Ingestion Speedup & Refactoring

Proposal for C4GT 2023 | Organization : cQube

Shashwat Mahajan | Date: 7th June 2023

SYNOPSIS

CQube is a comprehensive, ready-to-use, and customizable solution for data processing and policy implementation in education and other fields. Although existing proof-of-concepts (POCs) have laid the groundwork, significant improvements are needed in the data ingestion process, as well as optimizing the use of hardware resources. Alongside this, the codebase requires refactoring and restructuring for improved maintainability, and a robust testing suite needs to be developed to ensure the reliability of the system.

Data is a transformative asset in today's world. Its importance lies in enabling informed decision-making, driving business growth and innovation, improving efficiency, fostering scientific research, shaping public policy, enhancing customer experiences, managing risks, and empowering individuals. As data continues to grow exponentially, its significance will only continue to increase across various sectors and domains.

Data Ingestion is the process of collecting, importing, and storing data from various sources for analysis and utilization. In today's world, where data has become the real wealth, data ingestion plays a crucial role in enabling organizations to extract valuable insights, make informed decisions, and gain a competitive edge. Data Ingestion plays a crucial role in CQube by collecting, processing, and storing various educational data for analysis and decision-making.

Thus, the cQube project aims to become a versatile and powerful platform for data ingestion and observability, enabling effective policy implementation across various sectors and involving diverse stakeholders.

Keywords: cQube, Data, Data Ingestion, POCs

TECH-STACK

Programming Language: TypeScript

Backend Framework: NestJS

Database: Hasura

Testing Framework: Jest

The cQube project utilizes a tech stack that includes TypeScript, NestJS, Hasura, and Jest. Here's a breakdown of the tech stack components:

TypeScript: TypeScript is a statically typed superset of JavaScript that provides enhanced developer productivity and tooling. It brings type safety and improved code organization to the project.

NestJS: NestJS is a progressive Node.js framework for building efficient and scalable server-side applications. It follows the modular design pattern and provides features like dependency injection, middleware, and decorators for building robust and maintainable APIs.

Hasura: Hasura is an open-source engine that connects to your databases and provides a real-time GraphQL API. It allows for efficient data retrieval and manipulation, making it suitable for managing and querying data in the cQube project.

Jest: Jest is a JavaScript testing framework with a focus on simplicity and speed. It provides an

intuitive API for writing unit tests, as well as tools for creating end-to-end (e2e) tests. Jest will be used to write and execute both unit tests and e2e tests for the cQube project.

The combination of TypeScript, NestJS, Hasura, and Jest provides a solid foundation for building scalable, maintainable, and tested applications. This tech stack allows for efficient development, seamless integration with databases, and comprehensive testing to ensure the reliability and quality of the cQube project.

PROJECT OVERVIEW

cQube is a ready-to-use, pre-packaged, and configurable Data Policy Governance (DPG) solution aimed at enabling effective policy implementation in education and other sectors.

The purpose of this project is to improve cQube by optimizing the data ingestion process for improved speed and hardware resource utilization, restructuring the code base for better maintainability, and developing a comprehensive testing suite. The project will be implemented using Typescript, NestJS, and Hasura .

TASKS WITH THEIR PROPOSED SOLUTION

OpenTelemetry #88

Description

Tracking runtimes of every single operation that is happening using open telemetry.

Problem Overview

In Cqube many operations like fetching data, processing information, and executing specific functions are running in unison. To ensure the system performs optimally, it is important to track and measure the runtime or the time taken by each of these operations. To address this issue, cQube has integrated OpenTelemetry, which is a set of tools and libraries used for observability in software systems.

Proposed Solution and Working Plan

Integrating the OpenTelemetry library into the cQube codebase to enable runtime tracking of operations.

Identifying the critical operations such as Data fetching, processing, Data Ingestion, database queries, or any other significant tasks that impact system performance.

We?ll need the @opentelemetry/api package for the core OpenTelemetry API, and the @opentelemetry/node for working with Node.js.

```
npm install @opentelemetry/api
npm install @opentelemetry/node
```

Make use of Tracing API by OT. The Tracing API consist of these main classes:

TracerProvider is the entry point of the API. It provides access to Tracers.

Tracer is the class responsible for creating Spans.

Span is the API to trace an operation.

Span will be used in our project. A Span represents a single operation within a trace. It encapsulates a start timestamp & an end timestamp which can be used to track runtimes.

References from web implementing this:

https://opentelemetry.io/docs/specs/otel/trace/api/#span

https://github.com/open-telemetry/opentelemetry-specification/blob/main/specification/metrics/semantic_conventions/runtime-environment-metrics.md

Test Cases #79

Description

Creating a Master ticket to track the test case coverage and availability for different services.

Problem Overview

The problem at hand is to track test case coverage and availability for different services in a master ticket. The ticket is specifically focused on E2E (end-to-end) tests and unit tests for various components related to data ingestion, parser services, dataset grammar, and dataset services. So we need to contribute a test case for any function that would increase test coverage.

