Towards Integrating Workflow and Database Provenance[⋆]

Fernando Chirigati and Juliana Freire

Polytechnic Institute of NYU
Computer Science and Engineering Department
fernando.chirigati@gmail.com,juliana.freire@nyu.edu

Abstract. While there has been substantial work on both database and workflow provenance, the two problems have only been examined in isolation. It is widely accepted that the existing models are incompatible. Database provenance is fine-grained and captures changes to tuples in a database. In contrast, workflow provenance is represented at a coarser level and reflects the functional model of workflow systems, which is stateless—each computational step derives a new artifact. In this paper, we propose a new approach to combine database and workflow provenance. We address the mismatch between the different kinds of provenance by using a temporal model which explicitly represents the database states as updates are applied. We discuss how, under this model, reproducibility is obtained for workflows that manipulate databases, and how different queries that straddle the two provenance traces can be evaluated. We also describe a proof-of-concept implementation that integrates a workflow system and a commercial relational database.

Keywords: Workflow Provenance, Database Provenance, Reproducibility

1 Introduction

Provenance for digital objects is becoming increasingly important both in industry and science, not only due to regulations such as HIPAA and Sarbanes Oxley, but also due the fact that computational scientific results must be reproducible [6]. The area of provenance management has been very active and there is a rich body of work on different aspects of provenance. Work on *database provenance* has focused on techniques to represent provenance for tuples in a relational database and to propagate provenance through queries [3]. For *scientific workflows*, there have been proposals that address issues such as capture, modeling, storage, and querying for provenance information [4, 8].

However, an important problem has received much less attention: how to combine database and workflow provenance. For scientific workflows that interact with data stored in databases, unless there is a model that combines the different

⋆ The original publication is available at http://www.springerlink.com/.

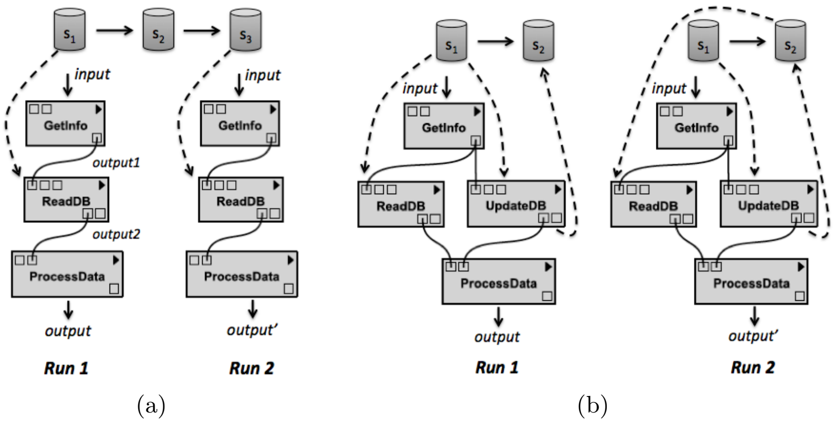


Fig. 1: This figure illustrates challenges in reproducing a workflow run when database access is involved. (a) shows two runs of the same workflow which accesses a database; because the database has changed in between the two runs, even though the workflows have the same structure and input data (i.e., *input*), their results differ—one derives *output* and the other derives *output'*. (b) shows a workflow that, due to its DAG structure, can have two different execution orderings, and depending on the ordering, the outputs may also be different.

kinds of provenance, it is not possible to maintain accurate provenance of the complete process, and consequently, results cannot be reproduced. Consider the example shown in Fig. 1a. The workflow on the left receives as input *input* and outputs *output*. One of its modules, *ReadDB*, consumes data from a database. If this workflow is executed at a later time, even if it uses the same input, there is no guarantee that it will produce the same result. In particular, agents that are external to and not observable by the workflow system may modify the database in between the executions. This happens even in scenarios where database access is always observable by the workflow system. For example, Fig. 1b shows a workflow that both reads from and writes to a database. Note that either *ReadDB* or *UpdateDB* modules can be executed after *GetInfo*. The workflow has two possible execution orderings: if *ReadDB* is executed first (*Run 1*), it will read the initial state s_1 of the database; otherwise, it will read the state that results from the changes applied by *UpdateDB* (*Run 2*). Depending on the execution order, the result produced by the workflow might be different, and without information about how the database changed, the workflow run might not be reproducible.

