Data Quality technical documentation

Abstract

Case study aims to build a POC that will show usage of one of the most popular packages Soda SQL for ensuring high data quality in a wide range of storages.

Problem definition

A lot of companies from a wide range of industries who collect data in databases, cloud storages, data warehouses face data quality problems. Process of profiling data on its flow is very challenging, since there are no tools that will meet all business requirements simultaneously. Implemented case study shows how open source tools and libraries could be used to build data quality analysis for ETL jobs. First of all, such solutions should be flexible in choosing data storages that should be profiled and have a comprehensive set of metrics to apply on top of data. Additionally, profiling after data transformations and aggregations will allow us to track metrics during data flow and reduce the time for researching broken values. Soda SQL - ideal solution for addressing mentioned use cases. It utilizes user-defined input to prepare SQL queries that run tests on dataset in a data source to find invalid, missing, or unexpected data. When tests fail, they surface the data that you defined as “bad” in the tests. Armed with this information, data engineering team can diagnose where the “bad” data entered your data pipeline and take steps to prioritize and resolve issues.
For more information how Soda SQL works please refer to the official description: https://docs.soda.io/soda-sql/concepts.html
Soda SQL supports a wide range of storages including PostgreSQL, Snowflake, Amazon Redshift, GCP Big Query, MySQL, even Apache Spark dataframes and much more where you can scan your data.

