From Tracking Lineage to Enhancing Data Quality and Auditing: Adding Provenance Support to Data Warehouses with `ProvETL`

Matheus Vieira[1], Thiago de Oliveira[2], Leandro Cicco[2], Daniel de Oliveira[1] and Marcos Bedo[1]
[1]*Institute of Computing, Fluminense Federal University, Brazil*
[2]*Information Technology Superintendence, Fluminense Federal University, Brazil*

Keywords: Data Warehousing, ETL, Provenance, Data Quality, Business Intelligence.

Abstract: Business intelligence processes running over Data Warehouses (BIDW) heavily rely on quality, structured data to support decision-making and prescriptive analytics. In this study, we discuss the coupling of *provenance* mechanisms into the BIDW Extract-Transform-Load (ETL) stage to provide lineage tracking and data auditing, which *(i)* enhances the debugging of data transformation and *(ii)* facilitates issuing data accountability reports and dashboards. These two features are particularly beneficial for BIDWs tailored to assist managers and counselors in Universities and other educational institutions, as systematic auditing processes and accountability delineation depend on data quality and tracking. To validate the usefulness of provenance in this domain, we introduce the **`ProvETL`** tool that extends a BIDW with provenance support, enabling the monitoring of user activities and data transformations, along with the compilation of an execution summary for each ETL task. Accordingly, **`ProvETL`** offers an additional BIDW analytical layer that allows visualizing data flows through provenance graphs. The exploration of such graphs provides details on data lineage and the execution of transformations, spanning from the insertion of input data into BIDW dimensional tables to the final BIDW fact tables. We showcased **`ProvETL`** capabilities in three real-world scenarios using a BIDW from our University: personnel admission, public information in paycheck reports, and staff dismissals. The results indicate that the solution has contributed to spotting poor-quality data in each evaluated scenario. **`ProvETL`** also promptly pinpointed the transformation summary, elapsed time, and the attending user for every data flow, keeping the provenance collection overhead within milliseconds.

1 INTRODUCTION

Business Intelligence (BI) processes have become commonplace for decision-making across various corporate activities and domains. These processes are typically integrated with Data Warehouses (BIDW) (Kimball and Ross, 2011) or even Data Lakes (Nargesian et al., 2023), especially when dealing with unstructured data. One domain in which BIDW is particularly useful is the strategic planning and management of Universities and other educational institutions (Rudniy, 2022). The challenges in this context include monitoring courses, student performances, maintenance and personnel costs, and research initiatives. Deans, managers, and board members can benefit from accurate reports and consistent data to make informed decisions on new projects and social policies.

Typically, required data to support these decisions (or estimate potential educational impacts) are structured and stored in relational databases through large-scale educational information systems. Additional data are gathered from government sources, press releases, non-governmental organizations, and *ad-hoc* spreadsheets. Such heterogeneous data must undergo conformation and consolidation into the Data Warehouse through Extract-Transform-Load (ETL) processes (Vassiliadis et al., 2002).

The context of this study revolves around an institutional BIDW of our University, encompassing more than five million tuples. The BIDW is a collection of six Data Marts, each following the star schema and associated with their respective ETL routines. The transformation processes are essential to conform and standardize data from heterogeneous sources, ensuring proper consumption and avoiding inconsistencies. ETL routines are specifically designed with the assumption data sources are *static*, *i.e.*, their schemas do not change over time. However, changes in the schema may be usual as data sources are hetero-

Vieira, M., de Oliveira, T., Cicco, L., de Oliveira, D. and Bedo, M.
From Tracking Lineage to Enhancing Data Quality and Auditing: Adding Provenance Support to Data Warehouses with ProvETL.
DOI: 10.5220/0012634500003690

In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 1*, pages 313-320
ISBN: 978-989-758-692-7; ISSN: 2184-4992

geneous, requiring the adaptation of existing routines, which can be error-prone (Rahm and Bernstein, 2006). Those errors may result in loading inconsistent data into the BIDW, which makes the identification of such inconsistencies a top priority.