Combining database and workflow provenance is challenging because of an inherent mismatch in the models used to represent them. Workflows adopt a functional, stateless model where each module is a function that receives some input data and generates a new output: the workflow structure and inputs uniquely identify the outputs [12]. Databases, in contrast, adopt a stateful model: every time a transaction commits, there is a new state that reflects the changes applied in the transaction. Thus, accesses to databases *break* the stateless (and deterministic) scientific workflow model.

In this paper, we propose a new model to integrate database and workflow provenance. To the best of our knowledge, this is the first approach that supports reproducibility of workflows in the presence of database accesses. This model was inspired by the observation that transaction temporal databases provide a suitable abstraction to represent database changes that is compatible to the execution model of scientific workflows. The key intuition here is that, because transaction temporal databases keep track of each state of the database by its transaction time, it is possible to uniquely identify and retrieve each state [9]. Consequently, by recording information about the database states it observes or generates, a workflow system is able to faithfully reproduce workflow executions that involve database access. Because our model relies on a temporal model that is currently supported by commercial relational databases such as Oracle RDBMS [14] and DB2 [5], it is practical and amenable to implementation. In fact, we describe how we have implemented this model in the VisTrails system [7] using the Oracle Total Recall functionality [15].

Besides the ability to accurately reproduce a workflow run, our approach also supports queries that straddle database and workflow provenance. Because the provenance information from the different systems is connected, we have a graph that allows the complete lineage of data artifacts to be computed, e.g., the workflow modules that affected a given database relation, or the relation states that contributed to the derivation of a data product by a workflow. As we discuss in Section 2, it is also possible to obtain the provenance for individual tuples and to answer *how provenance* queries [3].

Related Work. Also with the goal of integrating database and workflow provenance, Acar et al. [1] proposed the use of a *common* provenance model. Their proposal was motivated by the fact that workflow specifications, unlike databases, are seldom accompanied by a formal specification, and this, they argue, makes it difficult to integrate database and workflow provenance. Amsterdamer et al. [2] proposed a framework to integrate the fine-grained database-style provenace with workflows that consist of Pig Latin [13] modules. To capture fine-grained provenance for modules, they translate Pig Latin expressions into nested relational calculus expressions. We attack an orthogonal problem: our goal is to *connect* the two different kinds of provenance so as to support reproducibility. Our approach makes no assumption about the semantics of workflow modules, which can be black boxes, and it also does not prescribe the use of a unified provenance model. Nonetheless, the information captured by our model makes it possible to answer queries that combine database and workflow provenance.

Outline. The remainder of the paper is organized as follows. We present our model that integrates workflow and database provenance in Section 2, where we also discuss how this model supports reproducibility as well as provenance queries. In Section 3, we describe our prototype that combines VisTrails provenance with Oracle Total Recall. We conclude in Section 4 where we outline directions for future work.

2 A Model for Integrating Workflow and Database Provenance

In this section, we begin by introducing some basic concepts about workflows and databases, and then formally define our integrated provenance model.

2.1 Background

Stateless Workflows. We assume a dataflow model for workflows. A workflow is represented as a directed acyclic graph (DAG), where vertices are modules (functions) that perform computations, and data flows through the edges which connect modules.