Proposed Solution and Working Plan

To address the problem of tracking test case coverage and availability for different services, I propose implementing a comprehensive solution that includes the following components:

Determine the metrics or criteria for measuring test case coverage. This could include the number of test cases executed, the percentage of coverage achieved, and specific test scenarios or functionalities that should be covered.

Design a master ticket template that captures the necessary information for each service/component. Include fields such as service/component name, test case status (e.g., not started, in progress, passed, failed), coverage metrics, and any additional relevant information.

Create a new master ticket for each service/component and fill in the template fields with the relevant information. Start by entering the service/component name and assign a unique identifier to each ticket.

Regularly update the master ticket to reflect the status of each test case. As the testing progresses, mark the status of each test case accordingly (e.g., not started, in progress, passed, failed). Make test cases for every function to increase test coverage.

Resources

https://medium.com/slalom-build/increasing-unit-test-coverage-with-jest-it-up-108fa5c79157
https://www.lambdatest.com/learning-hub/end-to-end-testing

Better .temp file management #77

Description

Effective implementation of .temp file by managing locks on files and cleanup during data ingestion.

Problem overview

The lack of proper file management and synchronization leads to errors and disrupts the smooth operation of the system. There are two main issues to address in this problem:

File Cleaning during Data Ingestion for Event Data:

The current approach performs file cleaning while the data ingestion process is ongoing, which introduces the risk of conflicts and errors.

File Lock Management:

Currently , the system lacks a mechanism to manage locks on files effectively. In scenarios where multiple processes or components need to access the same file, there is no synchronization mechanism in place.

Proposed Solution and Working plan

To effectively implement the management of locks on files and cleanup during data ingestion, we can follow these detailed steps, including the creation of a ".temp" file:

Choosing a suitable file lock management mechanism based on our technology stack and system requirements.

TypeScript/JavaScript provides various synchronization primitives that can be used to manage file locks, such as mutexes, semaphores, or condition variables.

These synchronization primitives can be used to coordinate access to shared resources, including

files, across multiple processes or components.

Libraries like async-lock or semaphore provide abstractions for synchronization primitives in TypeScript and can be utilized for file lock management.

These primitives help ensure exclusive access to files and prevent conflicts between concurrent processes or components..

Before accessing a file, attempt to acquire a lock for that file using the chosen file lock management mechanism. If the lock cannot be acquired, wait or handle the situation based on your specific requirements (e.g., retry after a certain period or log an error).

Once the lock is acquired, perform the necessary file operations such as reading, writing, or modifying the data.
After completing the file operations, release the acquired lock to allow other processes or components to access the file.

Implementing error handling and appropriate fallback mechanisms to handle exceptional situations where file locks cannot be acquired or released successfully. Also ,determining the specific scenarios or operations where temporary data needs to be stored during the data ingestion process.

Creating a ".temp" file with a unique name and the ".temp" file extension to serve as a placeholder for temporary data or work in progress.We?ll ensure that the ".temp" file is created in a designated temporary directory or location, separate from the main data files and will be used to store the temporary data generated or processed during the data ingestion process.

Updating the file management and synchronization logic to include handling the ".temp" file

appropriately, such as acquiring and releasing locks on the ".temp" file.

Testing the creation and usage of the ".temp" file, including acquiring and releasing locks on the ".temp" file, to ensure proper functionality.
Executing automated tests using Jest to validate the correctness and reliability of the implemented file management, lock management, and cleanup features.

Event Upsert #75

Description

Introducing new API endpoints exclusively for the Event Upsert operations to update data multiple times a day without any data loss .

Problem overview

The current approach for event upsert, which involves replacing the existing event data file with a new file, has a significant drawback. If new data is to be uploaded multiple times in a day, the old data will be overridden, resulting in data loss. This poses a challenge as there is a need to update event data while preserving previously uploaded data.

To address this issue, a proposed approach is to introduce a new API specifically for event upsert operations and utilize the existing API for uploading new data.

Proposed Solution and Working plan

New API for Event Upsert:

Creating a new API endpoint, PUT /ingestion/event, dedicated to handling event upsert operations. This API will allow us to update event data without overwriting the entire file.

```
// Defining a route for the event upsert API endpoint
app.put('/ingestion/event', (req, res) =>

// Getting the event data from the request body
const eventData = req.body;
```

Event Upsert Operation and validation :

When using the event upsert API, we need to provide the necessary data payload for the specific event needed to be updated. The system needs to identify the corresponding event in the existing event data file and update only the relevant fields or values, rather than replacing the entire file.

```
// Validate the incoming data against the event schema
const isValid = validateEventData(eventDataToUpdate, eventSchema);
if (!isValid) {
res.status(400).json({ message: 'Invalid event data' });
return;
}
```

Existing API for New Data Upload:

We?ll retain the existing API (/ingestion/new_programs) for uploading new event data and will be used to upload completely new data, distinct from the previously uploaded data multiple times in a

day.

