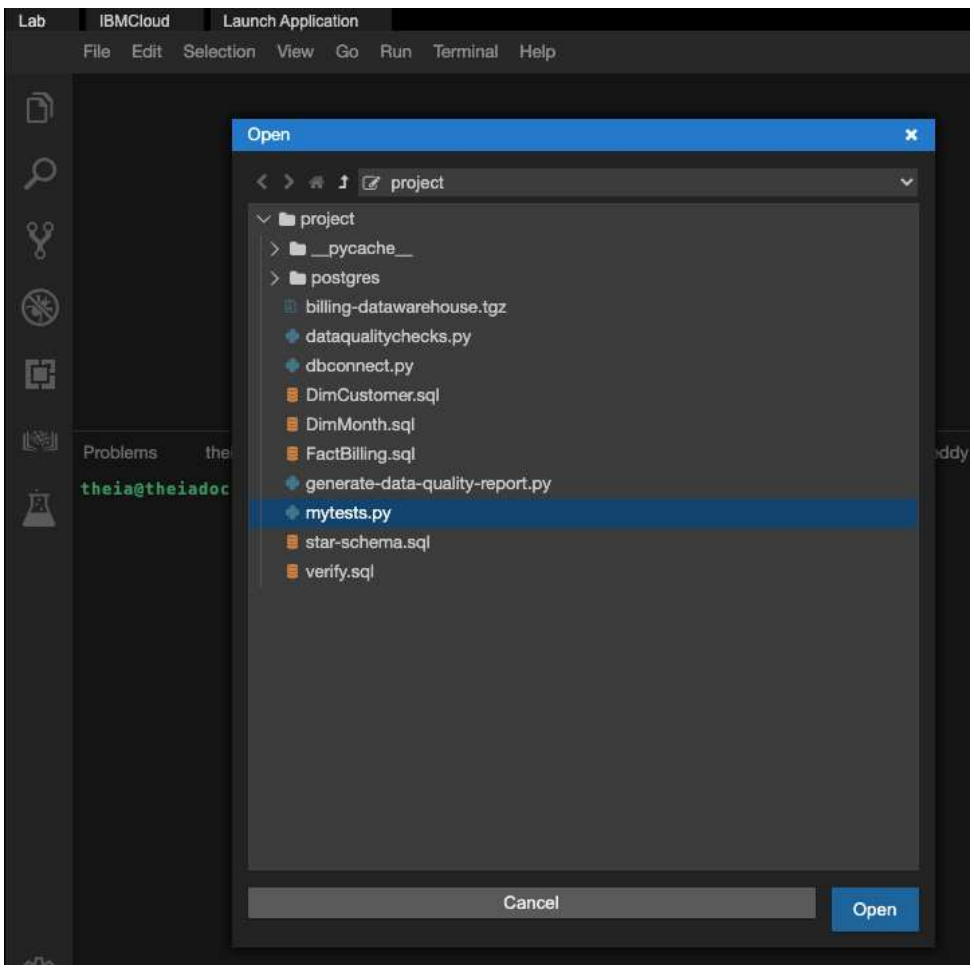
Copied! Executed!

You should see a list of tests that were run and their status.