Definition 1. *A workflow instance W is described by the tuple (M, C), where M is the set of modules and C is the set of connections. Each module $m \in M$ is represented by a function f_m, such that*

$$f_m \,:\, D_m^I \rightarrow D_m^O \tag{1}$$

where D_m^I is the domain of input values and D_m^O is the domain of output values. Since the definition of f_m is unknown, it is considered a black box[1]*. A connection $c \in C$ that connects module m to module n is described as the tuple (m, n, d), where d corresponds to the data product that flows from m to n and that creates the dependency between these modules.*

In the remainder of the paper, we represent a module m as $f_m(I_m) = < O_m >$, where I_m represents the input set and O_m is the output set of m. For instance, the workflow presented in the left side of Fig. 1a can be described by the following functions:

$$f_{GetInfo}(input) = < output_1 >,\ f_{ReadDB}(output_1) = < output_2 >,$$
$$f_{ProcessData}(output_2) = < output >$$

Stateful Databases. Transaction temporal databases keep track of the different states of a database as tuples are added, deleted or updated. Thus, these databases have all the necessary elements to support fine-grained provenance [9, 10] and to achieve reproducibility of results [10].

To model transaction temporal relations, we adapt the backlog scheme proposed by Jensen et al. [11]. A *backlog* is a relation that contains the complete history of changes in another relation. Any tuple affected by an update is added to the append-only backlog, and tuples in the backlog are never updated. Backlogs thus maintain a complete record of modifications in tuples of the database. Each tuple in the backlog can be uniquely identified by its valid and transaction

[1] A module is also associated with a set of parameters whose values may also be used by f_m, and thus contribute to the output of the module. To keep the notation simple, we do not explicitly show these parameters and their values.

times. In our model, it is sufficient to consider only transaction time. We also restrict the data manipulation language to the operations *select*, *insert*, *update* and *delete*. While to simplify the presentation, we focus on single-relation queries and transactions; as we discuss below, the model can naturally handle multiple relations. Similarly, while we assume that separate states are maintained for each relation, rather than for the whole database, states covering all relations can also be supported[2]. This scheme is defined below.

Definition 2. *Given a schema $\Re = (K, A)$ from a transaction temporal relation R, where K is the tuple identifier and A is the set of attributes for R, the schema $\Re_B$ of the corresponding backlog relation R_B is defined as*

$$\Re_B = (K,\, A,\, T,\, Op,\, U)\ , \tag{2}$$

where T is the transaction time when the tuple was included in the backlog, Op is the operation applied to the tuple at time T (I for insertion and D for deletion) and U is the user who managed the operation in the tuple.

When a set of tuples is first inserted into a relation, they are also inserted into the backlog; the transaction time T when the insertion took place is recorded for each tuple, and Op is set to “I”. If a tuple is deleted, this tuple is inserted again in the backlog, but with T set to the transaction time when the deletion was performed, and Op set to “D”. An update operation is represented by a deletion followed by an insertion, both with the same transaction time T.

The transaction time corresponds to the timestamp when the transaction was successfully committed. Consequently, all tuples with the same transaction time T were inserted in the backlog by the same transaction, i.e., they belong to the same state of the relation. A state represents a snapshot of a given relation at a certain time point. Since a new state is created for each successful transaction, we can uniquely identify a state by the transaction time. Because backlogs are append-only, they maintain all information needed to reconstruct each database state, and thus, they provide complete provenance for all tuples.

Definition 3. *The tuples in the backlog relation R_B represent the sequence of states $\mathcal{S}(R)$ for R:*

$$\mathcal{S}(R) = \{(S_1(R), T_1(R)), \ldots,\, (S_n(R), T_n(R))\}\ , \tag{3}$$

where $T_i(R), for\ 1 \le i \le n$, represent transaction times recorded in R_B, and $S_i(R)$ corresponds to the state of R at time i. A state $S_i(R)$ is defined as

$$S_i(R) = \{t_j \in R_B \mid time(t_j) \le T_i(R)\}\ , \tag{4}$$

where t_j is a tuple of R_B and $time(t_j)$ is the transaction time recorded for t_j.

[2] In practice, these choices will be determined by the underlying implementation of the temporal features in the database.

Note that the indices in the states indicate their order in time. Given the states $S_i(R)$ and $S_j(R)$, and $i < j$, then $T_i(R) < T_j(R)$. Using this model, besides being able to identify the states by the transaction times, it is also possible to identify the differences between two states. Below, we use a concrete example to illustrate this.