Error Handling:

At the end , implementing proper error handling and validation mechanisms within the new API for event upsert operations. We need to ensure that error messages are clear and informative, guiding how to correct any data inconsistencies or conflicts during the update process.

```
if (existingEventIndex !== -1) {

// Update the relevant fields in the existing event
const existingEvent = eventData[existingEventIndex];
Object.assign(existingEvent, eventDataToUpdate);

// Save the updated event data
// Code to persist the eventData array back to your data store

res.status(200).json({ message: 'Event updated successfully' });
else {
res.status(404).json({ message: 'Event not found' });
```

References from web implementing this:

https://sugarclub.sugarcrm.com/dev-club/b/dev-blog/posts/upsert-explained-by-upsert

https://experienceleague.adobe.com/docs/experience-platform/catalog/datasets/enable-upsert.html?lang=en

Dimension Upsert #74

Description

Redesigning the dimension upsert to increase efficiency .

Problem overview

The current approach for dimension upsert, which updates the dimension data in the database without updating the corresponding CSV file, creates a discrepancy between the database and the CSV file and leads to data loss if the YARN CLI ingest process is rerun, as the old data will be overwritten .

Proposed Solution and Working Plan

Update the API to accept a CSV file as input, where the name of the CSV file should match the dimension name. Ensure that the API validates all the required columns for the target table.

Implement validation logic to check for errors in the columns of the CSV file.

During ingestion, if there are errors in specific records, segregate those records and store them in an error file. Create a ingestion_error directory and upload the error file to this directory, enabling further analysis and resolution.

Perform an upsert operation on the target table, ensuring that new records are inserted and existing records are updated based on a conflict resolution strategy (e.g., primary key or unique constraint).

After the upsert operation, retrieve all the records from the target table to obtain the complete and updated dataset. Create a new CSV file using the retrieved records from the target table. Upload the newly created CSV file to the processed_input -> dimensions folder.

```
// Performing dimension upsert operation
const { dimensionName } = req.body;
const errorRecords = [ ];
const updatedRecords = upsertDimensionRecords(dimensionName,
                                    file.path);
```

Link to get a brief idea about Dimension Upsert:
https://www.sqlservercentral.com/articles/upsert-dimension-table

Persist configs for CSV Ingestion #73

Description

To develop a service to allow for saving config , function to figure out the last config that ran and delta between the current and the last config. and call out uncommitted changes like Prisma migration does. (Commit like a git and save hash) .

Problem overview

The current system lacks the capability to persist CSV ingestion configurations, making it difficult to track and manage the configurations used for data ingestion. There is a need for a service that allows for saving and managing these configurations, including the ability to determine the last

executed configuration, identify changes between configurations, and provide visibility into uncommitted changes.

Proposed Approach and Working plan

Configuration Persistence Service:

Developing a dedicated service to handle the storage and management of CSV ingestion configurations. The CSVIngestionConfig interface represents the structure of a configuration object.

```
const app = express();
app.use(express.json());

interface CSVIngestionConfig {
  id: string;
  name: string;
  filePath: string;
  delimiter: string;
  hasHeader: boolean;
  lastExecuted: string | null;
  uncommittedChanges: boolean;
}
let configurations: CSVIngestionConfig[ ] = [ ];
```

Saving Configurations:

By implementing an API endpoint (POST /configurations) users will be allowed to save CSV ingestion configurations which should include relevant details such as file paths, column mappings

etc.

```
// API to save a configuration
app.post('/configurations', (req, res) => {
  const newConfig: CSVIngestionConfig = req.body;
  configurations.push(newConfig);
  res.status(201).json({ message: 'Configuration saved successfully' });
});
```

Last Executed Configuration:

We?ll create a function that retrieves the information of the last executed configuration. This function can query the persistence service or maintain a separate metadata record indicating the last executed configuration.

Delta Calculation:

Developing a function to determine the deltas between the current configuration and the last executed configuration , identify changes in file paths, column mappings, transformations, or other parameters .

Uncommitted Changes:

Implementing a mechanism similar to Git commits to track and manage configuration changes. Associate a unique identifier or hash with each configuration, and when a configuration is modified or updated, it will prompt the user to commit the changes.

```
// API to track uncommitted changes
app.get('/configurations/uncommitted', (req, res) => {
    const uncommittedConfigs = configurations.filter(config =>
```

```
                                    config.uncommittedChanges);
    res.status(200).json(uncommittedConfigs);
    });
```

Error Handling and Logging:

Implement error handling mechanisms and proper logging to capture any issues or inconsistencies during configuration saving, retrieval, or comparison. Log relevant information for troubleshooting and analysis.

Allow for update in config. #72

Description

Allow for an update in config and CLI to include --update-only flag to figure out the delta and only make those changes to dimensions, datasets, and event definitions. Same with datasets.

