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

ABSTRACT

Data provenance is essential in applications such as scientific computing, curated databases, and data warehouses. Several systems have been developed that provide provenance functionality for the relational data model. These systems support only a subset of SQL, a severe limitation in practice since most of the application domains that benefit from provenance information use complex queries. Such queries typically involve nested subqueries, aggregation and/or user defined functions. Without support for these constructs, a provenance management system is of limited use.

In this paper we address this limitation by exploring the problem of provenance derivation when complex queries are involved. More precisely, we demonstrate that the widely used definition of Why-provenance fails in the presence of nested subqueries, and show how the definition can be modified to produce meaningful results for nested subqueries. We further present query rewrite rules to transform an SQL query into a query propagating provenance. The solution introduced in this paper allows us to track provenance information for a far wider subset of SQL than any of the existing approaches. We have incorporated these ideas into the *Perm* provenance management system engine and used it to evaluate the feasibility and performance of our approach.

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

EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia.

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 *Perm*, the provenance of a tuple t from the result of an algebra operator *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 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 op iff:*

$$op({T_1}^*, \ldots, {T_n}^*) = t \tag{1}$$

$$\forall i \in \{1, \ldots, n\} : \forall t^* \in {T_i}^* : op({T_1}^*, \ldots, {T_{i-1}}^*, t^*, {T_{i+1}}^*, \ldots, {T_n}^*) \neq \emptyset \tag{2}$$

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 $\mathbf{R} = (a, b)$ 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 $\alpha_{sum(a)}(R)$. 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.

[1] Multiple references to one relation are handled as different relations.

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 $\mathbf{R}/\mathbf{q}$ of a relation R or algebra expression q is a list of attributes $(a, b, \ldots)$. For a list 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., $\overset{B}{\Pi}$) and a duplicate removing set version (e.g., $\overset{S}{\Pi}$). If made clear from the context the bag/set indicator is omitted (e.g., Π). 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 *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 ($ANY, ALL, EXISTS$) are the algebraic representation of the SQL constructs with the same name[2] and produce a boolean result. T_{sub} is the nested algebra expression, *op* is a comparison operator (e.g., $<$ or $=$) and A is an attribute expression. We refer to the whole construct defined by a nesting operator as C_{sub}. We permit the use of a sublink query T_{sub} without one of the nesting operators, but require T_{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:

$$\sigma_{(a+b)\ =\ ANY\ (S)}(R) \qquad \Pi_{a,\ EXISTS\ (\sigma_{c=3}(S))}(R)$$

$$R \bowtie_{a\ <\ ALL\ (T)\ S} \qquad \sigma_{a\ =\ ANY\ (\sigma_{c=b}(S))}(R)$$

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:

$$\sigma_{a\ =\ ANY\ T_{sub}}(R)$$
$$T_{sub} = \sigma_{c=b \wedge c\ =\ ANY\ (\sigma_{d=c}(T))}(S)$$

A correlated attribute parameterizes the sublink query T_{sub}. For each tuple t from the algebra expression that is referenced, T_{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] IN and NOT IN can be simulated with boolean negation and ANY / ALL.

Set operators

$$\overset{S}{\Pi}_A(T) = \{a = (a_1, \ldots, a_n)^1 \mid \exists t \in T, t.A = a\}$$

$$T_1 \overset{S}{\cup} T_2 = \{t^1 \mid t \in T_1 \vee t \in T_2\}$$

$$T_1 \overset{S}{\cap} T_2 = \{t^1 \mid t \in T_1 \wedge t \in T_2\}$$

$$T_1 \overset{S}{-} T_2 = \{t^1 \mid t \in T_1 \wedge t \notin T_2\}$$

Bag operators

$$\overset{B}{\Pi}_A(T) = \{a = (a_1, \ldots, a_n)^{sum} \mid sum = \sum_{t^n \in T, t.A = a} (n)\}$$

$$T_1 \overset{B}{\cup} T_2 = \{t^{n+m} \mid t^n \in T_1 \wedge t^m \in T_2\}$$

$$T_1 \overset{B}{\cap} T_2 = \{t^{min(n,m)} \mid t^n \in T_1 \wedge t^m \in T_2\}$$

$$T_1 \overset{B}{-} T_2 = \{t^{n-m} \mid t^n \in T_1 \wedge t^m \in T_2\}$$

$$T_1 \times T_2 = \{(t_1, t_2)^{n \times m} \mid {t_1}^n \in T_1 \wedge {t_2}^m \in T_2\}$$

$$T_1 \bowtie_C T_2 = \{t = (t_1, t_2)^{n \times m} \mid {t_1}^n \in T_1 \wedge {t_2}^m \in T_2 \wedge t \models C\}$$

$$\sigma_C(T) = \{t^n \mid t^n \in T \wedge t \models C\}$$

$$T_1 \overset{+}{\bowtie}_C T_2 = \{(t_1, t_2)^{n \times m} \mid {t_1}^n \in T_1 \wedge {t_2}^m \in T_2\} \cup \{(t_1, t_2^{null})^n \mid {t_1}^n \in T_1 \wedge (\nexists t_2 \in T_2 : (t_1, t_2) \models C)\}$$

$$\alpha_{G,aggr}(T) = \{(t.G, res_1, \ldots, res_n)^1 \mid t \in T \wedge \forall i \in \{1, n\} : res_i = aggr_i(\Pi_{B_i}(\sigma_{G=t.G}(T))\}$$