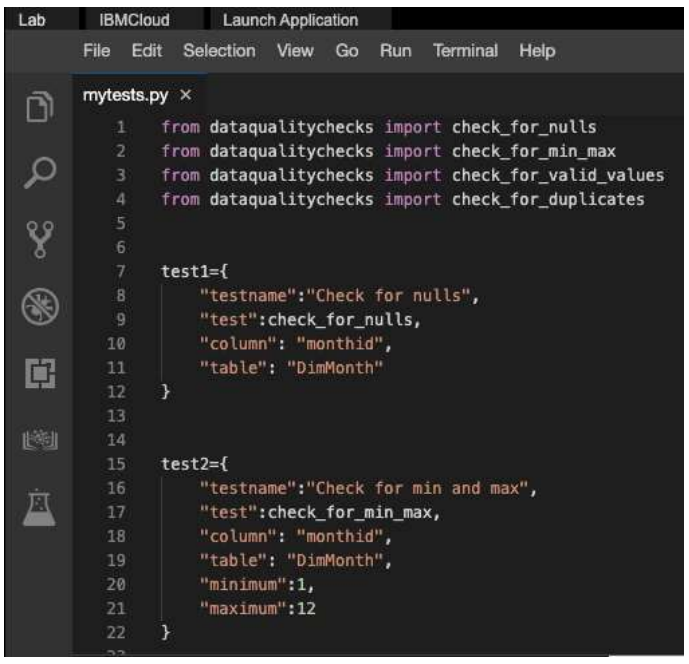
Exercise 4 - Explore the data quality tests

Open the file `mytests.py` in the editor by using the steps below.





You should now see the file opened in the editor.



The file `mytests.py` contains all the data quality tests.

It provides a quick and easy way to author and run new data quality tests.

The testing framework provides the following tests:

- check_for_nulls - this test will check for nulls in a column
- check_for_min_max - this test will check if the values in a column are with a range of min and max values
- check_for_valid_values - this test will check for any invalid values in a column
- check_for_duplicates - this test will check for duplicates in a column

Each test can be authored by mentioning a minimum of 4 parameters.

- testname - The human readable name of the test for reporting purposes
- test - The actual test name that the testing micro framework provides
- table - The table name on which the test is to be performed
- column - The table name on which the test is to be performed

Exercise 5 - Check for nulls

Let us now see what a `check_for_nulls` test looks like.

Here is a sample `check_for_nulls` test:

```
1
2
3
4
5
6
```

```
test1={
    "testname":"Check for nulls",
    "test":check_for_nulls,
    "column": "monthid",
    "table": "DimMonth"
}
```

Copied!

All tests must be named as `test` following by a unique number to identify the test.

- Give an easy to understand description for `testname`
- mention `check_for_nulls` for `test`
- mention the column name on which you wish to check for nulls
- mention the table name where this column exists

Let us now create a new `check_for_nulls` test and run it.

The test below checks if there are any null values in the column `year` in the table `DimMonth`.

The test fails if nulls exist.

Copy and paste the code below at the end of mytests.py file.

```
1
2
3
4
5
6
```

```
test5={
    "testname":"Check for nulls",
    "test":check_for_nulls,
    "column": "year",
    "table": "DimMonth"
}
```

Copied!

Save the file using Menu -> File -> Save

Run the command below to generate the new data quality report.

```
1
```

```
python3 generate-data-quality-report.py
```

Copied! Executed!

Exercise 6 - Check for min max range

Let us now see what a `check_for_min_max` test looks like.

Here is a sample `check_for_min_max` test

```
1
2
3
4
5
6
7
8
```

```
test2={
    "testname":"Check for min and max",
    "test":check_for_min_max,
    "column": "monthid",
    "table": "DimMonth",
    "minimum":1,
    "maximum":12
}
```

Copied!

In addition to the usual fields, you have two more fields here.

- minimum is the lowest valid value for this column. (Example 1 in case of month number)
- maximum is the highest valid value for this column. (Example 12 in case of month number)

Let us now create a new `check_for_min_max` test and run it.

The test below checks for minimum of 1 and maximum of 4 in the column `quarter` in the table `DimMonth`.

The test fails if there any values less than minimum or more than maximum.

Copy and paste the code below at the end of mytests.py file.

```
1
2
3
4
5
6
7
8
```

```
test6={
    "testname":"Check for min and max",
    "test":check_for_min_max,
    "column": "quarter",
    "table": "DimMonth",
    "minimum":1,
    "maximum":4
}
```

Copied!

Save the file using Menu -> File -> Save

Run the command below to generate the new data quality report.

```
1
```

```
python3 generate-data-quality-report.py
```

Copied! Executed!