Problem overview

The current system lacks the capability to update configurations and selectively apply changes to dimensions, datasets, and event definitions. Additionally, the CLI tool does not provide an option to identify the delta changes and apply them selectively. There is a need to allow for configuration updates and introduce a --update-only flag in the CLI tool to handle delta changes efficiently.

To address this problem, a proposed solution is to enhance the system by enabling configuration

updates and introducing a --update-only flag in the CLI tool. This will allow users to identify the delta changes between configurations and apply those changes selectively to dimensions, datasets, and event definitions.

Proposed Approach and Working Plan

Configuration Update:

Enhancing the configuration persistence service to allow for updating configurations by implementing an API endpoint, PUT /configurations/{config_id}, to enable users to modify existing configurations.

Delta Calculation:

Developing a function that calculates the delta changes between the updated configuration and the last executed configuration and compares the two configurations and identify the specific changes made to dimensions .

--update-only Flag in CLI:

Extend the existing CLI tool with a --update-only flag. This flag will instruct the CLI tool to analyze the delta changes between the current and last executed configurations and apply only those changes to the respective dimensions, datasets, and event definitions.

```
const args = yargs.option('update-only', {
description: 'Apply only the delta changes to dimensions, datasets, and
            events',
type: 'boolean',
}).argv;
```

```
const updateOnly: boolean = args['update-only'];


if (updateOnly) {
const deltaChanges: DeltaChangescalculate= DeltaChanges(updated
                                                Config, lastExecutedConfig);
applySelectiveUpdates(deltaChanges);
} else {
// Perform full updates
}
```

Selective Updates:

Implementing the logic in the CLI tool to read the delta changes and make selective updates to the dimensions, datasets, and event definitions based on the identified changes.

```
 function applySelectiveUpdates(deltaChanges: DeltaChanges) {
    // Update dimensions
    deltaChanges.dimensions.forEach(dimensionChange => {
      const dimensionIndex = dimensions.findIndex(dimension =>
      dimension.dimensionId === dimensionChange.dimensionId);
      if (dimensionIndex !== -1) {
       dimensionChange.changes.forEach(fieldChange => {
       dimensions[dimensionIndex][fieldChange.fieldName] =
       fieldChange.newValue;  });
  });
```

```
// Update datasets (similar logic as dimensions)

// Update events (similar logic as dimensions)
  }
```

Testing and Validation:

Performing thorough testing to validate the configuration update functionality, delta calculation, and selective updates using the CLI tool.

Define errors and add them to the output folder. #71

Description

To identify, process, remove and store the errors, logs, and other required information in the output folder before actually ingesting the data. Additionally these errors should be documented as an error registry as well.

Problem overview

In the current system, there is a need to enhance the error handling mechanism during the data ingestion process. The configuration file should allow for defining an output location where errors, logs, and other relevant information can be stored. The task is to identify different types of errors that may occur during data ingestion, such as foreign key violations, and implement a process to handle, remove, and store these errors in the output folder. Additionally, there should be an error registry to document and track these errors for future reference.

Proposed Approach and Working Plan

Error Types Identification:

Analyzing the data ingestion process and identifying various types of errors that can occur. Some examples include ?

```
enum ErrorType {
  ForeignKeyViolation = 'ForeignKeyViolation',
  DataFormatInconsistency = 'DataFormatInconsistency',
  MissingRequiredFields = 'MissingRequiredFields',
  DataValidationFailure = 'DataValidationFailure'
}
```

Error Handling and Removal:

When an error occurs during data ingestion, the system will capture and process the error accordingly. Removing the erroneous data from further processing to prevent it from affecting the rest of the ingestion process.

```
// Example usage
handleDataIngestionError(ErrorType.MissingRequiredFields, record);
```

Output Folder Configuration:

Updating the configuration file to include an output location parameter. This can be done by adding a field to your existing configuration object:

```
const config: IngestionConfig = {
```

```
  // Existing configuration values
  outputFolder: '/path/to/output/folder'
};
```

Error Storage:

Developing a mechanism to store the identified errors in the output folder by creating error-specific files or directories to organize the errors based on their types.

Error Registry:

Implementing an error registry to document and track the errors encountered during the data ingestion process. This registry should capture details about each error, including its type, timestamp, affected records, and any relevant context information.

Error Reporting and Documentation:

Generating error reports or summaries based on the stored errors.

```
function generateErrorReport() {
// Generate error report based on the stored errors in the errorRegistry
}
```

Expose Ingestion Progress

Description

To update the CLI to add a new progression mapper that allows for sharing the progress of

ingestion for every file and every row.

Problem overview

In the current system, there is a need to provide visibility and share the progress of the data ingestion process. Additionally, the CLI tool lacks a progression mapper that allows for tracking the progress of ingestion for every file and every row. There is a requirement to enhance the system by exposing the ingestion progress and updating the CLI tool to include a new progression mapper that accurately reflects the progress at various stages.

