Provenance for Nested Subqueries

Boris Glavic
Database Technology Research Group
Department of Informatics
University of Zurich
glavic@ifi.uzh.ch

Gustavo Alonso
Systems Group
Department of Computer Science
ETH Zurich
alonso@inf.ethz.ch

1. INTRODUCTION

Data provenance is information about the origin of data. Provenance information can be used to estimate the quality of data, to gain additional insights about it or to trace errors in transformed data back to its origins.

In general, the provenance of a given data item includes all the source data items used as input to the transformation(s) that created the given data item. However, to consider the complete input of a transformation as the provenance of its output is often misleading and contra-intuitive. As a general rule, a certain part of the input influences only part of the output. Given that provenance information tends to be quite large, it is important to narrow down exactly what input data contributes to what output data. In the literature, different interpretations of contribution are used to define which subset of the input of a transformation belongs to the provenance of a given part of the output (e.g., Why-provenance, Where-provenance, etc.) [3, 11, 19, 18, 2, 7]. As a result, the behavior of a provenance management system (PMS) is determined to a great extent by how provenance contribution is defined.

Regardless of the interpretation of provenance contribution used, and to the best of our knowledge, all existing PMS [7, 17, 6] support only a subset of the SQL language. A key part of the functionality missing is support for subqueries embedded within other queries. We refer to such subqueries as sublinks. If a sublink refers to attributes outside the sublink query we call it a correlated sublink. A sublink that includes other sublinks is called nested. Sublinks, whether in the form of correlated and/or nested subqueries, are an important feature in complex decision support queries and scientific applications. They are also widely used in data warehouses. A Provenance Management System that does not support subqueries is of limited use in practice.

In this paper, we present a way to address this limitation and discuss its implementation within the Perm Provenance Management System [12]. Cui and Widom [7] have proposed a definition of provenance contribution that is a variant of Why-provenance for relational tuples. This definition is more suited for the relational model than the variants of Where-provenance introduced in [3, 11] and the annotation propagation semantics from [6, 10]. We have implemented a PMS called Perm that uses the definition of provenance contribution proposed by Cui and Widom. Perm is an enhanced relational database management system (implemented as an extension to PostgreSQL) that allows a user to query the provenance of data using an extension of SQL. Perm uses rewrite rules that transform a query into a query that propagates provenance alongside with the query results. The transformed query is expressed in relational algebra and can thus be executed by the unmodified execution engine of the underlying database system, stored as a view and used as a subquery. The big advantage of Perm is that it does not change the data model and uses SQL to obtain the provenance.

Perm, like all existing PMS, does not support queries with sublinks. Our approach to introduce support for sublinks is based on several steps, each one of them important on its own. First, we show that the definition of provenance contribution used in [7] fails in the presence of sublinks. This definition produces incorrect results (false positives) when
used for queries with sublinks and it is not well defined if there are several sublinks. Second, we show how this definition of contribution (defined next in more detail) can be extended to accommodate sublinks in the queries, thereby producing correct results and supporting an arbitrary number of sublinks (Section 2). Third, we describe several query rewrite strategies that, given a query, rewrite it so that it also computes its provenance according to the extended definition (Section 3). Finally, we present an exhaustive performance study using TPC-H and synthetic data that explores the behavior of these strategies in a real PMS (Section 4). Our experiments demonstrate that provenance for queries with correlated sublinks is inherently an expensive operation. Yet, the most generic strategy we propose can be applied to all types of sublinks for moderate database sizes. For larger data sets, we present additional strategies that are more efficient but are only applicable to uncorrelated sublinks. In doing so, we identify several potential optimizations that will be pursued as part of future work.

2. PROVENANCE AND SUBLINKS

In this section we first define provenance contribution following Cui and Widom. We then introduce an algebra that supports sublinks. We use this algebra to derive the provenance of queries with single sublinks and extend the analysis towards correlation, nesting and queries with more than one sublink. We also provide an extended definition of provenance contribution that accommodates sublinks.

2.1 Defining Provenance Contribution

In \textit{Perm}, the provenance of a tuple \( t \) from the result of an algebra operator \( \mathit{op} \) is a tuple of maximal subsets \((T_1^*, \ldots, T_n^*)\) of the input relations \( T_1, \ldots, T_n \) of the operator \(^1\) that fulfills the following conditions:

**Definition 1 (Provenance Contribution).** For a relational algebra operator \( \mathit{op} \) with input relations \( T_1, \ldots, T_n \), a tuple \((T_1^*, \ldots, T_n^*)\) of maximal subsets of the input relations is said to contribute to the provenance of a tuple \( t \) from the result of \( \mathit{op} \):\footnote{Multiple references to one relation are handled as different relations.}

\[
\begin{align*}
\mathit{op}(T_1^*, \ldots, T_n^*) & = t \\
\forall i \in \{1, \ldots, n\}: \forall t^* \in T_i^* : \\
\mathit{op}(T_1^*, \ldots, T_{i-1}^*, t^*, T_{i+1}^*, \ldots, T_n^*) & \neq \emptyset
\end{align*}
\]

The intuition behind this definition is that a subset of an input relation should be included in the provenance of an result tuple if it produces exactly this tuple (condition 1) and if each tuple in this set contributed to the result (condition 2). This definition naturally captures the semantics of relational operations. For example given a relation \( R = \{ (1,3), (2,2), (3,6) \} \) with schema \( R = (a,b) \) and the relational algebra statement \( \sigma_{a=3}(R) \) the set \{3,6\} is the provenance of the tuple \{3,6\}. As a second example consider the query \( \sigma_{a=3}(T) \). For this query all tuples from \( R \) are in the provenance of the single result tuple \( (6) \).

The provenance for a set of result tuples is defined as the union of the provenance of each of the tuples from the set. For an algebra statement the provenance can be computed incrementally for each operator in the statement, because the provenance according to Definition 1 is transitive.

2.2 Extending the Algebra with Sublinks

The extended relational algebra for sublinks used throughout this paper is presented in Figure 1. The algebra operates on bags (multi-sets). The cardinality of a tuple is denoted by a superset, as in \((1,2)^3\). The schema \( R/q \) of a relation \( R \) or algebra expression \( q \) is a list of attributes \((a, b, \ldots)\). For a \( A \) of attribute expressions (expressions over attributes, constants and functions) and a tuple \( t, t.A \) is the projection of \( t \) on \( A \).

Projection and set operations are provided as a bag semantics version \((e.g., \Pi)\) and a duplicate removing set version \((e.g., \Pi)\). If made clear from the context the bag/set indicator is omitted \((e.g., \Pi)\). The projection operator projects its input on a list of expressions over attributes, constants, functions and renamings (represented by \( a \rightarrow b \)). Aggregation groups on a list of grouping attributes \( G \) and computes the results of the aggregate functions from \textit{agg} for each group of tuples. The result of the aggregation includes the aggregate function results and the values of the grouping attributes (one tuple per group).