Definition 4. *The difference (or delta) between two states $S_i(R)$ and $S_j(R)$, where $i < j$, is computed as follows:*

$$\Delta_{j,i}(R) = S_j(R) - S_i(R) \tag{5}$$

Example 1. Consider the following scenario. We have an empty relation *Emp*. A transaction that inserts two tuples in *Emp* is executed and successfully committed at transaction time 10 by user *fchirigati*. Then, at transaction time 15, user *jfreire* commits a transaction that corrects the job information about employee *Robert*. Finally, at transaction time 20, a new tuple is inserted in the relation by user *fchirigati*. The backlog which reflects these operations is shown below.

K	*Name*	*Job*	*T*	*Op*	*U*
1	Robert	Researcher	10	I	fchirigati
2	Claire	Assistant Director	10	I	fchirigati
1	Robert	Researcher	15	D	jfreire
1	Robert	Research Assistant	15	I	jfreire
3	Eric	Administrative Director	20	I	fchirigati

The set of states $\mathcal{S}(Emp)$ corresponds to the different timestamps in the backlog:

$$\mathcal{S}(Emp) = \{(S_1(Emp), 10),\ (S_2(Emp), 15),\ (S_3(Emp), 20)\}$$

The delta $\Delta_{3,1}(Emp)$ between $S_1(Emp)$ and $S_3(Emp)$ is:

1	Robert	Researcher	15	D	jfreire
1	Robert	Research Assistant	15	I	jfreire
3	Eric	Administrative Director	20	I	fchirigati

2.2 Integrating Workflow and Database Provenance

As discussed in Section 1, a key challenge in integrating database and workflow provenance to support reproducibility stems from the inherent mismatch between the two provenance models. In what follows, we show how this problem can be addressed for databases which adopt a temporal transaction model. In order to connect the workflow provenance to the database provenance, we need to capture information about the database states observed by the workflows. Given a module m in a workflow instance W that either consumes or modifies data in a relation R of a transaction temporal database, for each execution of

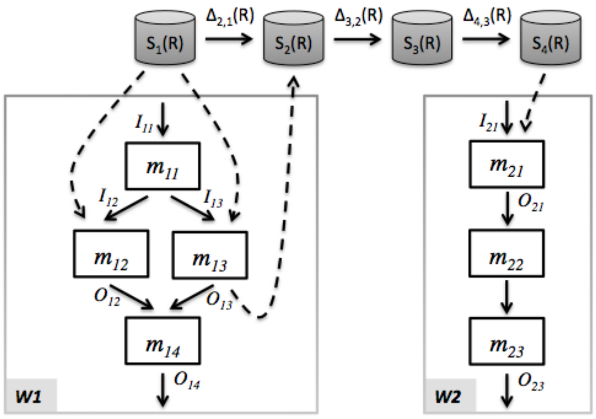


Fig. 2: In the integrated provenance model, database states observed by the workflow are recorded in the provenance (*dashed arrows* represent the database accesses.) Together with the workflow provenance, they not only provide the complete derivation chain for the workflow results, but also enable result reproducibility.

m, we store, in the provenance, information about the states before and after the module execution:

$$f_m(I_m, [R, T_b(R)]) = < O_m, [R, T_a(R)] > \tag{6}$$

where $T_b(R)$ and $T_a(R)$ are the transaction times of the states of relation R before and after the module execution, respectively. Since a transaction time uniquely determines a state, by storing this information, we can retrieve the state.

If $T_a(R)$ and $T_b(R)$ are the same, it means that m did not modify R. Otherwise, if the transaction times are different, a new state was created, and this implies that data was being inserted, updated or deleted from R.