Exercise 7 - Check for any invalid entries

Let us now see what a `check_for_valid_values` test looks like.

Here is a sample `check_for_valid_values` test:

```
1
2
3
4
5
6
7
```

```
test3={
    "testname":"Check for valid values",
    "test":check_for_valid_values,
```

```
4.     "column": "category",
5.     "table": "DimCustomer",
6.     "valid_values":{'Individual','Company'}
7. }
```

Copied!

In addition to the usual fields, you have an additional field here.

- use the field `valid_values` to mention what are the valid values for this column.

Let us now create a new `check_for_valid_values` test and run it.

The test below checks for valid values in the column `quartername` in the table `DimMonth`.

The valid values are Q1,Q2,Q3,Q4

The test fails if there any values less than minimum or more than maximum.

Copy and paste the code below at the end of mytests.py file.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1. test7={
2.     "testname":"Check for valid values",
3.     "test":check_for_valid_values,
4.     "column": "quartername",
5.     "table": "DimMonth",
6.     "valid_values":{'Q1','Q2','Q3','Q4'}
7. }
```

Copied!

Save the file using Menu -> File -> Save

Run the command below to generate the new data quality report.

```
1. 1
```

```
1. python3 generate-data-quality-report.py
```

Copied! Executed!

Exercise 8 - Check for duplicate entries

Let us now see what a `check_for_duplicates` test looks like.

Here is a sample `check_for_duplicates` test

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. test4={
2.     "testname":"Check for duplicates",
3.     "test":check_for_duplicates,
4.     "column": "monthid",
5.     "table": "DimMonth"
6. }
```

Copied!

Let us now create a new `check_for_duplicates` test and run it.

The test below checks for any duplicate values in the column `customerid` in the table `DimCustomer`.

The test fails if duplicates exist.

Copy and paste the code below at the end of mytests.py file.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. test8={
2.     "testname":"Check for duplicates",
3.     "test":check_for_duplicates,
4.     "column": "customerid",
5.     "table": "DimCustomer"
6. }
```

Copied!

Save the file using Menu -> File -> Save

Run the command below to generate the new data quality report.

```
1. 1
```

```
1. python3 generate-data-quality-report.py
```

Copied! Executed!

Practice exercises

1. Problem:

   Create a `check_for_nulls` test on column `billedamount` in the table `FactBilling`

▼ Click here for Hint

Use the check_for_nulls test with column=billedamount and table=FactBilling

▼ Click here for Solution

Copy and paste the code below at the end of mytests.py file.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. test9={
2.     "testname":"Check for nulls",
3.     "test":check_for_nulls,
4.     "column": "billedamount",
5.     "table": "FactBilling"
6. }
```

Copied!

2. Problem:

   Create a `check_for_duplicates` test on column `billid` in the table `FactBilling`

▶ Click here for Hint
▼ Click here for Solution

Copy and paste the code below at the end of mytests.py file.

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
6. 6
```

```
1. test10={
2.     "testname":"Check for duplicates",
3.     "test":check_for_duplicates,
4.     "column": "billid",
5.     "table": "FactBilling"
6. }
```

Copied!

3. Problem:

   Create a `check_for_valid_values` test on column `quarter` in the table `DimMonth`. The valid values are 1, 2, 3, 4

▶ Click here for Hint
▼ Click here for Solution

Copy and paste the code below at the end of mytests.py file.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1. test11={
2.     "testname":"Check for valid values",
3.     "test":check_for_valid_values,
4.     "column": "quarter",
5.     "table": "DimMonth",
6.     "valid_values":{1,2,3,4}
7. }
```

Copied!

Congratulations!! You have successfully finished this lab.

Authors

Ramesh Sannareddy

Other Contributors

Rav Ahuja

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2023-05-04	0.3	Vladislav Boyko	Updated code blocks to respective formats
2022-08-03	0.2	Lakshmi Holla	included pandas installation
2021-09-22	0.1	Ramesh Sannareddy	Created initial version of the lab