The nesting operators \((\textit{ANY}, \textit{ALL}, \textit{EXISTS})\) are the algebraic representation of the SQL constructs with the same name\(^2\) and produce a boolean result. \( T_{\text{sub}} \) is the nested algebra expression, \( \mathit{op} \) is a comparison operator \((e.g., \prec \) or \( = \) and \( A \) is an attribute expression. We refer to the whole construct defined by a nesting operator as \( C_{\text{sub}} \). We permit the use of a sublink query \( T_{\text{sub}} \) without one of the nesting operators, but require \( T_{\text{sub}} \) to produce exactly one result attribute and tuple in this case. Sublinks are allowed in selection, projection and join conditions. For example, the following expressions are valid algebra expressions:

\[
\begin{align*}
\sigma_{a+b}(S) & = \textit{ANY} (S)(R) \\
\Pi_{a, \textit{EXISTS} (\sigma_{c-a}(S))}(R) \\
R \mathit{\prec} a & = \textit{ALL} (T) S \\
\sigma_{a} & = \textit{ANY} (\sigma_{c-a}(S))(R)
\end{align*}
\]

In SQL, sublinks can be used in the GROUP BY and HAVING clause in addition to the SELECT and FROM clause. This can be simulated in our algebra using selection (for HAVING) and projection on sublinks before applying aggregation (for GROUP BY). The second selection example from above contains a correlation \((c = b)\) referencing attribute \( b \) from relation \( R \). Correlation attribute references have to reference an attribute from the input of the operator or, in the case of nested sublinks an attribute from a containing sublink. For example:

\[
\begin{align*}
\sigma_{a} & = \textit{ANY} T_{\text{sub}}(R) \\
T_{\text{sub}} & = \sigma_{c-b \land c} = \textit{ANY} (\sigma_{d=c}(T))(S)
\end{align*}
\]

A correlated attribute parameterizes the sublink query \( T_{\text{sub}} \). For each tuple \( t \) from the algebra expression that is referenced, \( T_{\text{sub}} \) is evaluated for the parameter bound to the value of the referenced attribute.

In what follows, we apply Definition 1 to compute the provenance of queries with sublinks. We distinguish between sublinks in selections and sublinks in projections. For queries without sublinks, Definition 1 produces correct results. For sublinks, the resulting provenance includes tuples that do not actually contribute to the result (false positives). Furthermore, if more than one sublink is used, the definition is ambiguous.

\(^2\)\text{IN and NOT IN} can be simulated with boolean negation and \textit{ANY} / \textit{ALL}.}
2.3 Single Sublinks in Selections

Recall that according to Definition 1, the provenance of an operator \( op \) and result tuple \( t \) is a subset of the input relations used by the operator. For an operator \( op \) that contains sublinks, the provenance thus includes subsets of the sublink queries results. Note that Definition 1 defines only the provenance of single operators and not of algebra statements composed of more than one operator. The provenance for an query is computed by recursively computing the provenance for each operator in the query. Thus tracing the provenance of a tuple from the query result back to the relations in the database (called base relations) that are accessed by the query. This computation is sound because provenance according to Definition 1 is transitive.

At first we consider the uncorrelated case for selection operators containing only a single sublink. In the remainder, we extend our investigations towards other operators, multi-sublink queries and correlation. For a given query \( q \) of the form \( \sigma_C(T) \) that contains a sublink \( S_{\text{sub}} \) in condition \( C \), we are trying to find subsets \( T^* \) of \( T \) and \( S_{\text{sub}} \) that fulfill the conditions of Definition 1 and, thus, form the provenance of \( q \).

To apply Definition 1 to our algebra extended with sublinks some preliminaries are needed. Note that the value of a sublink is constant for a fixed tuple from the operator’s input and fixed tuples of all correlated relations, if the sublink is correlated.

According to Definition 1, a subset \( T_{\text{sub}}^* \) of the sublink query contributes to a tuple \( t \) of the query result iff \( T_{\text{sub}}^* \) produces \( t \) (condition 1), each tuple from \( T_{\text{sub}}^* \) produces not the empty set (condition 2) and \( T_{\text{sub}}^* \) is the maximal subset with these properties. The definition implicitly states a way to find the provenance of an algebra operator: Find a subset of the input that fulfills the conditions and cannot be extended without breaking the conditions. Showing that every tuple of \( T_{\text{sub}} - T_{\text{sub}}^* \) would break the conditions if unioned with an arbitrary subset of \( T_{\text{sub}}^* \) that fulfills the conditions proves that \( T_{\text{sub}}^* \) is maximal.

From the definition of the selection operator we know that if a tuple \( t \) is in the result of the operator, \( t \) is also in the input and \( t \) fulfills the condition \( C \) (denoted by \( t \models C \), see Figure 1). A sublink \( C_{\text{sub}} \) can play different roles in the condition \( C \) of a selection according to an input tuple \( t \). One possible role is that condition \( C \) is only fulfilled, if \( C_{\text{sub}} \) is true. The second possibility is that \( C \) is true iff \( C_{\text{sub}} \) is false. The last role is that \( C \) is independent of the result of \( C_{\text{sub}} \). We refer to these three influence roles as \textit{regtrue}, \textit{regfalse} and \textit{ind}. Note that \( T_{\text{sub}}^* \) can depend on the influence role of the sublink. Therefore, we consider the provenance of each sublink type for each influence role separately.

For the provenance derivation of a query with an ANY- or ALL-sublink we introduce two auxiliary sets \( T_{\text{sub}}\text{true}(t) \) and \( T_{\text{sub}}\text{false}(t) \) that are parameterized by a tuple \( t \) from the input of the selection:

\[
T_{\text{sub}}\text{true}(t) = \{ t' \mid t' \in T_{\text{sub}} \land A \text{ op } t' \}
\]

\[
T_{\text{sub}}\text{false}(t) = \{ t' \mid t' \in T_{\text{sub}} \land \neg(A \text{ op } t') \}
\]
We have to show that condition 2 is not fulfilled for \( \sigma \). Every tuple \( t \) substituted for \( \text{sub} \) fulfills conditions 1 and 2. Let \( \text{sub} \) have to show that \( \text{reqtrue} \) or \( \text{reqfalse} \) or \( \text{ind} \).

**EXISTS or T\(_\text{sub}\) sublink**

\[
T_{\text{sub}}^* = T_{\text{sub}}
\]

Figure 2: Provenance for single sublink queries

contain at least one tuple from \( T_{\text{sub}}^{\text{true}} \). If \( C_{\text{sub}} \) is false then \( C_{\text{sub}} \) is false for all subsets of \( T_{\text{sub}} \), because there are no tuples in \( T_{\text{sub}} \) that fulfill \( A \) op \( t \). These facts can be used to derive the provenance of an ANY-sublink. If the sublink is \( \text{reqtrue} \), \( T_{\text{sub}} \) makes \( C_{\text{sub}} \) true and therefore fulfills condition 1 of the contribution definition, but condition 2 is only fulfilled for tuples from \( T_{\text{sub}}^{\text{true}} \). Thus \( T_{\text{sub}} = T_{\text{sub}}^{\text{true}} \). If the sublink is \( \text{reqfalse} \) or \( \text{ind} \), \( T_{\text{sub}} \) fulfills conditions 1 and 2 and is therefore the provenance of \( t \). Figure 2 gives an overview of the provenance for single sublinks.

It is possible to formally characterize the provenance produced by definition 1 when applied to ANY-sublinks in selections as follows:

**Theorem 1 (Provenance of ANY-Sublinks).** Let \( \sigma_C(T) \) be an selection that contains an ANY-sublink \( C_{\text{sub}} \) in condition \( C \). The provenance of a result tuple \( t \) according to sublink \( C_{\text{sub}} \) is:

\[
T_{\text{sub}}^* = \begin{cases} T_{\text{sub}}^{\text{true}} & \text{if } C_{\text{sub}} \text{ is } \text{reqtrue} \\ T_{\text{sub}} & \text{if } C_{\text{sub}} \text{ is } \text{reqfalse} \text{ or } \text{ind} \end{cases}
\]

**Proof.**

For a selection \( \sigma_C(T) \) with a sublink \( C_{\text{sub}} \) in condition \( C \) we have to show that \( T_{\text{sub}}^* \) as defined above is the maximal set fulfilling conditions 1 and 2. Let \( C(T) \) be condition \( C \) with \( T \) substituted for \( T_{\text{sub}} \). We use \( C^* \) as a shortcut for \( C(T_{\text{sub}}^*) \).