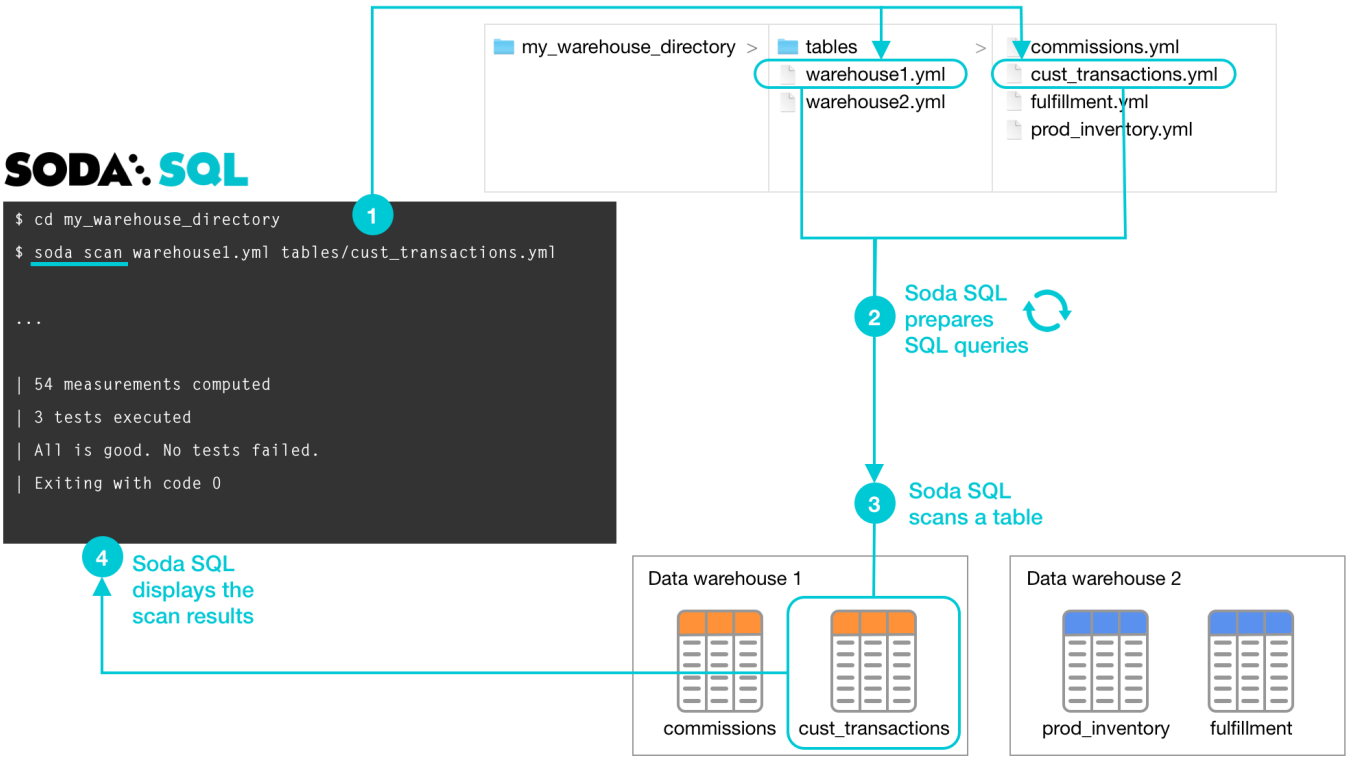
Soda SQL description

How Soda SQL works

After installing Soda SQL, you have to run the ***soda create warehouse_type*** command to set up data warehouse and *env_vars* YAML files, add login credentials to the *env_vars* YAML, then run ***soda analyze*** command. As a result, YAML files in the

/tables directory will be created that map to datasets in the data storage. After everything is done you are ready to scan.

The following image illustrates what Soda SQL does when scan is initiated from CLI.



Soda check could be easily integrated into orchestrated tools like Airflow, Dagster, or dbt Core™, to automate and schedule search for “bad” data using Python code. This is a more flexible way than running commands from CLI because it allows us to have more control over test results.
For example, Airflow provides PythonOperator and BashOperator for running Soda SQL logic.

Example of a python function with Soda SQL logic is shown below.

```
def run_soda_scan(warehouse_yml_file, scan_yml_file):
    from sodasql.scan.scan_builder import ScanBuilder
    scan_builder = ScanBuilder()
    # Optionally you can directly build the warehouse dict from Airflow secrets/variables
    # and set scan_builder.warehouse_dict with values.
    scan_builder.warehouse_yml_file = warehouse_yml_file
    scan_builder.scan_yml_file = scan_yml_file
    scan = scan_builder.build()
    scan_result = scan.execute()
    if scan_result.has_test_failures():
        failures = scan_result.get_test_failures_count()
        raise ValueError(f"Soda Scan found {failures} errors in your data!")
```

Tests definition

A test is a check that Soda SQL performs when it scans a dataset in a data source. Technically, it is a Python expression that checks metrics to see if they match the parameters defined for a measurement. A single Soda SQL scan runs against a single dataset in the data source, but each scan can run multiple tests against multiple columns.

Soda tests are defined in a scan YAML file which is associated with a specific dataset in the data source. Writing tests is possible using a built-in set of metrics that Soda SQL applies to an entire dataset, built-in column metrics that Soda SQL applies to individual columns or using custom metrics (also known as SQL metrics) that apply to an entire dataset or to individual columns.

Regardless of where it applies, each test is generally comprised of three parts:

- metric - property of the data in your data source
- comparison operator
- value

However, sometimes tests can have a fourth element to check whether data is valid. Validness is defined by column configuration key and expected format.

For example, the user defined the valid_format as date_eu or dd/mm/yyyy format. The metric invalid_percentage refers to the valid_format configuration key to determine if the data in the column is valid.

Full-structured YAML file with examples of usage of all mentioned features is shown below.