Proposed approach and Working plan

Progression Mapping:

Defining a progression mapping mechanism that tracks the progress of data ingestion at different stages, including :

```
private async insertRow(row: Row): Promise<void> {
// Perform database insertion
}
private async validateFile(file: File): Promise<void> {
// Perform file validation
}
private async transformData(file: File): Promise<Row[]> {
// Perform data transformation
}
```

Implementing within the Ingestion Process:

Updating the ingestion process to incorporate the progression mapping mechanism. This ensures that progress is tracked and updated as each file and row is processed.

Expose Ingestion Progress:

Status endpoint or a real-time progress update is accessed during the ingestion process. The progress will indicate the overall status, as well as the progress of individual files and rows.

```
// Expose ingestion progress through an API endpoint
app.get('/progress', (req, res) => {
const progress = ingestionService.getIngestionProgress();
res.json(progress);
});
```

CLI Tool Enhancement:

Updating the command-line interface to display the current progress of data ingestion, including the progress for each file and row , ensuring that the CLI output provides a clear and informative representation of the progress.

Error Handling and Reporting:

If any errors occur during the ingestion process, accurately reflect them in the progress updates and CLI output.

Testing and Validation:

We?ll be performing comprehensive testing through Jest to validate the progression mapping, progress reporting, and CLI enhancements and to verify that the progress accurately reflects the

status of data ingestion .

Manage OOM Errors when ingesting large files > 1 GB

Description

To handling large files by ?

1.Setting up a limit to memory

2.Add a benchmark for polars for the cap on maximum memory per deployment.

Problem overview

In the current system, there is a need to address the issue of Out-of-Memory (OOM) errors when ingesting large files, specifically those exceeding 1 GB in size. To tackle this problem, a solution is required to manage memory consumption during the ingestion process, prevent OOM errors, and ensure efficient processing of large files. A similar issue :

https://github.com/pocketbase/pocketbase/issues/836

Proposed approach and Working plan

Setting up Memory Limit:

Implementing a memory limit configuration to control the maximum amount of memory allocated during the ingestion process and setting an appropriate memory threshold based on system requirements.

```
// Configuring memory limit
const MAX_MEMORY_LIMIT = 2 * 1024 * 1024 * 1024;  // 2GB (adjustable)
```

Memory Benchmarking for Polars:

Evaluating the memory consumption of the Polars library, which is used for data processing and ingestion. Also , conducting benchmarking tests to measure memory usage for different file sizes and use the benchmark results to determine the optimal memory limit per deployment.

```
// Perform memory benchmarking for different file sizes and operations
    const fileSize1GB = 1024 * 1024 * 1024;
    benchmarkMemoryUsage(fileSize1GB, 'Data Processing');
```

Memory Optimization Techniques:

Implementing data chunking (memory optimization technique) to reduce the memory footprint during the ingestion process.

File Chunking and Parallel Processing:

Implementing the file chunking techniques to break down large files into smaller, manageable chunks for processing And enabling parallel processing of these file chunks to leverage the available system resources effectively.

```
function processFileChunks(file: File, chunkSize: number):
Promise<void> {
    const totalChunks = Math.ceil(file.size / chunkSize);
    return new Promise((resolve, reject) => {
    const fileReader = new FileReader();
```

```
    fileReader.onload = async (event) => {
    const chunkData = event.target.result
 // Process the chunked data
 }}
}
```

Error Handling and Retry Mechanism:

When an OOM error occurs, gracefully handling the error, log relevant information, and possibly retrying the ingestion process with adjusted parameters or optimization techniques.

Documentation and Communication:

Updating the system documentation to include information about memory management techniques and limitations.

Make docker-compose.yml parametric with env variables.

Description

Currently the docker-compose.yaml exposes secrets for a DB. Task is to remove that and move those to an env file.

Problem overview

In the current setup, the docker-compose.yml file contains hard-coded secrets for the database

configuration. This poses a security risk as sensitive information is exposed in the file. Additionally, the file lacks flexibility to be used in different environments. To address these issues, the solution is to parametrize the docker-compose.yml file using environment variables and securely store the DB secrets in an environment file.

Proposed approach and Working plan :

Identifying the configuration values in the docker-compose.yml file that need to be made parametric and determine which values should be sourced from environment variables .

Creating an Environment File:

Creating a separate environment file (.env file) to store the DB secrets and other sensitive configuration values and storing the necessary environment variables in this file using the key-value format.

Replacing Secrets in docker-compose.yml:

Removing the hard-coded secrets from the docker-compose.yml file and replacing the secrets with environment variable placeholders that will be sourced from the environment file. Sample is shown below?

```
version: '3'
services:
app:
image: myapp:latest
ports:
- ${APP_PORT}:3000
environment:
```

-DB_HOST=${DB_HOST}

-DB_PORT=${DB_PORT}

-DB_USERNAME=${DB_USERNAME}

-DB_PASSWORD=${DB_PASSWORD}

Parameterize the Configuration:

Updating the docker-compose.yml file to reference the environment variables instead of hard-coded values. Also replacing the fixed values with the corresponding environment variable placeholders throughout the file.