\( C_{\text{sub}}(t) \) be the sublink \( C_{\text{sub}} \) with \( \{t\} \) substituted for \( T_{\text{sub}} \).

**Assume \( C_{\text{sub}} \text{ is } \text{reqtrue} \) \((T_{\text{sub}}^* = T_{\text{sub}}^{\text{true}})\):**

**Condition 1:** We have to show \( \sigma_C(T^*) = t \), which is equivalent to \( C^* = \text{true} \). From the definition of \( T_{\text{sub}}^{\text{true}} \) and the definition of the ANY-sublink (existential quantification) we know that \( C_{\text{sub}}(t^*) \) is true for each tuple \( t^* \in T_{\text{sub}}^{\text{true}} \) and, hence, also for \( T_{\text{sub}}^* \). From \( C_{\text{sub}} \) is \( \text{reqtrue} \) we can deduce that \( C^* \) is fulfilled.

**Condition 2:** As shown for condition 1 \( C_{\text{sub}}(t^*) \) is true for \( t^* \in T_{\text{sub}}^* \) and thus \( \sigma_C(T^*) = t \neq \emptyset \).

Maximality: Assume \( T_{\text{sub}}^* \subset T_{\text{sub}} \) fulfills conditions 1 and 2. Every tuple \( t^* \) from \( T_{\text{sub}} \) does not belong to \( T_{\text{sub}}^{\text{true}} \) and thus does not fulfill \( A \) op \( t^* \). It follows that \( C_{\text{sub}}(t^*) = \text{false} \) holds. If we add such a tuple \( t^* \) to \( T_{\text{sub}}^* \) then condition 2 is not fulfilled for \( t^* \) and thus \( T_{\text{sub}}^* \) is maximal.

**Assume \( C_{\text{sub}} \text{ is } \text{reqfalse} \) or \( \text{ind} \) \((T_{\text{sub}}^* = T_{\text{sub}})\):**

**Condition 1:** We have to show \( \sigma_C(T^*) = t \). From \( T_{\text{sub}}^* = T_{\text{sub}} \) follows \( C^* = C \) and thus \( \sigma_C(T^*) = t \).

**Condition 2:** If \( C_{\text{sub}} \) is \( \text{reqfalse} \) we can deduce from the definition of the ANY-sublink that there exists no tuple \( t^* \in T_{\text{sub}} \) that fulfills condition 1, because \( T_{\text{sub}} \) is the maximal subset of \( T_{\text{sub}} \). Therefore, \( T_{\text{sub}}^* = T_{\text{sub}} \).

![Table 1](image1)

<table>
<thead>
<tr>
<th>( q_1 = \sigma_a = \text{ANY}(\Pi_c(S))(R) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>result tuple</td>
</tr>
<tr>
<td>(1,1)</td>
</tr>
<tr>
<td>(2,1)</td>
</tr>
</tbody>
</table>

![Table 2](image2)

<table>
<thead>
<tr>
<th>( q_2 = \sigma_c &gt; \text{ALL}(\Pi_c(S))(S) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>result tuple</td>
</tr>
<tr>
<td>(3,2)</td>
</tr>
</tbody>
</table>

Figure 3: Examples for sublink provenance

The provenance for ALL-sublinks is derived analogously to the provenance for ANY-sublinks, except that an ALL-sublink uses universal quantification instead of existential quantification. An ALL-sublink is true if the comparison condition is fulfilled for all tuples from \( T_{\text{sub}} \). If \( C_{\text{sub}} \) is true then \( C_{\text{sub}}(T) \) is true for all \( T \subset T_{\text{sub}} \). It follows that \( T_{\text{sub}} \) fulfills conditions 1 and 2, if \( C_{\text{sub}} \) is \( \text{reqtrue} \) or \( \text{ind} \). If \( C_{\text{sub}} \) is false then \( T_{\text{sub}} \) contains at least one tuple \( t \) that does not fulfill condition \( t.A \) op \( t' \), but is allowed to contain tuples that fulfill the condition. If \( C_{\text{sub}} \) is \( \text{reqfalse} \), \( T_{\text{sub}} \) therefore fulfills condition 1, but only tuples from \( T_{\text{sub}}^{\text{false}} \) fulfill condition 2. Thus \( T_{\text{sub}}^* = T_{\text{sub}}^{\text{false}} \). The proof is analog to the proof for ANY-sublinks.

**2.3.2 ALL-sublinks in Selections**

For the provenance derivation of EXISTS-sublinks we can use the fact that an EXISTS-sublink is true if \( T_{\text{sub}} \) produces a result with at least one tuple. Thus, \( C_{\text{sub}}(T) \) is true if \( T \) is a non-empty subset of \( T_{\text{sub}} \). If \( C_{\text{sub}} \) is \( \text{reqtrue} \) then \( T_{\text{sub}} \) fulfills condition 1 and condition 2. If \( C_{\text{sub}} \) is \( \text{reqfalse} \) then \( T_{\text{sub}} = \emptyset \) and thus \( T_{\text{sub}} \) fulfills conditions 1 and 2. In summary, \( T_{\text{sub}}^* = T_{\text{sub}} \) independently of the influence role.

**2.3.3 EXISTS-sublinks in Selections**

The provenance derivation for sublinks without a specific sublink operator is trivial, because a \( T_{\text{sub}} \)-sublink either has a single result tuple or produces the empty set. In both cases condition 1 is fulfilled for \( T_{\text{sub}} = T_{\text{sub}} \). Condition 2 follows from condition 1, because \( T_{\text{sub}} \) contains at most one tuple. Therefore, \( T_{\text{sub}}^* = T_{\text{sub}} \).
2.3.5 Examples

The provenance of three example queries is given in Figure 3. For $q_1$, the sublink is $\text{retrue}$ for all input tuples $t$ from $R$, but for each $t$, only one tuple $t'$ from $S$ in $R^{\text{true}}(t)$. In query $q_2$ the ALL-sublink is also $\text{retrue}$. For tuple $(4,5)$, the only tuple that fulfills condition $C$, all tuples from relation $R$ are included in the provenance of $(4,5)$. In Query $q_3$, the sublink is $\text{refalse}$ for input tuple $(2,1)$ and $\text{reind}$ for input tuple $(3,2)$.

2.4 Single Sublinks in Projections

For sublinks used in a projection instead of a selection, the same influence roles apply as for selection.\(^4\) The main difference is that different tuples from the input of the operator can produce the same result tuple. The set of tuples from the input $T$ that produce a tuple $t$ from the output is the provenance $T^*$ of $t$ according to $T$ (see [4, 12]). Let w.l.o.g. $T^* = \{t_1, \ldots, t_n\}$. We know that for every $t_i \in T^*$, the expression $E$ in which $C_{\text{sub}}$ is used in produces the same result, because otherwise $T^*$ would not fulfill condition 1. For two tuples $t_i$ and $t_j$ from $T^*$, the influence role of $C_{\text{sub}}$ and the sets $T_{\text{sub}}^{\text{true}}$ and $T_{\text{sub}}^{\text{false}}$ can be different. For each tuple $t_i$ from $T^*$, the results established in the last section hold.