Provenance (Freire et al., 2008; Herschel et al., 2017) presents a natural solution to support users in tracking the steps of a process, following the data derivation path, and monitoring relevant metadata and parameters associated with data transformations. Provenance data have been employed in workflows from various domains, such as healthcare (Corrigan et al., 2019). We observe ETL routines are similar to large-scale workflows as they are iterative processes that face the challenge of feedback loops and involve multiple data transformations, users, datasets, and parameters (Silva and others., 2021).

Therefore, in this study, we propose the integration of provenance capturing and storing mechanisms into the ETL routines of a BIDW. Specifically, we introduce a novel strategy for automatically *injecting* instructions into the ETL transformations to collect and store both prospective and retrospective provenance in a dedicated database. This database serves as a valuable source of information for monitoring, auditing, and correcting issues in the BIDW, thereby enhancing data quality. Our solution, named **ProvETL**, integrates a provenance schema with injection strategies to create a BIDW extension that addresses the following questions: *(i)* Which data output was produced after the ETL transformation consuming a given input? *(ii)* Which user executed this transformation with specific input data and setup? *(iii)* What is the summary for the execution of this transformation and input data (in terms of an aggregation function, such as `COUNT` and `AVG`)? *(iv)* What transformation sequence generates this data product?

ProvETL also provides a web-based analytical layer with a provenance graph of the executed ETL routine, allowing users to interactively explore answers to previous questions by traversing the graph. To showcase the capabilities of our proposed solution, we investigated three real-world scenarios related to personnel admission, public information in paycheck reports, and staff dismissals from our University BIDW. The results indicate that **ProvETL** was effective in *(i)* detecting poor-quality inputs and *(ii)* identifying transformations susceptible to noisy data with acceptable processing overhead.

The remainder of this paper is divided as follows. Section 2 discusses the background. Section 3 introduces the material and methods, while Section 4 presents the qualitative and quantitative results. Finally, Section 5 offers the conclusions.

2 BACKGROUND

Provenance Data. This term denotes metadata that explains the process of generating a specific piece of data. Thus, provenance records data derivation paths within a particular context for further querying and exploration, which fosters reproducibility in scientific experiments. We apply this concept as a rich source of information, encompassing the routines, parameter values, and data transformations executed by users during the ETL process, *i.e.,* a resource for monitoring and assuring the data quality of the BIDW. The W3C recommendation, PROV (Groth and Moreau, 2013), defines a data model for representing provenance data. It conceptualizes provenance in terms of *Entities*, *Agents*, *Activities*, and various relationship types. Here, an entity could represent a specific table of the OLTP database. *Activities* are actions performed within the ETL routines that affect entities, such as currency standardization, with execution times and error messages. Lastly, an *Agent* is a user responsible for executing ETL routines. The specification of an ETL pipeline can be viewed as *Prospective Provenance* (*p-prov*), a form of provenance data that logs the steps carried out during ETL routines. Another category of provenance is *Retrospective Provenance* (*r-prov*), which captures details related to the execution process, such as when transformations are executed.

Provenance Capturing. Capturing provenance is a process that can take two forms: *(i)* based on instrumentation or *(ii)* instrumentation-free. In the first category, the collection of provenance or the invocation of a provenance system, *e.g.*, YesWorkflow (McPhillips et al., 2015), requires injecting specific calls into the script or program where provenance needs to be captured. While this approach offers the advantage of capturing only the necessary provenance, it does require some effort from the users. The latter category captures every action performed in the script without requiring any modification in the source code. The approaches in this category, *e.g.*, noWorkflow (Murta et al., 2015), present the advantage of being transparent for the user, but it may result in collecting a substantial volume of data with a significant portion may be irrelevant for the user analysis, *e.g.*, registering a file opening.

Data Warehousing. The BIDW model encompasses four main layers. The first layer represents *External Data Sources*, containing all input data that may provide valuable insights. Data sources include relational databases (*i.e.*, OLTP databases), JSON

files, and spreadsheets. The second layer is the *Data Staging Layer*, which stores data already extracted from external data sources but not yet standardized, cleaned, and aggregated. It serves as a data preparation area, where ETL routines transform data before loading them into the BIDW. The ETL process can be conducted using off-the-shelf tools or in-house scripts. The *Data Warehouse Layer* contains consolidated BIDW data (modeled as either a star schema or a snowflake schema) ready for querying (Kimball and Ross, 2011). The last BIDW layer (*Presentation Layer*) contains the tools responsible for consuming data from the BIDW and supporting decision-making, typically through analytical dashboards.