To simplify the presentation, so far we have assumed single-relation queries and transactions. However, it is straightforward to extend the model to work with multiple relations: transaction times must be stored for all relations used in a query. For instance, if a module m retrieves data from relations R_1 and R_2, states of both relations need to be explicit in the model:

$$f_m(I_m, [R_1, T_b(R_1)], [R_2, T_b(R_2)]) = < O_m, [R_1, T_a(R_1)], [R_2, T_a(R_2)] > \tag{7}$$

Fig. 2 shows an example of the information captured by our model. Both workflows (W_1 and W_2) have modules that manipulate data in relation R. The sequence of states for R, which is maintained by the database system, is also shown in the figure. For instance, modules m_{12} and m_{13} of W_1 can be represented by functions

$$f_{m_{12}}(I_{12}, [R, T_1(R)]) = < O_{12}, [R, T_1(R)] > \tag{8}$$

$$f_{m_{13}}(I_{13}, [R, T_1(R)]) = < O_{13}, [R, T_2(R)] > \quad (9)$$

respectively. Given these functions, we know that m_{12} only retrieves data from relation R, since states before and after the execution are the same, and that m_{13} modifies relation R—a new state is created. In this case, we also know that m_{13} is responsible for $\Delta_{2,1}$, i.e., the set of changes applied to $S_1(R)$. Note that the integrated model also exposes parts of the database provenance that, although not directly observed by the workflows, affect their results. In this example, only the changes in $\Delta_{2,1}$ are visible to the workflow. Nonetheless, when W_2 executes, m_{21} reads state $S_4(R)$; as state $S_2(R)$ was the last one managed by the workflow system, we know that $\Delta_{4,2}$ ($\Delta_{4,3} + \Delta_{3,2}$) was performed by external agents.

2.3 Enabling Reproducibility

When a workflow manipulates data in a database, to ensure reproducibility, it is necessary to take into account the database states, since different database states might lead to different results (see Fig. 1a). In our model, database states are uniquely identified and made explicit in the provenance as input and output for the workflow. Because states before the execution of a module are captured in the provenance, by re-enacting the workflow instance using the stored state, it is possible to reproduce its results. Consider, for example, workflow W_2 of Fig. 2. Reproducing this workflow instance is possible because we have recorded the following information:

$$f_{m_{21}}(I_{21}, [R, T_4(R)]) = < O_{21}, [R, T_4(R)] >$$

In this case, if the current state of relation R is not $S_4(R)$, module m_{21} needs to retrieve data from R as if R were in this state. Because we know that the transaction time for the original execution was $T_4(R)$, we can retrieve the corresponding state $S_4(R)$. Then, m_{21} can use this state in its execution, and the results can be correctly reproduced.

2.4 Querying Provenance

With workflow and database provenance integrated, it is possible to perform queries that straddle the workflow and database systems. Below, we use examples to illustrate different queries that are supported by the integrated model.

Lineage of an output. Consider Fig. 2. Using our model, through the set of workflow connections and links to the database states, we can trace the complete provenance of output O_{14}, which includes information about both input data I_{11} and database state $S_1(R)$, which is used to produce outputs that are fed into m_{14}.

$$lineage(O_{14}) = \{W_1, I_{11}, [R, T_1(R)]\}$$

Another important benefit of our model is that the combined provenance includes information about changes to the database which may affect the results of the workflow, even though they may not be directly observed by the workflow system. Consider for example a query to find the lineage of output O_{23}. Even though the changes in $\varDelta_{3,2}(R)$ were performed by agents external to the workflow system, the set of operations are present in the backlog and can thus be retrieved from the database. In addition, by following the provenance links, we can also infer that states $S_1(R)$ and $S_2(R)$, as well as workflow W_1 with input I_{11}, have contributed to O_{23}.

Lineage of a database state. Consider module m_{13} in Fig. 2. The function corresponding to this module that is stored as provenance is shown in Equation 9. From this function, we can infer that the output of m_{13} (O_{13}) depends on its input (I_{13}) as well as on state $S_1(R)$ ($T_1(R)$). Besides, we can also infer that state $S_2(R)$ is derived by this module using input I_{13} and state $S_1(R)$. In this case, we have the following:

$$\mathit{lineage}(S_2(R)) = \{W_1, f_{m_{13}}, I_{13}, [R, T_1(R)]\}$$

Note that, as we have both $T_1(R)$ and $T_2(R)$, we can not only retrieve $S_1(R)$ and $S_2(R)$ from $\mathcal{S}(R)$, but also $\varDelta_{2,1}(R)$. Consequently, it is also possible to construct answers that include fine-grained information about the effect of m_{13} on relation R. In other words, we know exactly which tuples were inserted, updated or deleted by m_{13}.