Let $T_{\text{sub}}^*(t_i)$ be the provenance of a tuple $t$ for one tuple $t_i \in T^*$, if computed using the results from the last section. Intuitively, one would expect that the provenance of $t$ should include all these sets. Let $\bigcup T_{\text{sub}}^*$ be defined as follows:

$$\bigcup T_{\text{sub}}^* = \bigcup_{i \in \{1, \ldots, n\}} T_{\text{sub}}^*(t_i).$$

We can formally show that the provenance of a result tuple $t$ of a projection that contains sublinks is (1) the union of the provenance derived for each input tuple $t'$ that was used to produce $t$, and (2) that the provenance of each $t'$ is derived as it is done for sublinks in selection.

**Theorem 2** (Provenance of projection sublinks). Let $\Pi_A(T)$ be an projection that contains a sublink $C_{\text{sub}}$ in projection list $A$. The provenance of a result tuple $t$ according to sublink $C_{\text{sub}}$ is:

$$T_{\text{sub}}^* = \bigcup T_{\text{sub}}^*$$

**Proof.**

To prove that a subset $T_{\text{sub}}'$ of $T_{\text{sub}}$ is the provenance of a tuple $t$ according to $T_{\text{sub}}$, we have to show that condition 2 holds for every $t_i$ from $T^*$ and for every $t'$ from $T_{\text{sub}}'$. In addition, condition 1 must hold and $T_{\text{sub}}'$ has to be maximal. Proving that condition 1 is fulfilled breaks down to showing that for every tuple $t_i \in T^*$ the following holds: $A(t) = \Pi_A(t)$, where $A(T)$ is the projection list $A$ with $T$ substituted for $T_{\text{sub}}$.

For $\text{EXISTS}^{-}$ and $\text{sub}^{-}$-sublinks, $T_{\text{sub}}(t_i) = T_{\text{sub}}$ holds for every $i$ and, thus, $\bigcup T_{\text{sub}}^* = T_{\text{sub}}(t_i)$. It follows that conditions 1 and 2 are fulfilled for $T_{\text{sub}}^*$ when $T_{\text{sub}}^*$ is defined. To prove that $T_{\text{sub}}^* = \bigcup T_{\text{sub}}^*$ holds for ANY and ALL sublinks too, we have to show that the conditions from Definition 1 are fulfilled. We only present the proof for ANY-sublinks. The proof for ALL-sublinks is analogous.

\(^4\)The $\text{ind}$ role can appear if the sublink is used in an expression. For instance, $\Pi_{\text{true} \lor \text{ind}} = \text{ANY}(S)(R)$.

Condition 1: We have to show that $\forall t_x \in T^* : \Pi_{\text{true} \lor \text{ind}}(t_x) = t$ holds. That can be proven by showing that $A(t_x) = A$. If $C_{\text{sub}}$ is $\text{retrue}$ then $A(t_x) = A$ if $C_{\text{sub}}(T_{\text{sub}}^*)$ is true. $\bigcup T_{\text{sub}}^*$ contains $T_{\text{sub}}^*(t_x)$ and therefore contains at least one element $t'$ that fulfills condition $t.A$ on $t'$. In consequence $A(t_x) = A$. If $C_{\text{sub}}$ is $\text{refalse}$, no tuple from $T_{\text{sub}}^*$ and, thus, also no tuple from $\bigcup T_{\text{sub}}^*$ fulfills condition $t.A$ on $t'$. Hence, $A(t_x) = A$. At last, if $C_{\text{sub}}$ is ind, $A(t_x)$ is independent of the result of $C_{\text{sub}}$ and $A(t_x) = A$ trivially holds.

Condition 2: Let $t^* \in \bigcup T_{\text{sub}}^*$. It follows that $t^* \in T_{\text{sub}}^*(t_i)$ for some $i$ and thus $\Pi_{\text{true} \lor \text{ind}}(t_i) = t \neq \emptyset$. Because projection is a monotone operator, $\Pi_{\text{true} \lor \text{ind}}(T^*) \neq \emptyset$ holds too.

Maximality: We have to distinguish two cases. If there is at least one tuple $t_i$ for which $C_{\text{sub}}$ is $\text{ind}$ or $\text{refalse}$, then $T_{\text{sub}}^* = T_{\text{sub}}^*$, which is the maximal subset of $T_{\text{sub}}$. If $C_{\text{sub}}$ is $\text{retrue}$ for all $t_i \in T^*$, then for each tuple $t'$ from $T_{\text{sub}} - \bigcup T_{\text{sub}}^*$ the condition $t.A$ on $t'$ is false for each $t_i$ and, hence, for every set $T_{\text{sub}} \subseteq T_{\text{sub}}^*$ fulfilling conditions 1 and 2, $\Pi_{\text{true} \lor \text{ind}}(T_{\text{sub}}') \neq t$ holds. It follows that $T_{\text{sub}}^*$ is maximal. \(\square\)

2.5 Multiple Sublinks

In this section we extend the results stated for single sublinks to queries with multiple sublinks and show that if a selection or projection contains more than one sublink the contribution definition from [7] is not well-defined anymore. We demonstrate how the definition can be extended to produce meaningful results and that the extended definition has the additional advantage that it excludes tuples from the provenance of single sublink queries that do not contribute to the result.

As an example to illustrate the problem consider the following selection on the relations $R = \{(1), (2), (3), \ldots, (100)\}$, $S = \{(1), (5)\}$ and $U = \{(5)\}$ with schemas $R = (a)$, $S = (c)$ and $U = (a)$:

$$\sigma_{C_1 \lor C_2}(U)$$

$$C_1 = (a = \text{ANY} R) \quad C_2 = (a > \text{ALL} S)$$

For the tuple $t = (5)$ from $U$ the sub-condition $C_1$ of the selection condition $C$ is true and the sub-condition $C_2$ false. The problem with this example is that there are no maximal subsets $R^*$ and $S^*$ that fulfill conditions 1 and 2, because a solution that maximizes one set requires that the other set is not maximized. The following solutions both fulfill conditions 1 and 2, but maximize only $R^*$ or $S^*$:

**Solution 1**

$$R^* = \{5\}$$

**Solution 2**

$$R^* = \{1, \ldots, 100\}$$

$$S^* = \{1, 5\}$$

This problem arises because the contribution definition only requires the provenance to produce the same result tuples as the complete input relations, but not to produce the same results for sublink expressions in the query. This leads to the ambiguity examined in the example. Solution 2 fulfills the conditions of the contribution definition, but the results of $C_1$ and $C_2$ are different from the results of $C_1$ and $C_2$ produced by the original query.

Intuitively, the provenance of a tuple $t$ according to a sublink query $T_{\text{sub}}$ should include only tuples that produce the same result of the sublink $C_{\text{sub}}$ as in the original query, because other tuples would only have contributed if a query
from another sublink produced a different result. This behavior can be achieved if the contribution definition is extended by a third condition that restricts the provenance to generate the same results for all sublinks of the query for each input tuple. A side-effect of this restriction is that the ind influence role does not exist anymore, because it allows the provenance to produce a different result for a sublink.