Talend ETL Suite. In this study, we adopt the off-the-shelf ETL tool Talend Studio[1] following the already coded ETL routines from the real-world BIDW we examine in the experiments (our solution does not depend on the ETL tool). Talend provides a unified suite for ETL and data management that addresses data integration issues and provides data quality and sharing mechanisms. It enables users to implement complex ETL routines visually and to extend transformations based on their specific needs.

Related Work. Various provenance management solutions are available across different contexts, but most existing studies focus on supporting scientific experiments. Within the context of ETL routines, propose a provenance model designed for ETL routines as a vocabulary based on the deprecated Open Provenance Model (OPM) standard (Kwasnikowska et al., 2015). Despite its advantages, the approach relies on an outdated provenance standard since OPM was replaced by PROV in 2013. (Zheng et al., 2019) introduce a tool named PROVision, which aims to leverage fine-grained provenance to support ETL and matching computations by extracting content within data objects. While PROVision represents a step forward, it primarily focuses on fine-grained provenance. Additionally, it requires users to model the ETL routines using PROVVision, limiting the choice of tools. (Reis Jr et al., 2019) also propose a provenance architecture for open government data, storing metadata in a Neo4J graph database. Provenance is implemented within the UnBGOLD tool, which is designed for Linked Open Government Data and is unable to handle generic ETL routines.

In this study, we aim to bridge the gap in utilizing provenance data to enhance data quality within ETL routines in an agnostic manner, *i.e.*, without dependency on any specific ETL tool.

[1]https://www.talend.com/

3 MATERIALS AND METHODS

In this section, we introduce the **`ProvETL`** tool, which supports provenance data capturing and storing during the ETL stage of BIDW systems. The core idea of **`ProvETL`** involves intercepting data transformations performed by ETL routines by injecting provenance calls within the routines. The calls capture both r-prov and p-prov and store them in a relational database for analysis. The intercept-and-inject strategy takes place during the instrumentation of the ETL routine. Essentially, it involves adding a script to the routine to capture provenance data of interest, such as the user who started the ETL routine and the start/end time of each data transformation. **`ProvETL`** assumes that ETL routines can be implemented in various forms. Therefore, it relies on a generic provenance-capturing mechanism that avoids targeting a particular tool, such as Talend, Knime, and scripts in specific languages like Python.

Therefore, a **`ProvETL`** communication condition is to make HTTP requests within the ETL routine. Once provenance data have been captured, an HTTP request must be made, incorporating the collected data as the requested content (*i.e.*, body). The request is then intercepted by a **`ProvETL`** server, which stores the provenance data in a relational database. This message-passing approach involves an observable trade-off as it allows the capture of provenance from generic ETL routines while relying on a client-server application protocol, which is subject to bottlenecks on the server side and network throughput. To minimize performance issues, we addressed the following functional requirements.

Functional Requirements. The main functional requirements observed prior to the **`ProvETL`** construction are listed in Table 1.

Table 1: Functional requirements.

ID	Description
#1	Users may create traceable instances representing the dataflow
#2	Users may define the transformations that occur in the dataflow
#3	Users may define the input and output set schemas for the transformations
#4	The system must define endpoints to receive data from the dataflow transformations
#5	Users may visualize the dataflow on a graph
#6	Users may explicitly search provenance data

Requirement #1 involves providing an endpoint for creating user-defined dataflows. The user needs to specify a name and description for the dataflow,

which will be displayed later on the **ProvETL** web interface. Once the dataflow is created, the system should return an identifier for the subsequent instrumentation. Requirement #2 involves providing an endpoint to create the transformations that will be mapped to a dataflow. Requirement #3 ensures the specification of the schema for the input/output data transformations, including attribute names and their types. Both the input and output attributes return internal identifiers for subsequent instrumentation. Requirement #4 is analogous to an API definition, created internally by **ProvETL** during the execution of transformations to obtain data instances, and is not directly called by the user. Requirement #5 specifies the creation of a graphical interface for visualizing the provenance graph associated with the dataflow. Finally, Requirement #6 defines an entry point for querying collected data.