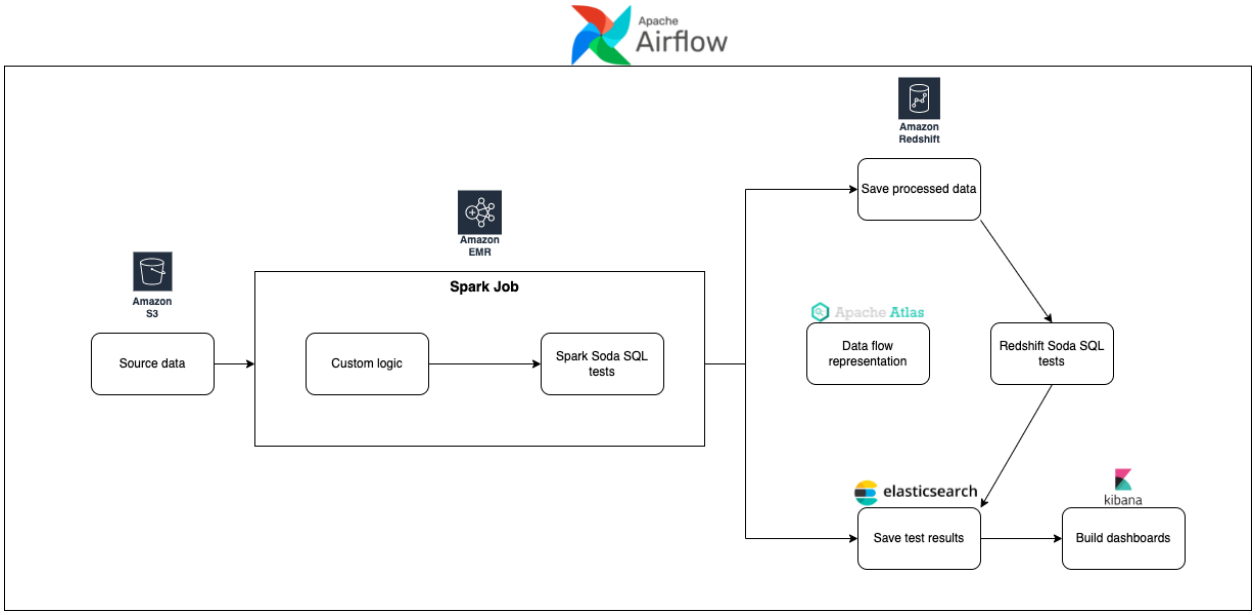
```
table_name: table_name
metrics:
  - row_count
  - missing_count
  - missing_percentage

# Validates that a table has rows
tests:
  - row_count > 0

# Tests that numbers in the column are entered in a valid format as whole numbers
columns:
  incident_number:
    valid_format: number_whole
    tests:
      - invalid_percentage == 0

# Tests that no values in the column are missing
  school_year:
    tests:
      - missing_count == 0

# Tests for duplicates in a column
  bus_no:
    tests:
      - duplicate_count == 0

# Compares row count between datasets
sql_metric:
  sql: |
    SELECT COUNT(*) as other_row_count
    FROM other_table
  tests:
    - row_count == other_row_count
```

For more information about possible metrics please refer to the official page https://docs.soda.io/soda-sql/examples-by-metric.html

Case Study implementation

Architecture

Below architecture shows how Soda SQL could be integrated into data pipelines that include a bunch of transformations and aggregations using Apache Spark and saving data into one of the popular DW Redshift with further representation test results using ELK stack and lineage of data flow using Apache Atlas.



Soda SQL scans can be applied anywhere in the ETL pipeline for measuring data quality and reducing time for analyzing places where data was broken.