Based on this we propose the following extended definition of provenance contribution:

**Definition 2 (Extended Provenance Contribution).**

Let \( op \) be an algebra operator with inputs \( T_1, \ldots, T_n \) and \( T_{sub_1}, \ldots, T_{sub_m} \). Each \( T_{sub_i} \) denotes an input relation that is used in a sublink \( C_{sub_i} \). Let \( C_{sub}(T_i^*, (t_1, \ldots, t_n)) \) with \( t_i \in T_i \) be the result of \( C_{sub} \) computed for the input tuples from \((t_1, \ldots, t_n)\), if \( T_i^* \) is substituted for \( T_i \). A tuple \( op^* = (T_1^*, \ldots, T_n^*, T_{sub_1}^*, \ldots, T_{sub_m}^*) \) of subsets of the input relations is the provenance of a result tuple \( t \) of the operator iff all elements from \( op^* \) fulfill conditions 1 and 2 of definition 1 and the following condition holds:

\[
\forall t = (t_1^*, \ldots, t_n^*) : \forall i \in [1, \ldots, n] : t_i \in T_i^* : \\
\forall j \in [1, \ldots, m] : \forall t' \in T_{sub_j}^* : \\
C_{sub_j}(T_{sub_j}, t) = C_{sub_j}(t', t) 
\]  

(3)

The extended contribution definition has the effect that the provenance of an operator with multiple sublinks is unique and the provenance for each sublink in a query is the same as for single sublink queries (Except that we only have the \( \text{true} \) and \( \text{false} \) influence roles).

Note that condition 3 is not required for single sublink queries. However, it should be applied to these queries too, because otherwise the provenance can contain tuples that do not contribute to the result of the sublink query (false positives). For instance, consider the query \( \sigma_{a=2 \land b=4} \) over the relations from Figure 3. For the result tuple \( t = (2, 1) \) the sublink is true and ind. Therefore the provenance of the sublink query is \( S^* = S \), but only the tuple \( t' = (2, 4) \) from \( S \) contributed to the result of the sublink.

Note that definition 2 produces the same provenance as definition 1 for queries with no sublink or only a single sublink. This can be formally proven as follows:

**Theorem 3 (Provenance under definition 2).** The provenance of each sublink of a query with multiple sublinks according to definition 2 is the same as for single sublinks according to definition 1.

**Proof.**

We prove this claim by showing that the provenance defined for single sublinks form the maximal sets fulfilling the conditions from definitions 1 and 2. We just present the proof for single input operators and selection, because the proofs for multi input operators and projection are analog.

Let \( t \) be a tuple from the output of selection \( \sigma_{c}(T) \) with a condition \( C \) that contains sublinks \( C_{sub_1}, \ldots, C_{sub_m} \). For a single input tuple the results of the sublinks are fixed and each \( C_{sub_i}(T_{sub_i}^*) \) is required to produce this fixed result. Let \( T_{sub_i}^* \) be the provenance of a sublink computed using the results for single sublinks. Obviously each of these sets fulfills condition 3 from Definition 2.

**Condition 1:** We know that \( C' = C(T_{sub_1}, \ldots, T_{sub_m}) \) is true because \( t \) is in the result of the selection. Therefore, \( \sigma_{C'}(T^*) = t \) holds.

**Condition 2:** Let \( i \) be an arbitrary number from \( \{1, \ldots, n\} \) and \( t^* \in T_{sub_j}' \). \( C' = C(T_{sub_1}', \ldots, t^*, \ldots, t_{sub_m}') \) is true iff \( C_{sub_i}(t^*) \) is true, because each \( C_{sub_i}(T_{sub_i}^*) \) is required to produce the same result as the original query. From the results for single sublinks follows that also \( C_{sub_i}(t^*) \) produces the same result as \( C_{sub_i}(T_{sub_i}) \) and, thus, \( \sigma_{C}(T^*) = \emptyset \).

Maximality: Let \( i \in \{1, \ldots, n\} \). Consider \( T_{sub_i}'' \) an arbitrary subset of \( T_{sub_i} \) that fulfills conditions 1 to 3. If we union \( T_{sub_i}'' \) with a tuple \( t' \) from \( T_{sub_i} - T_{sub_i}' \) we can deduce from the results for single sublinks that \( C_{sub_i}(t') \neq C_{sub_i}(T_{sub_i}) \). Therefore, condition 3 is not fulfilled. □

### 2.6 Correlated Sublinks

The difference between correlated sublinks and uncorrelated sublinks is that for correlated sublinks not only \( C_{sub} \) depends on the input of the operator the sublink is used in, but \( T_{sub} \) depends on the input too. If we consider single input operators like selection or projection, \( T_{sub} \) is constant for a fixed input tuple \( t \) from the operators input \( T \). For selection, a single output tuple is derived from one tuple of the input. Thus, the provenance of a tuple \( t \) is defined as for uncorrelated sublinks.

For projections, more than one input tuple can belong to the provenance of an output tuple \( t \). The attribute values of each of these input tuples \( t' \) parameterize the sublink query \( T_{sub} \). Therefore, the results of the sublink query \( T_{sub} \) can differ, depending on which of the input tuples is used to parameterize it. E.g., for the query \( q = \Pi_{a}(\sigma_{b=4}(S)) \) on the relations from Figure 3, the sublink query \( T_{sub} = \sigma_{b<4}(S) \) has three different results computed for the three tuples from \( R \):

\[
T_{sub}(1, 1) = \{(1, 3)\} \\
T_{sub}(2, 1) = \{(2, 4)\} \\
T_{sub}(3, 2) = \emptyset
\]

Because of the parameterization of the sublink query, the notion of a unique subset \( T_{sub}^* \) does not make sense anymore. For instance, if we compute the provenance of query \( q \) for the result tuple \( t \) (true), there is no set that fulfills condition 1 for both input tuples \((1, 1)\) and \((2, 1)\). We therefore compute the provenance of a projection with correlated sublinks according to an output tuple \( t \) and an input tuple \( t' \). This enables us to apply definition 1 and 2 as for uncorrelated sublinks. For example the provenance of tuple \((true)\) of the result of query \( q \) according to input tuple \((1, 1)\) is:

\[
R^* = \{(1, 1)\} \\
S^* = \{(1, 3)\}
\]

The restriction to a single input tuple seems to be a severe limitation, but in section 3 we demonstrate that this restriction poses no problem for provenance computation.

### 2.7 Nested Sublinks

Definitions 1 and 2 define the provenance of single operator expressions. The provenance of an algebra expression is computed by iteratively computing the provenance for each operator starting at the result of the outermost operator (see [12]). This form of computation is sound, because provenance is defined to be transitive (see [7]). Thus the provenance of nested sublinks can be computed iteratively too, by rewriting the outermost sublink first and using the results of this computation in the following computations.
3. QUERY REWRITE RULES

In this section we introduce algebraic rewrite strategies that transform a query \( q \) into a query \( q^+ \) that computes the provenance for all result tuples of \( q \), and prove the correctness of the rewrite rules used by these strategies. First we introduce the provenance representation used by \( \text{Perm} \) and the rewrite rules used for queries without sublinks. Then we present the \( \text{Gen} \) rewrite strategy that is applicable for all types of sublinks and the \( \text{Left, Move and Unn} \) strategies that are restricted to specific types of sublinks.

### 3.1 Relational Representation of Provenance

In contrast to the method presented in [7], we use a single relation to represent the provenance of a query. Representing the provenance as a tuple of relations has the disadvantage that the representation is not within the relational data model and thus can neither be queried using SQL nor stored in a standard relational DBMS. Additionally, in this representation the association between the original query results and the provenance is lost if the provenance of a set of result tuples is computed.

With the \( \text{Perm} \) provenance management system we follow a different approach, by representing the provenance of a query as a single relation and using algebra expressions to compute the provenance. This has the intrinsic advantage that the provenance computation benefits from query optimization techniques of relational DBMS, can be stored as a materialized view and used as a subquery in standard SQL queries.