Loading Environment Variables:

Configuring the environment variables that are needed to be loaded from the environment file when running the docker-compose command. We can use dotenv or specify the environment file directly when running the docker-compose command.

```
import * as dotenv from 'dotenv';
dotenv.config({ path: '.env' });
```

Override global config by program config if available.

Description

In the config.json, all program configs should override the global configs.

Problem overview

In the current system, there is a requirement to allow program-specific configurations to override global configurations. The configuration settings are stored in a config.json file, and it is necessary to ensure that program configurations take precedence over global configurations when both are present. This will provide flexibility in customizing settings at the program level while still maintaining a global configuration framework.

Proposed approach and Working plan

Reviewing structure of the config.json:

Reviewing the structure of the config.json file to ensure it accommodates both global and program-specific configurations and defining a clear hierarchy or structure that allows for easy identification and overriding of configurations.

Global Configuration:

Identifying the global configuration settings in the config.json file and defining default values for global configurations that will be used when program-specific configurations are not available.

```
function overrideConfigWithProgramConfig(globalConfigFile: string,
                programConfigFile: string): void {
// Read the global configuration
const globalConfig = JSON.parse(fs.readFileSync(globalConfigFile, 'utf8'));
```

Program Configuration:

Determining the structure and format of program-specific configurations within the config.json file. We?ll create a separate section or object for each program configuration, allowing for individual program customization.

```
// Read the program-specific configuration
const programConfig = JSON.parse(fs.readFileSync(programConfigFile, 'utf8'));
```

Loading and Merging Configurations:

If a program-specific configuration is available, override the corresponding global configuration settings.

```
// Merge program-specific configuration with global configuration
Object.assign(globalConfig, programConfig);
```

Handling Missing Program Configurations:

Accounting for scenarios where a program-specific configuration is missing in the config.json file, we?ll implement appropriate fallback mechanisms to ensure the system can handle missing program configurations gracefully.

Implement - quoteChar

Description

Task is to add a config param called quoteChar and ensure when ingesting data for datasets or dimensions, it allows defining the CSV quote character. Currently, it's a single quotes implementation.

Problem Overview

In the current implementation of cQube's data ingestion process, there is a need to introduce a new configuration parameter called "quoteChar" in the config.json file. This parameter will allow users to specify the CSV quote character to be used when ingesting data for datasets or dimensions. The existing implementation uses single quotes as the default quote character .

Proposed approach and Working plan

Updating the config.json Structure:

Modifying the structure of the config.json file to include the new "quoteChar" configuration parameter under the dimensions .

```
 interface Config {
        dimensions: {
          quoteChar: ?\? ;
          // Other dimension configurations...
          };
        // Other configurations...
    }
```

Validating Quote Character:

Validating that the quote character provided is a valid character used in CSV files.

Modify CSV Reader Implementation:

Updating the CSV reader code to read the "quoteChar" configuration parameter from the config.json

file and replacing the hard coded single quote with the configured value of the "quoteChar" parameter.

Implement Configuration Loading:

Writing a line of code to load the config.json file and parse its contents into memory, including the new "quoteChar" configuration parameter.

Making a test case to make sure our Implementation works:

Writing a test case file to ingest a sample data and apply it using config.json to implement and test the case.

My Implementation

I have implemented this feature as per my understanding and filed a PR against it https://github.com/ChakshuGautam/cQube-ingestion/pull/128. These are some referencing screenshots:

Implement - csvDelimeter

Description

cQube allows for specifying what data to be ingested using a config.json, a sample for which can be found here.

Add a config param called csvDelimeter and ensure when ingesting data for datasets or dimensions,

it allows defining the csvDelimeter. Currently, it's a comma.

Problem Overview

In the current implementation of cQube's data ingestion process, there is a need to introduce a new configuration parameter called "csvDelimiter" in the config.json file. This parameter will allow users to specify the CSV delimiter character to be used when ingesting data for datasets or dimensions. The existing implementation uses a comma as the default delimiter, but this needs to be made configurable to support different delimiter characters based on user requirements.

Proposed approach and Working plan

Updating the config.json Structure:
Modifying the structure of the config.json file to include the new "csvDelimiter" configuration parameter.

Validating Delimiter:
Implementing validation logic to ensure that the specified delimiter is valid character used in CSV files and conforms to the expected format.

Modifying CSV Reader Implementation:
Updating the CSV reader code to read the "csvDelimiter" configuration parameter from the config.json file and replacing the hardcoded comma with the configured value of the "csvDelimiter" parameter.

Implementing Configuration Loading:
Developing code to load the config.json file and extract the "csvDelimiter" parameter.

Use Configured Delimiter:

Modifying the data ingestion process for datasets and dimensions to utilize the configured delimiter by replacing the hardcoded comma with the value of the "csvDelimiter" configuration parameter when parsing CSV files.

