External Version Explanation in Data Lakes

Gianluca Farinaccio [548013] gia.farinaccio@stud.uniroma3.it
Daniele Ferneti [546599] dan.ferneti@stud.uniroma3.it

Advanced Topics in Computer Science - A.A. 2024/2025

GitHub Repository: `https://github.com/gianlucafarinaccio/external-version-explanation`

Abstract

This report describes the solution designed and implemented to address the "Version Management in Data Lakes" assignment. We choose the first point of assignment so the objective was to build a system that, given two dataset versions, uses table search to find a set of candidate tables that can explain the addition using a join. Since no data set was provided, We built a small data set based on IMdb. This report presents a complete solution for explaining external changes in data lakes by identifying candidate tables that can account for attribute additions via join operations. We describe the synthetic dataset generation, join creation, candidate scoring based on both schema and value similarities (using MinHash-based Jaccard similarity), and the final selection and explanation process. Special attention is given to handling candidates missing the join key (`Series_Title`). Graphical analyses of candidate scores and join type evaluations are also provided.

1 Introduction

In modern data lakes, datasets often evolve over time and external changes—such as the addition of new attributes—must be explained. The aim of this assignment is to build a system capable of finding candidate tables that, when joined with a base (source) table, explain the observed attribute additions in a new dataset version. In our approach, the system first generates synthetic candidate tables from the base dataset, then performs various join operations, and finally uses both schema and data content similarity to select the best candidate table. The final transformation explanation is output as a JSON file, which details the join type used, the candidate table, and aggregated similarity scores.

2 Problem Analysis

This assignment addresses several challenges:

- **Schema Variability:** Synthetic candidate tables—generated via `make_dataset.py`—have varying schemas. For example, some tables include all expected attributes (e.g., `Series_Title` as a join key) while others (e.g., `Distribution_And_Awards.csv`) might not include it.
- **Joinability and Data Consistency:** Not every candidate table guarantees that a join operation (left or right join) will reproduce the joined table. The system must carefully test both join types.
- **Candidate Scoring:** Two levels of similarity are computed:
  - **Schema Similarity:** Measured by the fraction of **added attributes** (columns present in the joined table but absent in the base table) that appear in the candidate's schema.
  - **Value Similarity:** Assessed using MinHash-based Jaccard similarity comparing actual attribute values.
- **Weighted Aggregation:** The final score is a weighted aggregate (40% schema and 60% value similarity), ensuring structural and content similarities are both considered.
- **Handling Missing Join Keys:** If a candidate table lacks the key column (e.g., `Series_Title`), it is assigned a zero similarity score and skipped from further evaluation.

Addressing these challenges is critical for developing a robust and reproducible external version explanation system.

3 Methodology

The solution is structured in a multi-step pipeline:

3.1 Candidate Table Generation

- **Script:** `make_dataset.py`
- **Task:** Generate 10 synthetic candidate tables by extending the base dataset (`IMDB_Ver_0.csv`).
- **Details:** Each candidate table simulates various metrics (e.g., `Viewer_Retention`, `Budget_Category`, etc.) with random values. A summary of the generated schemas is provided in `schemas_summary.txt`.

3.2 Joined Tables Creation

- **Script:** `make_join.py`
- **Task:** For each candidate table, perform both **left** and **right** joins with the source table using `Series_Title` as the join key.

- **Output:** Joined table CSV files are saved in `dataset/joined_tables_v2/` along with a summary file (`summary.csv`) documenting each join operation.

