PROJECT / RELEASE

Project Design Document

G1 - NGA

Janko Leskovac < jl8592@rit.edu>

Josip Pavlinic <jp3447@rit.edu>

Kristijan Nincevic <[kn1059@rit.edu](mailto:kn1059@rit.edu)>

Domagoj Dusic <dd8614@rit.edu>

Roko Hafner <rh9922@rit.edu>

Niko Galic Alviz < ng5850@rit.edu>

# Project Summary

The Schema2Script project is a Java-based application designed to convert database schemas from various formats (such as JSON and XML) into SQL scripts compatible with different database management systems (DBMS) like MySQL and Oracle. This tool provides a streamlined approach for users who need to work with schemas in multiple formats by offering a unified interface to handle schema parsing, structure analysis, and SQL generation. By implementing the MVC (Model-View-Controller) and Factory design patterns, the project is built to support modularity, making it easy to add new input formats or target databases with minimal code changes. The Schema2Script application allows users to upload a schema file, automatically detect its type, and generate a fully functional SQL script, simplifying what can otherwise be a complex and time-consuming task.

# Design Overview

The design of the Schema2Script project emphasizes core principles like separation of concerns, high cohesion, low coupling, and support for extendibility. From the beginning, the team focused on structuring the project to ensure that each component would serve a specific purpose, aligning well with the MVC pattern. This allowed the application to be divided into distinct modules: SchemaView for the user interface, SchemaController for handling business logic, and SchemaModel for data representation. This separation ensures that changes in one module, such as the view layout, won’t directly impact the data handling logic or parsing process. To support flexibility, we implemented the Factory pattern in ParserFactory and GeneratorFactory to dynamically select the appropriate parser or SQL generator based on the input file format and target DBMS.

One of the early design decisions that did not go as anticipated was attempting to combine parsing and SQL generation logic into a single component. This approach quickly led to issues with coupling and limited our ability to extend the project. Refactoring to separate the responsibilities of parsing and SQL generation helped address these concerns, improving the application’s extendibility. Additionally, we considered using a single parser for all file types but ultimately chose to implement specific parsers like JsonParser and XMLParser to handle format-specific nuances more effectively. Documenting these decisions, especially the "rejected alternatives," provides valuable insights for future developers who may extend the project with additional parsers or generators.

# Overall System Structure

A computer screen shot of a diagram

Description automatically generated

This class diagram illustrates the structure of the Schema2Script project, detailing how different components work together to parse schema files and generate SQL scripts. The main class, SchemaRunner, initializes the application and coordinates with SchemaController, which acts as the core logic handler, connecting SchemaModel (data structure) and SchemaView (user interface). When a schema file is uploaded, SchemaController uses ParserFactory to determine the appropriate parser (XMLParser or JsonParser) based on the file type. This parser reads the file and creates a SchemaModel representing tables, columns, and relationships. To generate SQL, SchemaController employs GeneratorFactory, selecting either SqlGenerator or OracleGenerator to produce database-specific scripts based on the schema data. Finally, the generated SQL script is displayed in SchemaView, allowing users to interact with the output. This setup promotes modularity and scalability, enabling easy expansion for new file formats or database types.

# 

# Sequence Diagrams

## **Sequence Diagram 1 (what feature / operation / scenario the diagram shows).**

A diagram of a schematic

Description automatically generated

In this sequence diagram, the MVC pattern is applied, starting with the view when the display() method is called to open the UI. Then, handleFileUpload() is triggered to initiate file selection, and fileChooser() opens to allow the user to choose a file. Once selected, the file is passed to the controller. The controller then uses the setFile(file) method to store the file in the model, followed by a confirmFileSet() callback to verify the file is successfully set.

## **Sequence Diagram 2 (what feature / operation / scenario the diagram shows).**

Same as above.

A diagram of a software project

Description automatically generated with medium confidence

In this sequence diagram, the MVC and Factory patterns work together to convert a file into an SQL script. The process begins in the controller after the user selects a file, triggering handleFileUpload() in the view. The file is passed to the model via the controller's get(file) method. The model then calls ParserFactory to create the appropriate parser (e.g., JsonParser), which processes the file using parse(file) and returns a SchemaModel. Next, the model uses GeneratorFactory to select the correct generator (e.g., SqlGenerator). The generator’s generate(SchemaModel) method produces the SQL script as a string, which is sent back through the controller and displayed in the view.