Data Model. **ProvETL** stores provenance data from ETL data transformations in a relational DBMS. However, any physical model can be used for data storage by mapping the conceptual model proposed in Figure 1, represented by a simplified Entity-Relationship Diagram. The central entity, *Dataflow*, models the dataflow and includes descriptive attributes. The second entity is *DataTransformation*, representing all data transformations. One instance of *Dataflow* may be associated with multiple instances of *DataTransformations*. The third entity is *DataSetSchema*, which represents the structure of transformed data, both at the input (before the transformation) and at the output (after the transformation). A *DataTransformation* can be related to multiple *DataSetSchemas*, and this relationship is categorized as *input/output*. A *DataSetSchema* includes a list of meta-attributes representing the dataflow attributes and their types. Finally, the last entity is the *DataSet*, representing the data files consumed/produced by data transformation, including the schema and values.

Architecture and Implementation. **ProvETL** was designed following a 3-layer pattern in a distributed architecture: *(i)* an Interface Layer, *(ii)* an Application Layer, and *(iii)* a Data Layer – Figure 2. The *Web Portal* in the *Interface Layer* allows users to interact with provenance data, listing the captured dataflows and the corresponding data transformations defined in the instrumentation stage. The transformations within a dataflow and its relationships are displayed as an expandable provenance graph built in the web interface with the Vis.js library. The REST API provides endpoints to receive collected provenance data from external ETL Clients, as previously defined in the data model (Figure 1). The *Application Layer* coordinates the **ProvETL** logic. This layer was implemented as a complete application developed in Java 8 using the Spring Boot framework version 2.7.14. The *backend* is subdivided into several modules, following a structure called Package by Layer: *(i)* the *Controller*, responsible for managing the endpoints, *(ii)* the *Service*, in charge of the enforcing logic rules, *(iii)* the *Repository*, responsible for accessing the data, and *(iv)* the *Entity*, responsible for defining the entities. Finally, the Data Layer contains the Provenance Database, which manages all collected provenance data in a persistent form.

ProvETL Setup. This stage involves minimal steps to implement the automatic intercept-and-inject strategy into ETL routines, which can be implemented in various ways, including scripts or specific ETL tools. The only requirement for integrating **ProvETL** into the existing ETL routines implemented in any ETL tool is the ability to establish an HTTP connection to make requests and receive confirmations from the **ProvETL** server-side. The first step for setting up **ProvETL** is the identification of the current dataflow (See Functional Requirement #1), which is persisted as one *Dataflow* entity by the system. After creating the entity following a user-submitted request, the endpoint provides a dataflow identifier. Thereafter, the transformations (jobs) must be defined, which is carried out by an HTTP request that also returns one identifier associated with the created transformation. Each data transformation definition includes the input and output datasets, which enables the instantiation of *DataSetSchema* entities. After obtaining the identifiers, we inject the provenance calls in the ETL routine, sending provenance data by making a series of HTTP requests, *i.e.*, r-prov data. An additional block of code is added for each data transformation to collect the current user executing the step in the ETL routine, the start and end time of the transformation, and the number of input and output tuples for the process. Then, another block of code is used to build the body of the HTTP request to the **ProvETL** server-side. Next, we present a code snippet to be added to a Python-based script that captures r-prov and considers the transformation summary as the counting of the involved tuples (data summarization is defined in terms of SQL aggregation functions, *e.g.*, `COUNT`, `SUM`, `MAX`, *etc.*). The endpoint has already been connected in the definition of the dataflow, *i.e.*, p-prov, so it can be directly invoked.

```
current_user = getpass.getuser()
current_time = datetime.now()
```

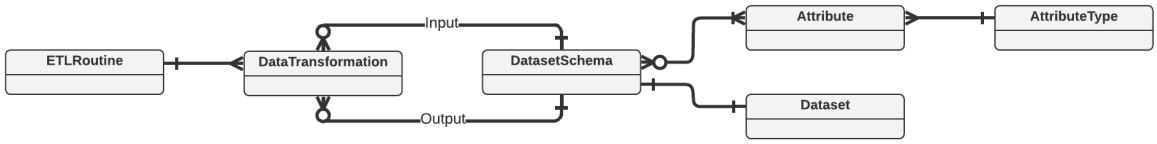

Figure 1: **ProvETL** Data Model.