Test results representation

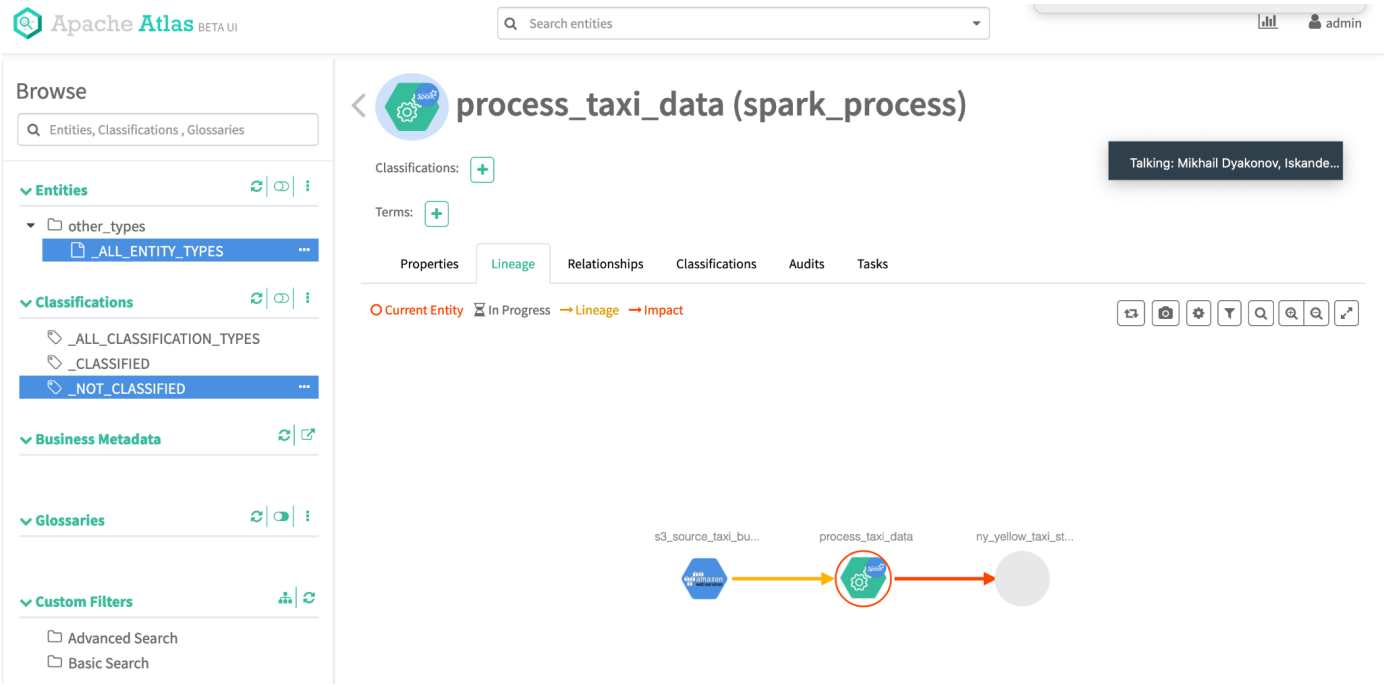
Collecting and representing test results is crucial for understanding scanned data, checking scan results and tracking possible problems over the time. One of the systems that could help to address mentioned problems is Elasticsearch for storing data and Kibana for graphical representation.
On the screenshot below is shown how test results look in table representation.

Time	columnName	execution_date	expression	passed	values.expression_result	dag_id	task_id
Jan 12, 2022 @ 16:49:27.128	vendorid	Jan 12, 2022 @ 16:49:27.128	invalid_percentage == 0	true	0.0	dq_case_study_dag	spark_job
Jan 12, 2022 @ 16:49:27.128	vendorid	Jan 12, 2022 @ 16:49:27.128	uniqueness == 0	true	0.0	dq_case_study_dag	spark_job
Jan 12, 2022 @ 16:49:27.128	min_passenger_count	Jan 12, 2022 @ 16:49:27.128	min >= 0	true	0	dq_case_study_dag	spark_job
Jan 12, 2022 @ 16:49:27.128	max_trip_distance	Jan 12, 2022 @ 16:49:27.128	max < 500	false	831.8	dq_case_study_dag	spark_job
Jan 12, 2022 @ 16:49:27.128	vendorid	Jan 12, 2022 @ 16:49:27.128	uniqueness == 0	true	0	dq_case_study_dag	run_soda_tests_in_redshift
Jan 12, 2022 @ 16:49:27.128	vendorid	Jan 12, 2022 @ 16:49:27.128	invalid_percentage == 0	true	0	dq_case_study_dag	run_soda_tests_in_redshift
Jan 12, 2022 @ 16:49:27.128	min_trip_distance	Jan 12, 2022 @ 16:49:27.128	min >= 0	true	0.0	dq_case_study_dag	spark_job
Jan 12, 2022 @ 16:49:27.128	min_passenger_count	Jan 12, 2022 @ 16:49:27.128	min >= 0	true	0	dq_case_study_dag	run_soda_tests_in_redshift
Jan 12, 2022 @ 16:49:27.128	max_passenger_count	Jan 12, 2022 @ 16:49:27.128	max < 10	true	7	dq_case_study_dag	run_soda_tests_in_redshift

Every row contains information about column name, execution date of specific DagRun and test result. This information allows track tests over the time and build dashboards.

Data Lineage

Data lineage uncovers the life cycle of data - it aims to show the complete data flow, from start to finish. Data lineage is the process of understanding, recording, and visualizing data as it flows from data sources to consumption. This includes all transformations the data underwent along the way, how the data was transformed, what changed, and why. Visualization of the data flow will make the process transparent and more understandable.
Screenshot below shows how data lineage looks like for an implemented case study.



Graph shows where data is stored, what kind of transformations are applied, where results are saved.