Lineage of a tuple. Using our model, it is also possible to retrieve the lineage of individual tuples. Given a tuple s inserted in the database by a workflow, we compute its lineage as follows. First, we search the backlog relation for s. By using its unique identifier K (Definition 2), a *select* operation can be performed in the backlog relation to get the set of transaction times T associated with s. Then, we search for each transaction time in $T_i \in T$ in the set M of the modules in workflow instance W. If T_i is an output, and not an input, of function f_m, it means that f_m created the state identified by T_i, i.e., f_m modified s. In this case, W and f_m are included in the lineage of s. We can also retrieve Op from the backlog relation to know exactly what was the modification that f_m performed in s.

How-provenance. If a workflow module modifies a relation R, using our model, it is possible to identify exactly which modifications were performed. As we can retrieve the set $\varDelta_{j,i}(R)$ of modifications from the backlog, we know exactly *how* a module modified the relation from $S_i(R)$ to $S_j(R)$.

3 Implementation

As a proof-of-concept for our model, we have implemented it using the VisTrails system [7, 16] and the Oracle RDBMS [14]. VisTrails is workflow-based data exploration system that provides support for provenance. Oracle is a leader in the relational database market and in their released system, they support temporal database features. Notably, the Total Recall [15] sub-system makes it possible to automatically track every change to the database as well as to query the historical information. Once the Total Recall option is enabled for a relation in the database, an append-only *history table* is created, which keeps track of the tuple-level changes applied to the relation. Like in the backlog scheme [11], each change to the relation recorded in the history table is identified by the transaction time of the modification.

An interesting aspect of Oracle Total Recall is the ability to query in a relation as of a time in the past. Given an identifier to a time in the past, Total Recall recovers the state associated with this time so queries can be performed. The identifier can be either a timestamp or a system change number (SCN), which is an integer that uniquely maps to a timestamp. Total Recall also allows the user to query versions of the relation within a time range, which includes all the modifications that occurred within that range. The syntax of these queries in SQL is as follows:

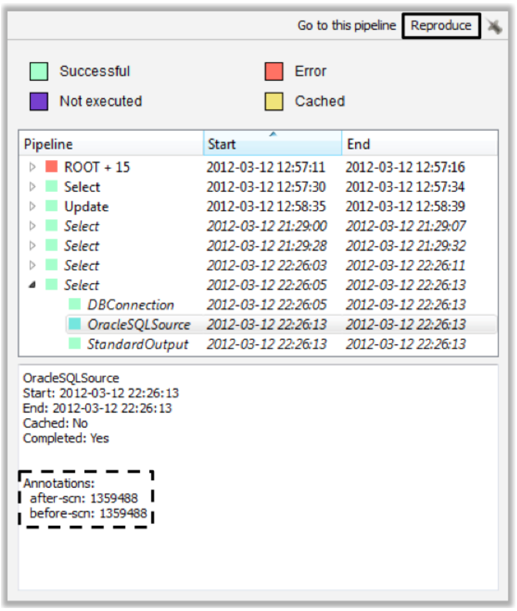
```
SELECT "column_name" FROM "table_name"
       AS OF "time"

SELECT "column_name" FROM "table_name"
       VERSIONS BETWEEN "time_1" AND "time_2"
```

Note that querying as of a time in the past is similar to retrieving a state from $\mathcal{S}(R)$ given its corresponding transaction time. Consequently, this syntax can be used to reproduce previous results. Also, querying between ranges of time is similar to retrieving the difference between states, i.e., the delta (Δ).

The VisTrails Total Recall Package. To support the integration between VisTrails and Oracle Total Recall provenance, we have created a *Total Recall* package for VisTrails[3]. This package consists of three modules: *DBConnection*, *CloseDBConnection* and *OracleSQLSource*. The first two modules are used to open and close a connection with an Oracle database, respectively. The third one is the module used to execute commands in the database. We assume that this module corresponds to the execution of a single transacation.[4] When *OracleSQL-Source* is executed, it automatically retrieves from the database provenance the transaction times, represented as SCN, associated with states before and after its execution ($T_b(R)$ and $T_a(R)$). This information is then recorded in the workflow provenance.