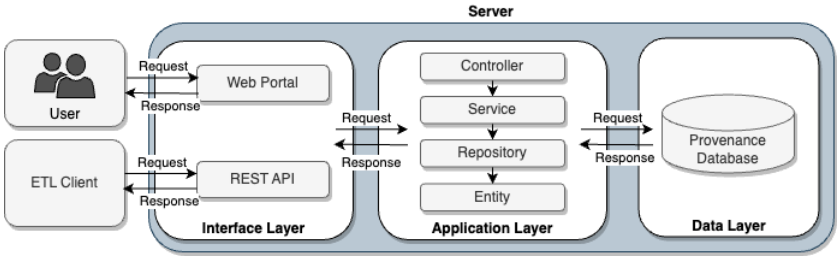

Figure 2: **ProvETL** three-layer architecture.

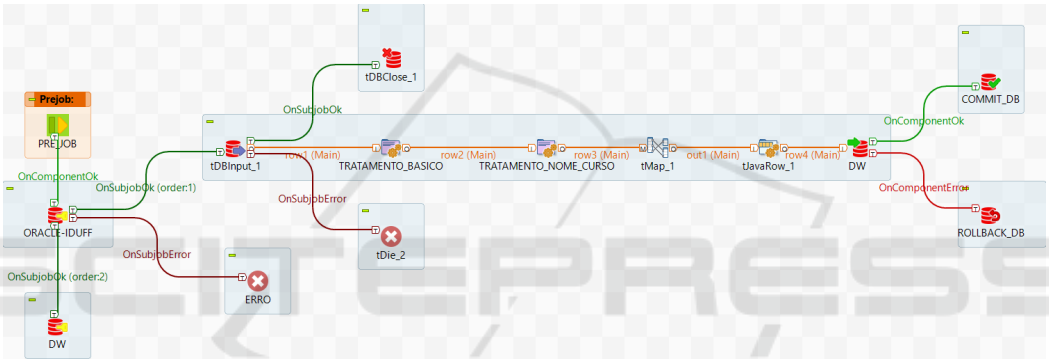

Figure 3: Example of BIDW Talend job that cleans and anonymizes enrolled students' data.

```
#Data summarization
data_in = [1, 2, 3, 4, 5]
payload = {"executedBy": current_user,
   "startedAt": current_time,
   "numberOfInputTuples": len(data_in)}
#Request building and posting
endp = "dataflows/1/transformations/3"
headers = {'Content-Type': 'application/json'}
requests.post(endp, data=json.dumps(payload),
   headers=headers)
```

The injected script will execute whenever the ETL routine runs, sending the r-prov data to **ProvETL** server-side. Received requests will be persisted into the database to be further queried in the graphical interface that presents the provenance graph.

4 ProvETL EVALUATION

We evaluated **ProvETL** in a practical BIDW of our University, encompassing six data marts and more than one million tuples. In particular, in this section, we discuss three case studies that showcase **ProvETL** capabilities in using provenance data for monitoring the ETL routines, detecting poor quality data (debugging), and accountability.

Experimental Setup. Our BIDW consumes data from several relational data sources, unstructured text files, and spreadsheets produced by internal and external sources. The Data Mart subjects include courses, students, dropouts, research projects, and staff information, with the last Data Mart directly supporting the dean of the university and the counselors. All components of the BIDW undergo a series of ETL routines implemented as Talend jobs. The BIDW model allows integration with existing ETL tools, but the project managers decided on Talend jobs. These jobs execute as pieces of Java code, and their outputs result in commits to an Oracle DBMS 12.2. Each ETL routine writes to a single BIDW table (Dimension or Fact) to ensure better isolation.

Figure 3 presents an example of an ETL routine responsible for cleaning and anonymizing undergraduate students' data. Each workflow component

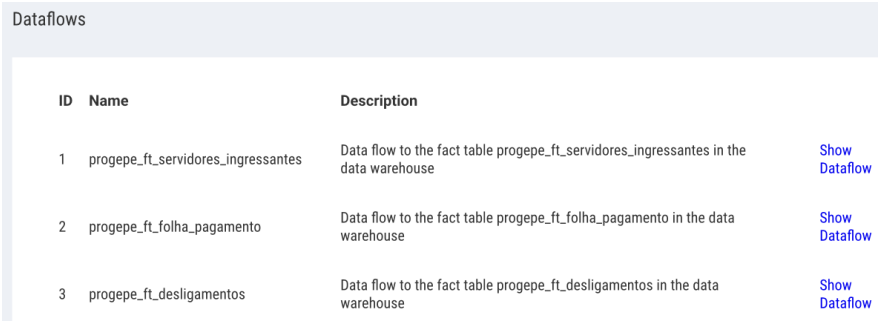

Figure 4: The three dataflows examined as **ProvETL**'s case studies.