Implement - caseSenstiveFKSearch

Description

cQube allows for specifying what data to be ingested using a config.json, a sample for which can be found here. .Task is to add a config param called caseSenstiveFKSearch and ensure when ingesting data for datasets, it allows for case insensitivity. Currently, it would ignore data when an FK doesn't match.

Problem Overview

In the current implementation of cQube's data ingestion process, there is a need to introduce a new configuration parameter called "caseSensitiveFKSearch" in the config.json file. This parameter will control the case sensitivity behavior when ingesting data for datasets that involve foreign key (FK) constraints. Currently, if the FK values do not match case sensitively, the data is ignored.

Proposed approach and Working plan

Updating the config.json Structure:

Modify the structure of the config.json file to include the new "caseSensitiveFKSearch" configuration parameter.

Modifying Dataset Ingestion Logic:

Updating the code to read the "caseSensitiveFKSearch" configuration parameter from the config.json file.

Case Insensitive FK Search:

Implementing logic to handle case insensitivity when performing FK search during data ingestion for datasets and modifying the FK comparison process to consider case insensitivity for matching FK values.

Implement Configuration Loading:

Developing a code to load the config.json file and extract the value of the "caseSensitiveFKSearch" configuration parameter.

Test and Validation:

Developing test cases to validate the behavior of the data ingestion process with different values of the "caseSensitiveFKSearch" parameter and executing them to ensure that the case insensitivity functionality is working as expected.

Implement - onlyCreateWhitelisted

Description

cQube allows for specifying what data to be ingested using a config.json, a sample for which can be found here. Add a new config param called onlyCreateWhitelisted and ensure that only the datasets with matching names are created when ingesting schema.

Problem Overview

In the current implementation of cQube's data ingestion process, there is a need to introduce a new configuration parameter called "onlyCreateWhitelisted" in the config.json file. This parameter will control the behavior of schema creation during the data ingestion process. When enabled, only the datasets with names that match the whitelist will be created, while others will be ignored.

Proposed approach and Working Plan

Updating the config.json Structure:

Modifying the structure of the config.json file to include the new "onlyCreateWhitelisted" configuration parameter.

Modifying Schema Creation Logic:

Updating the code to read the "onlyCreateWhitelisted" configuration parameter from the config.json file.

Implementing Whitelisted Dataset Creation:

Developing logic to check if a dataset's name matches any entry in the whitelist defined in the configuration and modifying the schema creation process to create only the datasets that match the whitelist.

Implementing Configuration Loading:

Developing code to load the config.json file and extract the value of the "onlyCreateWhitelisted" configuration parameter.

Test and Validation:

Developing test cases to validate the behavior of the data ingestion process with different values of the "onlyCreateWhitelisted" parameter and execute them to ensure that only the whitelisted datasets are created when the parameter is enabled.

Timeline

June 22nd - June 30th (Before the Coding Period)

Denotes the time after the submission of proposals and before the commencement of the coding period .

Interact with mentors and review timeline and milestones further and decide upon them.

Get to know and understand project deliverables better, decide upon regular meetings, and have a great one-on-one interaction while asking for feedback.

July 1st - July 8th (Week 1)

Review the project repository and its documentation to gain an overview of the codebase structure and organization.

Taking up two or three tasks per week to implement as a milestone .

Gaining knowledge about configuration parameters and implementing the QuoteChar #65.

Implementation of CSVDelimiter #64 and caseSenstiveFKSearch #63 based on prior knowledge and skills .

June 9th - July 15th (Week 2)

Implementing onlyCreateWhitelisted #62 as a config. parameter .

Explore the current .temp file management system utilized by cQube during the data ingestion process.

Optimizing .temp file management #77 through locks and other requirements

Starting up with implementing Open-Telemetry #88 within ingestion processes .

July 16th - July 29th (Week 3 & Week 4)

Concluding up with Open-Telemetry #88 with testing and validation .

Taking up the test cases implementation #79 and building the master ticket from scratch .

Implementing the modified approach for Event Upsert #75 through API.

Implementing the modified approach for Dimension Upsert #74 through the same approach as #75 .

July 30th- August 12th (Week 5 & Week 6)

Enhancing readability by implementing services under Persist configs for CSV Ingestion #73 .

Implementing Allow for update in config. #72 to make the system capable of updating configurations.

Updating CLI to implement Expose Ingestion Progress #69 that allows for sharing the progress of ingestion .

Making docker-compose.yml parametric with env variables #67 to hide crucial information from DB .

August 13th- August 26th (Week 7 & Week 8)

Gaining insights about the Error handling techniques under the cQube DAG.

Defining the errors and adding them to the output folder #71 and creating an error registry .

Managing OOM Errors when ingesting large files > 1 GB #68 by setting up a memory limit .

Overriding global config by program config (if available) #66 to provide flexibility

August 27th- August 30th (End of Program)

Discuss with mentors about the project and gain insights on performance and get feedback for improvements.

Discuss the future scope of the project and how to expand it further to make Obsrv practices even better.