[3] For more information about package creation in VisTrails, we refer the reader to the VisTrails' Users Guide [17].

[4] Note that if multiple transactions are required, they can be modeled using multiple modules.



Pipeline	Start	End
ROOT + 15	2012-03-12 12:57:11	2012-03-12 12:57:16
Select	2012-03-12 12:57:30	2012-03-12 12:57:34
Update	2012-03-12 12:58:35	2012-03-12 12:58:39
Select	*2012-03-12 21:29:00*	*2012-03-12 21:29:07*
Select	*2012-03-12 21:29:28*	*2012-03-12 21:29:32*
Select	*2012-03-12 22:26:03*	*2012-03-12 22:26:11*
Select	*2012-03-12 22:26:05*	*2012-03-12 22:26:13*
DBConnection	*2012-03-12 22:26:05*	*2012-03-12 22:26:13*
OracleSQLSource	*2012-03-12 22:26:13*	*2012-03-12 22:26:13*
StandardOutput	*2012-03-12 22:26:13*	*2012-03-12 22:26:13*

OracleSQLSource
Start: 2012-03-12 22:26:13
End: 2012-03-12 22:26:13
Cached: No
Completed: Yes

Annotations:
after-scn: 1359488
before-scn: 1359488

Fig. 3: Users may request workflows to be reproduced through the VisTrails provenance exploration interface.

So that users can reproduce prior workflow executions, we have extended the VisTrails provenance exploration interface: a user can select a particular execution and request it to be reproduced by clicking on the *reproduce* button (see Fig. 3). When users request to reproduce a workflow execution, the package checks the transaction times for the modules in the workflow that access the database. To simplify the discussion, let us assume there is only one such module. If $T_b = T_a$, it means that the transaction only retrieves data from the relation. For this case, the original query is automatically rewritten: the "AS OF" construct is used to ensure that the query will be run over the database state associated with T_b. It is important to note that the query rewrite is transparent to the user. Fig. 4 illustrates this process. The module *OracleSQLSource* performs a *select* operation over the relation *mountaineers*. When the workflow is executed (Fig. 4a), transaction times before and after the module execution are retrieved from the database provenance ($T_b = T_a = 19546$). When the workflow instance of Fig. 4a is reproduced (Fig. 4b), the system first detects that both T_b and T_a are the same; then, the module automatically modifies the query to retrieve the state associated with SCN = 19546.

If $T_b < T_a$, it means that the workflow module modified the relation. In theory, an approach similar to the read-only queries could be used. However, in practice, this operation is more complicated and its performance depends on the implementation of the underlying database system. The reason for this is the fact that, for the update to be re-applied, the original state must be reconstructed

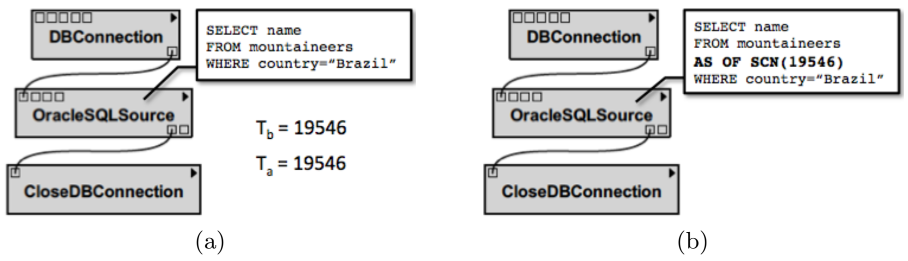


Fig. 4: When the workflow is executed (a), transaction times before and after the execution are retrieved. If this workflow instance is to be reproduced (b), the transaction times are detected to be the same, and then, the query is modified so the correct state can be used.

and materialized: all transactions that committed between T_a and T_b must be rolled back. If there are many such transactions, this operation can be time consuming.