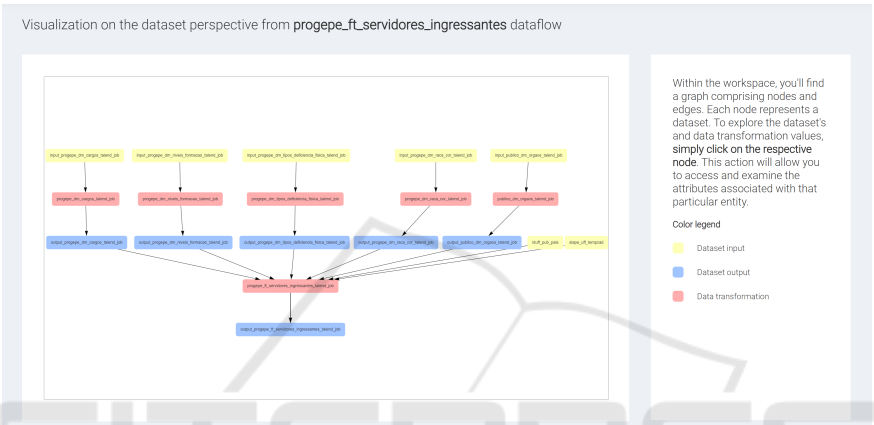

Figure 5: **ProvETL** provenance graph for the first BIDW case study.

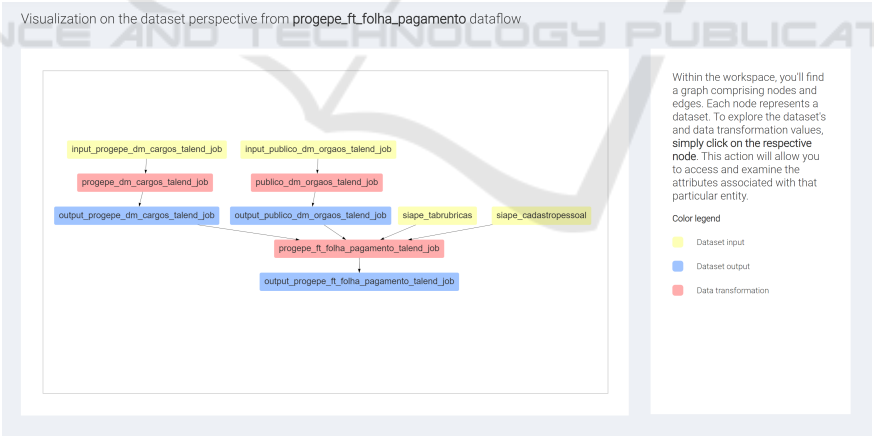

Figure 6: **ProvETL** provenance graph for the second BIDW case study.

performs a specific step, such as connecting to the DBMS, consuming input data, adjusting strings, treating missing values, and replacing coded terms until loading processed data into the DBMS. To show the potential of **ProvETL** in collecting provenance data from these types of ETL routines, we set up **ProvETL** for the case studies, following a similar step-by-step process discussed in Section 3. The case studies were selected after discussions with BIDW analysts, with staff personnel identified as the most relevant subject of interest. Figure 4 lists the three dataflows of interest.

We instrumented the three dataflows by injecting provenance instructions directly into the Talend jobs. We observe there is a native offer to package HTTP requests as a Talend component, which eases the **ProvETL** communication as it relies on being able to handle HTTP messages to collect provenance data.