In \( \text{Perm} \), the provenance of the result of a query \( q \) is represented as a single relation that contains all the original result tuples. Each original result tuple is extended by the contributing tuples from each base relation accessed by \( q \). If there is more than one contributing tuple from a base relation, the original tuple has to be duplicated. If a query \( q \) accesses base relations \( R_1, \ldots, R_n \), the schema of the provenance of \( q \) is \((\{q_1, \mathcal{P}(R_1), \ldots, \mathcal{P}(R_n)\}, \mathcal{P}(T))\). \( \mathcal{P}(T) \) is a unique renaming of the attributes from \( T \) and is called the provenance schema of \( T \). Due to space constraints we do not present the actual naming scheme used by the \( \text{Perm} \) system, but use the prefix \( p \) for provenance attributes instead.

For example, the provenance of the query \( q_{ex} = \Pi_{a,c}(\sigma_{a < c}(R \times S)) \) with base relations \( R = \{(1, 2), (3, 4)\} \) and \( S = \{(2), (5)\} \) and schemas \( R = (a, b) \) and \( S = (c) \) is:

<table>
<thead>
<tr>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( pa )</th>
<th>( pb )</th>
<th>( pc )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Note that the provenance representation used by \( \text{Perm} \) not only associates a result tuple \( t \) with its provenance, but also associates tuples from different input relations that were used together to derive tuple \( t \). This is similar to How-provenance introduced in [13]. The contribution definition used in [13] generates additional information of how the input tuples where combined to produce an output tuple, but only for a limited subset of relational algebra.

For example, if we represent the provenance of \( q_{ex} \) as \( (R, S) \), the information which tuples from \( R \) and \( S \) were used together to derive the result tuple \( (3, 5) \) is lost. In the \( \text{Perm} \) representation, this information can be derived from the fact that tuples \( (3, 4) \) and \( (5) \) are stored in the same provenance result tuple. This representation is well-suited for queries containing projections with correlated sublinks, because the provenance computed for each parameterization of a sublink query is stored together with the input tuples that where used to parameterize the query.

### 3.2 Rewrite Rules for Standard Operators

\( \text{Perm} \) computes the provenance of a query \( q \) by transforming it into a query \( q^+ \) propagating provenance by applying query rewrite rules (see [12]). In [7], provenance is computed tracing the path of a result tuple back to its origins. By propagating provenance information from the source to the result, the \( \text{Perm} \) provenance computation is similar to annotation propagation approaches, such as DBNotes[6] or MONDRIAN [10].

In \( \text{Perm} \) there is a rewrite rule for each algebra operator. A query is rewritten by iteratively applying the rewrite rules for the algebraic operators used in the query. In Figure 4 we present rewrite rules for the most important algebra operators. For the sake of brevity we do not present the proofs here, but it can be shown that the provenance produced by these rewrite rule fulfills the conditions of definition 1. The \( \bowtie \) operator is used to concat two lists of attribute names. For each rewrite rule, the list \( \mathcal{P} \) of provenance attributes is presented.

For example, reconsider query \( q_{ex} \) from above. Using the rewrite rules from Figure 4 this query is transformed into the following query:

\[
q_{ex}^+ = \Pi_{a,c,p_a,p_b,p_c}(\sigma_{a < c}(\Pi_{a,b} a \rightarrow p_{a,b} b \rightarrow p_{b,c}(R) \times \Pi_{c} c \rightarrow p_{c}(S)))
\]

### 3.3 Gen Rewrite Strategy

We now present the \( \text{Gen} \) rewrite strategy for queries that contain sublinks and prove that the strategy computes provenance according to Definition 2. The \( \text{Gen} \) strategy uses the \( \text{Perm} \) rewrite rules for standard algebraic expressions and additional rewrite rules to transform sublinks.

The two main problems developing rewrite rules for sublinks are that (1) the result of a sublink query is not included in the query result, and that (2) it is not immediately clear
how this solution will perform. It quite complicated in the general case and it is far from clear at performance optimization, the de-correlation approach is evaluated attributes values in the join condition. This is similar to sublink query, produce results for each correlated attribute sublinks but not in standard subqueries.

The sublink query, because correlations are only allowed in sublink. For correlated sublinks it is not possible to simply join the sublink query, because correlations are only allowed in sublinks but not in standard subqueries.

One approach to overcome this problem is to completely de-correlate the query by injecting the top query into the sublink query, produce results for each correlated attribute binding and propagate the bindings throughout the query. The propagated attributes are then used to bind the correlated attributes values in the join condition. This is similar to query de-correlation problem studied in area of query optimization (see [5]). The solutions from this field are only applicable for specific correlations. Even though we do not aim at performance optimization, the de-correlation approach is quite complicated in the general case and it is far from clear how this solution will perform.

To circumvent de-correlation the Gen strategy joins the original query with all theoretically possible provenance tuples and simulates the join by an additional sublink that filters out tuples that do not belong to the provenance. All possible provenance tuples for a sublink can be produced using the cross product of all base relations accessed by the sublink query. In some cases, e.g. if a base relation is the empty set, a tuple consisting only of null values is the provenance for this relation. Therefore, we extend each base relation $R$ with a tuple $null(R)$ that has the same schema as $R$ and all its attributes set to $null$.

Let $Base(T_{sub}) = R_1, \ldots , R_n$ be the list of all base relations accessed by a sublink query $T_{sub}$. The relation CrossBase($T_{sub}$) is defined as follows:

$$CrossBase(T_{sub}) = \Pi_{R_1 \ldots \Pi(R_1)} (R_1 \cup null(R_1)) \times \ldots \times \Pi_{R_n \ldots \Pi(R_n)} (R_n \cup null(R_n))$$

CrossBase($T_{sub}$) is the set of all possible provenance tuples of a sublink $T_{sub}$.

To restrict the CrossBase of a sublink to the actual provenance, we need to know which influence role $C_{sub}$ has for every input tuple $t$. In addition, we need to know the provenance of the sublink query $T_{sub}$. The provenance of the sublink query can be computed using the standard provenance rewrite rules to produce $T_{sub}^+$. To get the influence role, we can use the sublink $C_{sub}$. The join condition can then be formulated using $C_{sub}$ and a condition $C_{sub}'$ that filters out the provenance according to the influence role of $C_{sub}$. As presented in section 2, the provenance of a sublink is either $T_{sub}$, $T_{sub}^{true}$ or $T_{sub}^{false}$. Thus, $C_{sub}'$ is either $true$, $A op t$ or $\neg(A op t)$. We define $C_{sub}' = A op t$, because we do not need an extra selection condition for $true$, and $\neg(A op t)$ is expressed as $\neg C_{sub}'$.

$C_{sub}'$ and $C_{sub}$ are used to define a selection condition $J_{sub}$ for each sublink type:

$$J_{sub} = (C_{sub} \land C_{sub}') \lor \neg C_{sub} \quad \text{(ANY-sublink)}$$

$$J_{sub} = C_{sub} \lor (\neg C_{sub} \land \neg C_{sub}') \quad \text{(ALL-sublink)}$$

$$J_{sub} = true \quad \text{(EXISTS- or Tsub sublink)}$$

By applying logical equivalences, $J_{sub}$ for ANY- and ALL-sublink can be transformed into:

$$J_{sub} = C_{sub}' \lor \neg C_{sub} \quad \text{(ANY-sublink)}$$

$$J_{sub} = C_{sub} \lor \neg C_{sub}' \quad \text{(ALL-sublink)}$$

The condition $J_{sub}$ is used to restrict $T_{sub}^+$ to the actual provenance of $C_{sub}$. The join between CrossBase and $T^+$ is simulated with equality conditions between the attributes.
from CrossBase and the attributes from \( T_{\text{sub}}^+ \). In this comparison we have to consider \textit{null} values as equal and therefore use the comparison operator \( =_n \):
\[
a =_n b \Leftrightarrow a = b \lor (a = \text{null} \land b = \text{null})
\]
The simulated join condition between \( \text{CrossBase} \) and \( T_{\text{sub}}^+ \) is defined as follows:
\[
C_{\text{sub}}^+ = \text{EXISTS} \ (\sigma_{T_{\text{sub}}^+ \land \text{P}(T_{\text{sub}}^+)=n} T_{\text{sub}}^+ (\Pi_{\text{P}(T_{\text{sub}}^+)} \rightarrow T_{\text{sub}}^+ (T_{\text{sub}}^+)))
\]
\[
\lor \ (\neg \text{EXISTS}(T_{\text{sub}}) \land \text{P}(T_{\text{sub}}^+)=n \text{null})
\]
The first \( \text{EXISTS} \) sublink in \( C_{\text{sub}}^+ \) checks that a tuple from the \( \text{CrossBase} \) actually belongs to the provenance of the sublink \( C_{\text{sub}} \). To belong to the provenance a tuple has to be an element of \( T_{\text{sub}}^+ \). This is checked by the condition \( \text{P}(T_{\text{sub}}^+)=n \). In addition, the tuple has to fulfill the condition \( J_{\text{sub}} \). The second \( \text{EXISTS} \) sublink is needed to handle the special case of an empty sublink query result. In this case the provenance attributes are filled with \text{null} values.
\( C_{\text{sub}}^+ \) enables us to define the \textit{Gen} strategy rewrite rules for selections and projections with multiple sublinks. These rules are presented in Figure 5 (G1 and G2).

3.4 Correctness of the Gen strategy

\textbf{Theorem 4 (Gen Strategy Correctness).} For a query \( q \) with sublinks, the provenance computed by the rewritten query \( q^+ \) according to rules G1 and G2 is the provenance derived according to definition 2.

\textbf{Proof.}
To prove the correctness of the \textit{Gen} rewrite rules, we have to show that every tuple \( t \) produced by the rewritten query there is an original tuple with attached provenance, and that for every original tuple with attached provenance this tuple is included in the result of the rewritten query. We only present the proof for selection and ANY-sublinks here.

\textbf{Result Preservation:}
To show that the normal attributes \( T \) of a rewritten query \( q^+ \) contain exactly the tuples from \( q \) we have to show that
\[
\Pi_T(q^+) = \Pi_T(q).
\]
For rewrite rule G1 the input to the outer most selection is: \( \text{inn} = T^+ \times \text{CrossBase}(T_{\text{sub}}) \ldots \times \text{CrossBase}(T_{\text{sub}}) \). Trivially \( \Pi_T(\text{inn}) = \Pi_T(q) \), because none of the \text{CrossBases} generates the empty set. Therefore,
\[
\Pi_T(q^+) = \Pi_T(q) \quad \text{iff} \quad \text{the selection condition } C \text{ is fulfilled for every tuple from } T^+ \text{ and for each condition } C_{\text{sub}}^+ \text{ there is at least one tuple } t \text{ from } \text{CrossBase}(T_{\text{sub}}) \text{ that fulfills condition } C_{\text{sub}}^+. \text{ The first requirement is always meet, because every tuple from the result of } q \text{ fulfills the selection condition } C. \text{ The selection condition } C_{\text{sub}}^+ \text{ filters out tuples from } \text{CrossBase}(T_{\text{sub}}) \text{ that do not belong to the provenance of } C_{\text{sub}}. \text{ If no tuples belong to the provenance of } C_{\text{sub}}, \text{ the second } \text{EXISTS} \text{ condition selects a tuple with all attributes set to } \text{null}. \text{ Thus the second requirement is fulfilled too.}

\textbf{Correct provenance propagation:}
To show that rewrite rule G1 generates the provenance of a result tuple \( t \) according to Definition 2 we have to show that for \( \forall i \in \{1, \ldots, m\} \):
\[
\Pi_{T_{\text{sub}}^+}(T_{\text{sub}}^+(\sigma_{T_{\text{sub}}^+}(q^+))) = \Pi_{T_{\text{sub}}^+}(T_{\text{sub}}^+) \quad \text{holds.}
\]
This is equivalent to proving \( C_{\text{sub}}^+ \) for all tuples \( t' \) from \( \text{CrossBase}(T_{\text{sub}}) \).

\textbf{Proof.}
Let \( t' \) be a tuple from \( \text{CrossBase}(T_{\text{sub}}) \) that belongs to \( T_{\text{sub}}^+ \). The sub-condition \( \text{P}(T_{\text{sub}}^+)=n \) is fulfilled only for such tuples, because it compares \( t' \) with tuples from \( T_{\text{sub}}^+ \). If \( t' \in T_{\text{sub}} \), then \( C_{\text{sub}}^+(t') \) is fulfilled. From \( C_{\text{sub}}^+ \) is true we can deduce that \( C_{\text{sub}}^+ \) is fulfilled and hence \( C_{\text{sub}}^+(t') \) holds.

\textbf{3.5 Example of the Gen strategy}
As an example for an application of the \textit{Gen} strategy consider the query \( q = \sigma_{a=\text{null}}(S_1) \land S_2 \) on relations \( R \) and \( S \) with schemas \( R = (a, b) \) and \( S = (c) \). The rewritten query produced by the rewrite rule G1 is as follows:
\[
q^+ = \sigma_{C_{\text{sub}} \land C_{\text{sub}}^+}(\Pi_{a \land b \land pc \land pc} R) \land \Pi_{c \land pc} (S \land \text{null}(S))
\]
This results in:
\[
C_{\text{sub}} = (a = \text{null}(S))
\]
\[
C_{\text{sub}}^+ = \text{EXISTS} \ (\sigma_{a=\text{null} \lor c}(\Pi_{a \land c \land pc} (S \land \text{null}(S)))) \lor (\neg \text{EXISTS}(S \land \text{null}(S)) \\
\land \text{null}(S)))
\]
\[
C_{\text{sub}} = (a = \text{null}(S))
\]

3.6 Rewrite Strategies for Uncorrelated Sublinks
The \textit{Gen} strategy presented in the last section is inefficient, because it uses a cross product that is restricted by a complex sublink condition and sublinks in general are hard to optimize. Hence we developed strategies that are more efficient but are only applicable for uncorrelated sublinks. These strategies utilize the fact, that for uncorrelated sublinks the sublink query \( T_{\text{sub}}^+ \) contains no correlated attributes and, thus, can be used in a normal join.

For reasons of space, we omit the proofs of correctness for these strategies. The proofs are similar to the prove of the \textit{Gen} strategy.