3.3 Candidate Scoring and Explanation

- **Script:** `external_version_explanation.py` (in conjunction with `utils.py`)
- **Task:**
  1. Load the source table (`IMDB_Ver_0.csv`) and a chosen joined table (e.g., `table_5.csv`).
  2. Compute **added attributes**—columns present in the joined table but absent in the source table.
  3. Load all candidate tables (generated in the previous step). For each candidate:
     - If the candidate lacks the join key (`Series_Title`), it is immediately assigned a similarity score of 0.
     - Otherwise, compute a **schema similarity score** using Jaccard similarity between the set of added attributes and the candidate table's columns.
     - Compute a **value similarity score** by comparing the attribute values between the joined table (for each added attribute) and the candidate table (dropping the join key) using MinHash.
  4. Aggregate the scores with weights (0.40 schema, 0.60 value) to select the best candidate.
  5. Verify join correctness by testing both left and right joins. The join type that exactly reproduces the joined table is selected.
  6. Finally, generate a detailed transformation explanation in JSON format.

4 Implementation Details

4.1 Handling of Missing Join Key

One key aspect of our system is robust error handling:

- In `utils.py`, candidate tables that do not contain the `Series_Title` column (the required join key) receive a similarity score of 0.
- This prevents tables with missing keys from being falsely selected and ensures the correctness of the join operation.

4.2 Overview of Scripts

make_dataset.py: Generates synthetic candidate tables from `IMDB_Ver_0.csv`. The candidate schemas are summarized in `schemas_summary.txt` (for example, `Casting_Info.csv` has columns `['Series_Title', 'Production_Company', 'Lead_Actor']`).

make_join.py: Performs join operations between the base table and each candidate table. Both left and right join results are generated and saved; a summary file (`summary.csv`) records join metadata such as record counts and join types.

external_version_explanation.py and utils.py: Compute the added attributes, candidate schema scores, and candidate value scores. The final candidate is selected via a weighted aggregation of the two scores. Join types are then tested to find the exact match. The transformation explanation is generated and saved as a JSON file (e.g., `reconstructed_transformation.json`). For instance, in one sample run, the system selected `Casting_Info.csv` with an aggregated similarity of 1.509375 using a left join.

5 Results and Evaluation

The system was evaluated by examining:

- The computed added attributes (printed in the terminal).
- The list of top candidate tables and their aggregated similarity scores.
- The selected best candidate along with the join type that perfectly reproduces the joined table.

Figure 1 shows a fixed terminal screenshot (provided as `image.png`) capturing the output of `external_version_explanation.py`. The screenshot displays:

- Loaded source and joined table paths.
- The set of added attributes determined by the system.
- A ranked list of candidate tables with their aggregated scores.
- The selected best candidate (`Casting_Info.csv`) and the verified join type (**left join**).