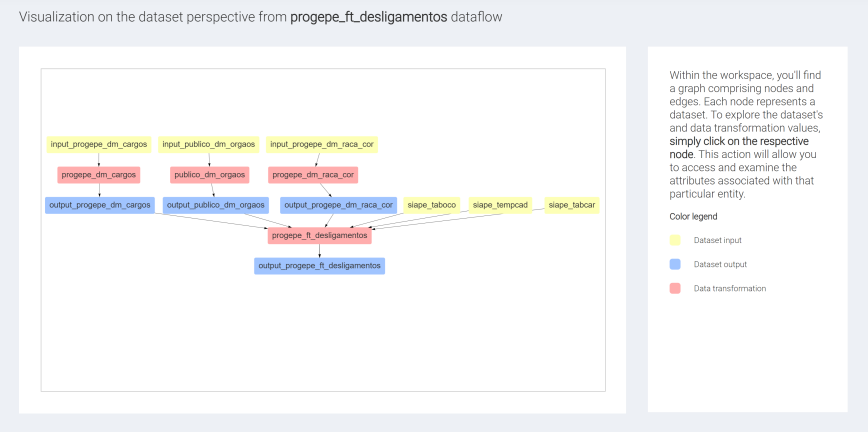

Figure 7: **ProvETL** provenance graph for the third BIDW case study.

A new component was added to every workflow that defines an ETL routine so that extra Java code could be executed. The inserted snippets make calls for collecting provenance data of interest and integration with **ProvETL** endpoints, taking into account the identifiers provided by the proposed API. All evaluations were conducted on a local workstation equipped with a dual-core Intel i5, 8 GB RAM, and a 500 GB hard drive running Windows 10.

Case Study 01 – Personnel Admission. The first case study involves collecting provenance data from all ETL routines that load data into the Fact table containing measures and metrics associated with recently hired university employees. The table includes references to Dimension tables, such as entry identification, year of admission, employee residence (by region/state), categorical information like nationality, gender, race, and position, as well as numerical measures such as bonuses, workload, and the number of legal dependents. References to Dimension tables are inserted by ETL routines before loading the Fact table, resulting in several routines associated with the Dimension tables and one major routine related to the Fact table that runs after every other ETL transformation. After mapping the dataflow (the first entry in Figure 4) and instrumenting the ETL routines with code injection, we run a data extraction from seven data sources – See Figure 5.

This dataflow summarizes the number of tuples by the `COUNT` aggregation function so the user can identify inconsistencies in the ETL routines. For example, in Case Study 01, the provenance data revealed that 3.22% of the tuples were duplicated in the `dm_tipos_deficiencia_fisica` Dimension table, which stored data regarding types of physical disabilities.

Case Study 02 – Public Paycheck Reports. The second case study involves collecting data from various data sources and populating several Dimensions and one Fact table with public information on personnel salaries and paychecks. Similar to Case Study 01, we instantiated a dataflow in **ProvETL** and defined data of interest in all related ETL transformations. Subsequently, we proceeded to instrument Talend jobs with Java components. After mapping the dataflow and ETL routines (second entry in Figure 4), we executed all the transformations with code injection over four data sources – See Figure 6. Here, the injected routine responsible for loading `PUBLICO_DM_ORGAOS` data revealed that 3.63% tuples were duplicated.

Case Study 03 – Former Personnel. Our final case study collects provenance from various data sources containing information on dismissed and former personnel. To gather the provenance data, we instantiated a dataflow on **ProvETL** and defined the granularity of interest in all connected ETL transformations. Next, we mapped the dataflow and ETL routines (third entry in Figure 4) and ran all the transformations over six data sources. Figure 7 illustrates the dataflow for Case Study 03. In the experimental evaluation over six data sources, all transformations produced no errors and loaded data into the BIDW. However, the injected routine responsible for processing the data to load the Fact table `PROGEPE_FT_DESLIGAMENTOS` within this dataflow revealed that 35.5% of the tuples had the `NIVEL_CLASSIFICACAO` of empty attributes.