4 Conclusion and Future Work

In this paper, we present a model that integrates database and workflow provenance. Inspired by work on transactional temporal databases, to bridge the gap between the stateless model of scientific workflow systems and stateful databases, our model explicitly captures and stores information about the database states observed by workflows. With this additional information, it is possible not only to reproduce workflow executions, but also to support lineage queries that go across provenance information in a database and workflow system. We have also described a prototype implementation of our model using the VisTrails system and the Oracle RDBMS. This implementation provides evidence that our approach is practical.

While this work provides a first step towards a solution to integrate workflow and database provenance, there are several problems we plan to address in future work. Notably, we would like to further investigate query languages and interfaces for querying the integrated provenance, as well as efficient strategies to evaluate these queries. Our model enables a rich set of queries over the combined provenance. However, some of these queries might be costly to evaluate. Querying the lineage of a tuple, for instance, can take a long time if we have a large set of modules M in the workflow instance W. This problem is compounded for queries that involve multiple workflow instances. Another potentially interesting aspect to consider are changes to the structure of relations, i.e., the data definition language (DDL) operations, which are not captured in the backlog scheme.

Acknowledgments. We thank Dieter Gawlick and Venkatesh Radhakrishnan for insightful discussions on the Oracle Total Recall option and for their guidance in the implementation of our prototype. We also thank Jan Van den Bussche for his feedback on our initial ideas for the integrated model.

References

1. Acar, U., Cheney, J., Bussche, J.V.D., Vansummeren, S., Buneman, P., Kwasnikowska, N.: A graph model of data and workflow provenance. In: Proceedings of the USENIX Workshop on the Theory and Practice of Provenance (TaPP). p. 11 (2010)
2. Amsterdamer, Y., Davidson, S.B., Deutch, D., Milo, T., Stoyanovich, J., Tannen, V.: Putting lipstick on pig: enabling database-style workflow provenance. Proceedings of VLDB Endowment 5(4), 346–357 (Dec 2011)
3. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Foundations and Trends in Databases 1(4), 379–474 (2009)
4. Davidson, S.B., Freire, J.: Provenance and scientific workflows: challenges and opportunities. In: Proceedings of the ACM SIGMOD. pp. 1345–1350 (2008)
5. A matter of time: Temporal data management in DB2 for z/OS (2010)
6. Fomel, S., Claerbout, J.: Guest editors' introduction: Reproducible research. Computing in Science & Engineering 11(1), 5–7 (Jan-Feb 2009)
7. Freire, J., Koop, D., Santos, E., Scheidegger, C., Silva, C., Vo, H.T.: The Architecture of Open Source Applications, chap. VisTrails. Lulu.com (2011)
8. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for computational tasks: A survey. Computing in Science and Engineering 10(3), 11–21 (2008)
9. Gawlick, D., Radhakrishnan, V.: Fine grain provenance using temporal databases. In: In Proceedings of the USENIX Workshop on the Theory and Practice of Provenance (TaPP) (2011)
10. Huq, M.R., Wombacher, A., Apers, P.M.G.: Facilitating fine grained data provenance using temporal data model. In: Proceedings of the International Workshop on Data Management for Sensor Networks (DMSN). ACM (2010)
11. Jensen, C.S., Soo, M.D., Snodgrass, R.T.: Unifying temporal data models via a conceptual model. Information Systems 19, 513–547 (1993)
12. Koop, D., Santos, E., Bauer, B., Troyer, M., Freire, J., Silva, C.T.: Bridging workflow and data provenance using strong links. In: Proceedings of SSDBM. pp. 397–415 (2010)
13. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the ACM SIGMOD. pp. 1099–1110 (2008)
14. Oracle database, `http://www.oracle.com/technetwork/database/enterprise-edition/overview`
15. Oracle total recall with oracle database 11g release 2 (2009)
16. The VisTrails Project, `http://www.vistrails.org`
17. The VisTrails Users' Guide, `http://www.vistrails.org/usersguide`