3.6.1 Left Strategy
As mentioned above, uncorrelated sublinks enable us to directly join rewritten sublink queries. The \textit{Left} strategy uses left-outer-joins to join the results of a query with the provenance of the sublinks. To join only tuples from a rewritten sublink query that belong to the provenance of a sublink we can utilize the join condition \( J_{\text{sub}} \) from the last section. To produce correct results for empty sublink queries we have to use outer joins. The rewrite rules (L1 and L2) of strategy Left are presented in Figure 5. Consider the example query \( q = \sigma_{a=\text{null}}(S_1) \land S_2 \) \( R \). After applying rewrite rule L1, the resulting query \( q^+ \) is:
\[
q^+ = \Pi_{a \land b \land pc \land pc} ( \sigma_{C_{\text{sub}} \land C_{\text{sub}}^+}(\Pi_{a \land b \land pc \land pc} (S \land \text{null}(S)))) \lor (\Pi_{a \land c \land pc} (S \land \text{null}(S)))
\]
This results in:
\[
C_{\text{sub}} = (a = \text{null}(S))
\]
\[
C_{\text{sub}}^+ = (a = c)
\]
3.6.2 Move Strategy

The rewrite rules of the Left strategy have the disadvantage that the sublink $C_{sub}$ is duplicated in the condition $J_{sub}$. This is unproblematic if the query optimizer is aware of the duplication and computes $C_{sub}$ only once. If the duplication is not recognized, query performance will suffer dramatically. To circumvent this potential problem, we introduce the Move strategy that uses modified versions of the Left rewrite rules. These rewrite rules have been modified to move selection sublinks into a projection. Thus we are able to use the results of a sublink in the selection and in condition $J_{sub}$. Rewrite rule $T_1$ uses this strategy for sublinks in selection. Selection condition $C_{tar}$ is selection condition $C$ with all sublinks replaced by the new projection attributes $C_1, \ldots, C_m$. For example, the result of the rewrite for query $q$ from above is:

$$q^+ = \Pi_{a,b,pa,pb,pc}(\sigma_{C_1}(\Pi_{a,b,pa,pb}(a = ALL(S)) \rightarrow C_1(R)) \land \langle(a=c) \Pi_{c,c\rightarrow pc}(S))$$

Rewrite rule $T_2$ is the modified rewrite rule for projection sublinks. A new inner projection on $A'$ is used to project on all expressions from $A$ that do not contain a sublink, and on the sublinks used in $A$. The modified outer projection on $A''$ includes all expressions from $A'$ that do not contain a sublink, all expressions from $A$ that contain sublinks and the provenance attributes of $T^+$. For each projection expression containing sublinks, the sublinks are replaced by the new attributes from $A'$. As an example consider the query $q = \Pi_{a,S}(R)$:

$$q^+ = \Pi_{a,c_1,pa,pb,pc}(\Pi_{a,S \rightarrow c_1(R) \land \langle(a=c) \Pi_{c,c\rightarrow pc}(S))}$$

3.6.3 Unn Strategy

Provenance computation can benefit from the de-correlation and un-nesting techniques developed for query optimization [4, 9, 1, 5, 16, 8, 14]. Besides the fact that these approaches are normally only suitable for specific types of sublink queries, the performance gain can be significant. For provenance queries we expect the performance gain to be even higher, because most techniques transform sublinks into joins for which the provenance rewrite rules are very efficient. In addition, the complex join conditions used in the Left and Move strategies can be omitted.

As an example for un-nesting we introduce the Unn strategy. The Unn strategy uses two rewrite rules $U_1$ and $U_2$ (see Figure 5) for specific types of sublinks. Rule $U_1$ is a specific rewrite for a selection with a condition $C = \text{EXISTS } T_{sub}$. Because the provenance of an EXISTS-sublink includes all tuples from $T_{sub}$ and $C$ is fulfilled only if $C_{sub}$ is true, the provenance can be computed using a simple cross product. Rule $U_2$ rewrites a selection with a condition that is a sin-
gle \textit{ANY}-sublink with an equality comparison. The sublink condition is true if the comparison condition is true. Therefore, $C_{sub}$ is always \textit{re}true and the sublink can be transformed into a join on the comparison condition.

4. IMPLEMENTATION AND EXPERIMENTS

4.1 Implementation

The rewrite strategies presented in the last section have been implemented in the Perm provenance management system [12]. \textit{Perm} is realized as an extension of the PostgreSQL-DBMS [15]. The \textit{Perm} module is located below the PostgreSQL analyzer and operates on the internal query tree representation used in PostgreSQL. The module applies provenance rewrite rules to the query tree produced by the analyzer. The output of the provenance rewrite module is passed to the planner and is subject to the standard query optimization of PostgreSQL. The original parser has been modified to recognize language extensions that trigger the rewrite process. For instance, the keyword \textit{PROVENANCE} is used in the select clause to mark a query for provenance rewriting.

For example, to produce the provenance of the algebra expression $\sigma_{a=3,b=\text{ANY}(S)}(R)$, the following statement in extended SQL can be used:

\begin{verbatim}
SELECT PROVENANCE * FROM R WHERE a = 3 AND b = ANY (SELECT * FROM S);
\end{verbatim}

For a more detailed explanation of the SQL extension used by \textit{Perm} and the adaptation of the rewrite rules for query trees the interested reader is referred to [12].

4.2 Experiments

To evaluate the performance of the sublink rewrite strategies we conducted a series of experiments. All experiments where performed on a 2GHz Intel-dual-core machine with 1GB of main memory running MacOS version 10.15.

4.2.1 TPC-H Benchmark

The first series of experiments uses the TPC-H benchmark [20], because there is no standard benchmark for PMS. TPC-H is a stress test of the ideas presented in this paper, because of its subset of complex queries with different sublink patterns. The TPC-H benchmark consists of 22 query templates that can be randomly parameterized with a query generator. Nine of these queries contain sublinks, from which three contain only uncorrelated sublinks. We restrict the experiments to these queries. The query generator was used to generate 100 instance of each of these queries. The \textit{Gen} strategy was used for all 9 queries with sublinks. For the queries with uncorrelated sublinks (11,15 and 16) the \textit{Left} and \textit{Move} strategy are also applied (none of the queries fulfills the requirements to apply the \textit{Unn} strategy). The average execution times for each strategy and databases of sizes 1MB, 10MB, 100MB and 1GB are given in Figure 6. Queries with a run time of more than 6 hours or that are not supported by a strategy are excluded from the results.

As apparent from the results, the \textit{Gen} strategy scales only for moderate database sizes. For a database size of 10mb the runtime of some queries took several hours (queries 2,17,22). The \textit{Left} and \textit{Move} strategies computed the provenance for uncorrelated sublinks for database sizes up to 1GB. It is important to be aware of the inherent complexity of these queries. For instance, the provenance computation of query 11 for a database size of 1GB produces approximately 38 million tuples. In the experiments the \textit{Move} strategy showed no significant improvement in comparison to the \textit{Left} strategy. The examination of the query plans generated for these strategies showed that the cost-estimation for the complex join conditions used in these strategies were extremely inaccurate. As part of future work we will explore making the query optimization cost-model of PostgreSQL provenance-aware to improve performance.

4.2.2 Synthetic Data

In addition to the TCP-H experiments, we also tested \textit{Perm} on synthetic data. The goal is to discover potential optimizations. We produced tables with two integer attributes (a and b) in sizes from 10 to 500000 tuples. The attribute values where drawn from a gaussian distribution with a fixed mean and a standard derivation of 100 times the table size. For the experiment we used two parameterized queries. The first query is a selection with an \textit{ANY}-sublink and equality
The average run times are given in Figures 7, 8 and 9.

The results demonstrate that the specialized Unn strategy outperforms the other strategies by an order of magnitudes. The Left and Move strategies showed a significant improvement over the Gen strategy.

As these results show our approach can be used to determine the provenance of queries with sublinks. Something that was not possible with any of the existing systems. The inherent cost of computing the provenance for complex queries and large databases is evident in our results. Yet, there is much room for optimization, as the specialized strategies presented in this paper indicate. As part of future work we plan to investigate the applicability and impact of other de-correlation and un-nesting techniques for provenance computation. Another optimization would be to develop new physical operators that propagate provenance. This could lead to significant improvements, because the algebraic rewrites often require the recreation of intermediate results, leading to expensive join or sublink operations.

5. CONCLUSIONS

In this paper we have presented a solution to the problem of computing the provenance of queries with sublinks. To the best of our knowledge, Perm is the first PMS that supports such functionality. Our experiments indicate that our approach is feasible for databases of moderate size. For larger databases we provide optimized strategies and discuss potential optimizations.

6. REFERENCES