Provenance Overhead. Capturing and storing provenance data offers analytical advantages for users to monitor and debug ETL routines. However, it also introduces a processing overhead. We measured the

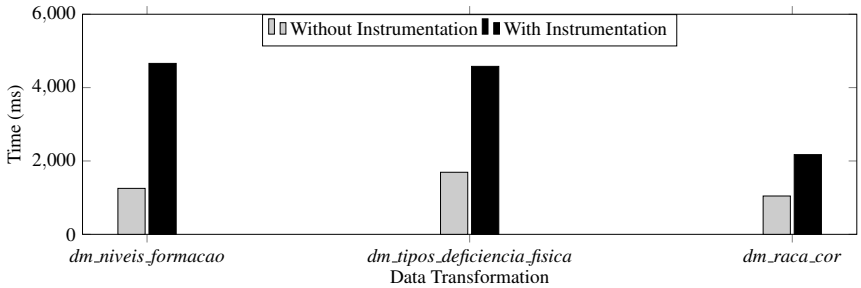

Figure 8: Average elapsed time to perform ETL routines with and without instrumentation.

overhead imposed by capturing provenance in all of our instrumented ETL routines in the BIDW data staging area, *i.e.*, the average elapsed time with and without instrumentation of the ETL routines. We observed that some data transformations significantly dominated others in terms of elapsed time, meaning they were more costly than other parallel transformations. As the Fact tables in all dataflows act as barriers for parallel ETL executions, we report only the cost of those dominant data transformations in Figure 8, *i.e.*, routines `dm_niveis_formacao`, `dm_tipos_deficiencia_fisica`, and `dm_raca_cor`. Figure 8 presents the average elapsed times after ten executions for each data transformation. While there is a large margin for improvement in future work, the execution time was still within milliseconds, which was deemed affordable by the BIDW analysts.

5 CONCLUSIONS

In this paper[2], we discussed integrating provenance mechanisms into ETL routines through a provenance-aware extension to BIDWs, named **`ProvETL`**. We evaluated **`ProvETL`** in three real-world scenarios involving personnel admission, paycheck reports, and staff dismissals, using data from our University BIDW. We effectively detected quality inputs and identified potentially problematic transformations within acceptable processing overhead.

REFERENCES

Corrigan, D., Curcin, V., et al. (2019). Challenges of deploying computable biomedical knowledge in real-world applications. In *AMIA 2019*.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Computing in Sc. & Eng.*, 10(3):11–21.

Groth, P. and Moreau, L. (2013). W3C PROV - An Overview of the PROV Family of Documents.

Herschel, M., Diestelkämper, R., and Lahmar, H. B. (2017). A survey on provenance: What for? what form? what from? *The VLDB Journal*, 26(6):881–906.

Kimball, R. and Ross, M. (2011). *The data warehouse toolkit: the complete guide to dimensional modeling*.

Kwasnikowska, N., Moreau, L., and Bussche, J. V. D. (2015). A formal account of the open provenance model. *ACM Trans. Web*, 9(2).

McPhillips, T. M. et al. (2015). Yesworkflow: A user-oriented, language-independent tool for recovering workflow information from scripts. *CoRR*, abs/1502.02403.

Murta, L., Braganholo, V., Chirigati, F., Koop, D., and Freire, J. (2015). noworkflow: capturing and analyzing provenance of scripts. In *IPAW*, pages 71–83.

Nargesian, F., Pu, K. Q., Bashardoost, B. G., Zhu, E., and Miller, R. J. (2023). Data lake organization. *IEEE Trans. Knowl. Data Eng.*, 35(1):237–250.

Rahm, E. and Bernstein, P. A. (2006). An online bibliography on schema evolution. *SIGMOD Rec.*, 35(4):30–31.

Reis Jr, C. P., da Silva, W. M., Martins, L. C., Pinheiro, R., Victorino, M. C., and Holanda, M. (2019). Enhancing open government data with data provenance. In *Int. Conf. on Man. of Dig. EcoSystems*, pages 142–149.

Rudniy, A. (2022). Data warehouse design for big data in academia. *Computers, Materials & Continua*, 71(1).

Silva, R. F. and others. (2021). Workflows community summit: Bringing the scientific workflows research community together.

Vassiliadis, P., Simitsis, A., and Skiadopoulos, S. (2002). Conceptual modeling for etl processes. In *DOLAP*, DOLAP '02, page 14–21. ACM.

Zheng, N., Alawini, A., and Ives, Z. G. (2019). Fine-grained provenance for matching & etl. In *ICDE*, pages 184–195.

[2]This study was funded in part by FAPERJ - G. SEI E-26/202.806/2019 (247357) and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.