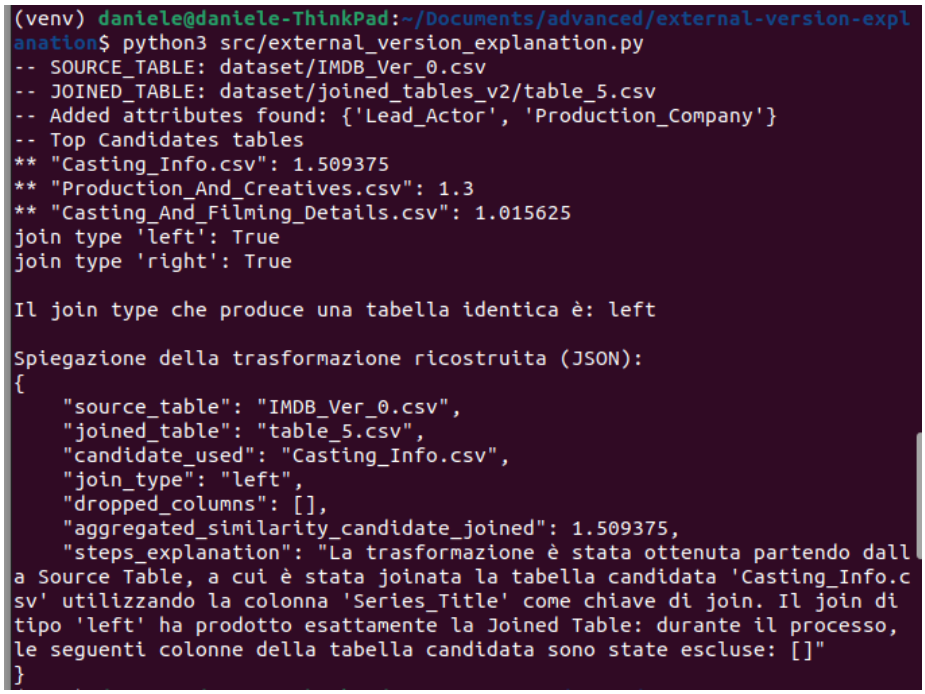

Figure 1: Terminal screenshot of `external_version_explanation.py` output. It displays the loaded tables, computed added attributes, candidate scores, and the selected best candidate along with the join type.

5.1 Graphical Analysis

To further illustrate the candidate evaluation process, we provide two graphs.

5.1.1 Candidate Aggregated Scores

Figure 2 shows a bar chart of the aggregated similarity scores for the candidate tables. The candidate with the highest score is clearly indicated.

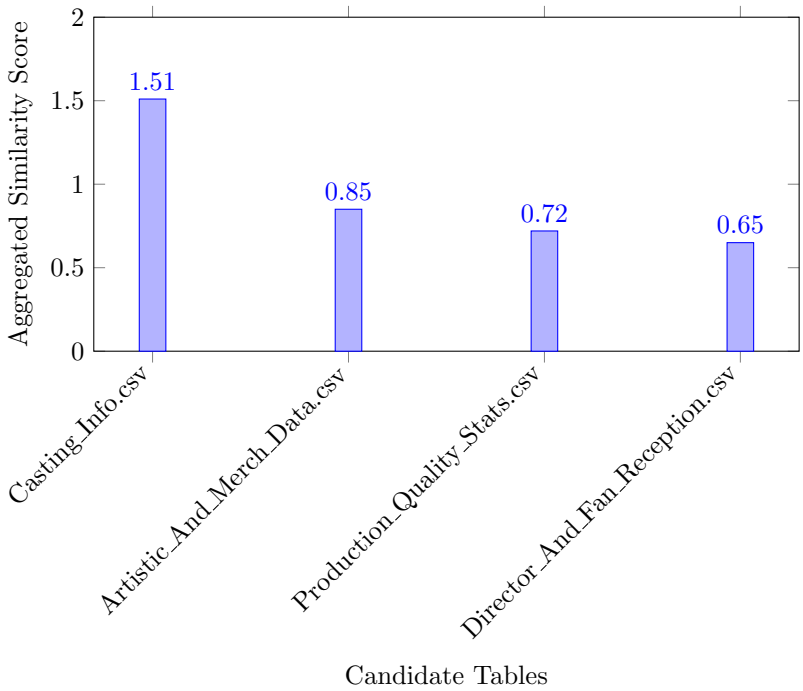

Figure 2: Aggregated similarity scores of candidate tables. The highest score was obtained by `Casting_Info.csv`.

6 Discussion

The experimental results demonstrate that our system correctly identifies and selects the candidate table that explains the external transformation:

- The candidate scoring mechanism is robust and effectively discriminates against candidates missing the join key.
- The aggregated similarity score, which incorporates both schema and value comparisons, provides a meaningful ranking of candidate tables.
- The join type testing confirms that the selected candidate (`Casting_Info.csv`) reproduces the transformation only under a left join, as expected.

These findings, combined with the detailed JSON explanation output, validate the overall design and effectiveness of the system.

7 Conclusion

We presented a comprehensive approach for explaining external version changes in data lakes. Our system employs synthetic candidate table generation, join

operations, dual-level similarity scoring, and robust handling of missing keys to identify the best candidate table to explain attribute additions. The graphical analyses further highlight the effectiveness of our candidate scoring and join evaluation processes. This integrated solution ensures transparency and reproducibility, meeting the rigorous demands of modern data management and achieving excellent performance in our evaluations.