Future Development

Upon successful completion of this project, the enhanced cQube solution will serve as a robust and performant DPG solution for effective policy implementation. Further development can focus on additional features, scalability improvements, integration with external systems, and enhancing the user interface.

Integration with Additional Data Sources: Currently, the project focuses on optimizing the data ingestion process for CSV files. However, there is potential for expanding the scope to include other data sources, such as JSON, Excel, or Databases. This would involve extending the data ingestion capabilities and providing support for various data formats.

Advanced Data Processing Techniques: The project can explore advanced data processing techniques to further enhance the system's performance and efficiency. This may include implementing data streaming mechanisms, distributed processing frameworks, or leveraging cloud-based services for scalable data ingestion and processing.

Real-time Data Streaming: Consider incorporating real-time data streaming capabilities into the data ingestion pipeline. This allows for immediate analysis and visualization of streaming data, enabling quicker decision-making and providing real-time insights into educational data.

This list is by no means exhaustive in nature.Development of this project shall by no means be confined to the coding period of this program and shall go on with respect to the spirit of open source.

Availability

Q. When do your classes and exams finish?

My exams for the 2nd year just got over and hence I have no upcoming exams for the next 3 months. My classes for the 3rd year shall commence in offline mode from 17th July onwards.

They can generally extend up to 2 pm at max, but that won?t be an issue as I generally get up early in the morning and that?s when I do most of my coding-related activities.

Managing classes along with the program and one on one mentor interactions would not be a problem.

Q. Do you have a full- or part-time job or internship planned for this summer?

No. I do not have any full or part-time job or internship planned for this entire summer and hence would be available for the project.

Q. How many hours per week do you have available for a summer project?

I will be able to consistently put in an average of 60 hrs per week for the summer project.

I am passionate about this project as well as the opportunities which it opens up for me. I would like to see this through till the end and contribute to this wonderful project.

Personal Background

About Me and My Motivation to Apply in cQube

My name is Shashwat Mahajan , and I'm on a thrilling coding adventure! Currently an undergraduate at the prestigious National Institute of Technology, Jalandhar, pursuing my passion for Computer

Science . As I wrap up my second year of engineering, I can't help but feel a surge of excitement for what lies ahead.

Coding and exploring new tech stacks are like fuel to my soul. The thrill of learning something novel and diving into uncharted territories is what keeps me going. It's the reason I chose this field as my major, and now I'm determined to transform it into a full-fledged profession.

My repertoire of tech stacks is as diverse as the colors of a rainbow. From Full Stack Web Development to the intricacies of DevOps with Docker and Kubernetes, and even delving into the world of Machine Learning , I've dipped my toes into various domains. But what truly ignites my passion is working with startups and open-source organizations. The challenge of building things from scratch and the constant opportunity to learn something new is what drives me.

Participating in C4GT during my summer break has been a revelation. It has been a whirlwind of exploration and growth, where I've had the chance to work with awe-inspiring organizations and expand my knowledge base. In just a month, I've had the privilege of immersing myself in new technologies and gaining valuable insights that fuel my coding journey. My time working with the cQube mentors has been wonderful and I have learnt so much in this brief time which excites me further for what is to come.

Every step I take in this incredible world of coding motivates me to become an exceptional developer. I'm eagerly awaiting the next chapter of my journey, where I can make a meaningful impact and contribute to the ever-evolving landscape of technology.

So, here's to the thrill of coding, the joy of learning, and the endless possibilities that lie ahead. Let's embrace this adventure together and create something remarkable.

LinkedIn: https://www.linkedin.com/in/shashwat-mahajan-40374b278/

Github: https://github.com/shashwatm1111

Email: shashwatm.cs.21@nitj.ac.in , shshwtmhjn@gmail.com

Resume:

https://drive.google.com/file/d/1bzq18oy1sY3yYDkVM45eS5W8fBpvFasA/view?usp=sharing

Contact Details

Open Source Experience

I am a budding technologist, eager to explore the vast potential of open source. I would consider myself fairly new to open source.I have learnt some tech stacks and contributed here and there. I have also made contributions to the GirlScript Summer of Code 2023 program . As part of C4GT, I discovered some great organizations and I have tried to do my part in contributing.

Why Me?

I sincerely believe that I can pull off these projects and work on whatever issues I face. I thrive in collaborative environments, embracing diverse ideas and perspectives to drive innovation. With my strong work ethic, attention to detail, and commitment to excellence, I am confident that I can contribute effectively to the success of the project. Moreover, my participation in the C4GT program has exposed me to a wide range of technologies and has honed my skills in tackling complex

problems. I am eager to apply my knowledge and skills to make a positive impact and contribute meaningfully to the team's goals. By selecting me, you would gain a dedicated and enthusiastic team member who is eager to learn, grow, and contribute to the success of the project.

Thank You Note

Thank you for giving me the time to read my proposal. I would also like to thank each and every member of the cQube team for their prompt replies of our doubts and teaching us a lot in this time.