Sublinks

$$A \; op \; ANY \; T_{sub} = \exists t \in T_{sub} : A \; op \; t$$

$$A \; op \; ALL \; T_{sub} = \forall t \in T_{sub} : A \; op \; t$$

$$T_{sub} = a \;\; (\text{ for } \mid T_{sub} \mid = 1 \wedge \mid \mathbf{T_{sub}} \mid = a)$$

$$EXISTS \; T_{sub} = \mid T_{sub} \mid > 0 \Leftrightarrow \exists t \in T_{sub}$$

Figure 1: Relational algebra underlying our approach

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 C_{sub} in condition C, we are trying to find subsets T^* and ${T_{sub}}^*$ of T and T_{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 ${T_{sub}}^*$ of the sublink query contributes to a tuple t of the query result iff ${T_{sub}}^*$ produces t (condition 1), each tuple from ${T_{sub}}^*$ produces not the empty set (condition 2) and ${T_{sub}}^*$ 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 $T_{sub} - {T_{sub}}^*$ would break the conditions if unioned with an arbitrary subset of T_{sub} that fulfills the conditions proves that ${T_{sub}}^*$ 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_{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_{sub} is true. The second possibility is that C is true iff C_{sub} is false. The last role is that C is independent of the result of C_{sub}. We refer to these three influence roles as *reqtrue*, *reqfalse* and *ind*. Note that T^*_{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 ${T_{sub}}^{true}(t)$ and ${T_{sub}}^{false}(t)$ that are parameterized by a tuple t from the input of the selection:[3]

$${T_{sub}}^{true}(t) = \{t' \mid t' \in T_{sub} \wedge t.A \; op \; t'\}$$

$${T_{sub}}^{false}(t) = \{t' \mid t' \in T_{sub} \wedge \neg(t.A \; op \; t')\}$$

2.3.1 ANY-sublinks in Selections

With the influence roles and the auxiliary sets ${T_{sub}}^{true}$ and ${T_{sub}}^{false}$ we are able to derive the provenance for *ANY*-sublinks, by finding maximal subsets ${T_{sub}}^*$ for each influence role that fulfill conditions 1 and 2 from Definition 1. An *ANY*-sublink C_{sub} is true if the comparison condition $A \; op \; t'$ is fullfilled for at least one tuple t' from T_{sub}. Thus, if C_{sub} is true then C_{sub} is true for all subsets of T_{sub} that

[3] If made clear from the context parameter t is omitted.

ANY-sublink

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

ALL-sublink

$$T_{sub}{}^{*} = \begin{cases} T_{sub} & \text{if } C_{sub} \text{ is } \textit{reqtrue} \text{ or } \textit{ind} \\ T_{sub}{}^{false} & \text{if } C_{sub} \text{ is } \textit{reqfalse} \end{cases}$$

EXISTS or $\mathbf{T_{sub}}$ sublink

$$T_{sub}{}^{*} = T_{sub}$$

Figure 2: Provenance for single sublink queries

contain at least one tuple from $T_{sub}{}^{true}$. If C_{sub} is false then C_{sub} is false for all subsets of T_{sub}, because there are no tuples in T_{sub} that fulfill $A\ op\ t'$. These facts can be used to derive the provenance of an ANY-sublink. If the sublink is *reqtrue*, T_{sub} makes C_{sub} true and therefore fulfills condition 1 of the contribution definition, but condition 2 is only fulfilled for tuples from $T_{sub}{}^{true}$. Thus $T^{*}_{sub} = T_{sub}{}^{true}$. If the sublink is *reqfalse* or *ind*, T_{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_{sub} in condition C. The provenance of a result tuple t according to sublink C_{sub} is:*

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

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

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

Condition 1: We have to show $\sigma_{C^{*}}(T^{*}) = t$, which is equivalent to $C^{*} = true$. From the definition of $T_{sub}{}^{true}$ and the definition of the ANY-sublink (existential quantification) we know that $C_{sub}(t')$ is true for each tuple $t' \in T_{sub}{}^{true}$ and, hence, also for $T_{sub}{}^{*}$. From C_{sub} is *reqtrue* we can deduce that C^{*} is fulfilled.

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

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

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

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

Condition 2: If C_{sub} is *reqfalse* we can deduce from the definition of the ANY-sublink that there exists no tuple $t' \in T_{sub}$ that fulfills condition $t.A\ op\ t'$. Therefore, for each tuple $t^{*} \in T_{sub}$ the condition $C_{sub}(t^{*})$ is false and $C(t^{*})$ is true. It follows that for each $t^{*} \in T_{sub}{}^{*}$ the condition $C_{sub}(t^{*})$ is false and hence $\sigma_{C(t^{*})} = t \neq \emptyset$. If C_{sub} is *ind* C is fullfiled independently of the result of C_{sub} and conditions 1 and 2 hold.

Maximality: T_{sub} is the maximal subset of T_{sub}. □

R

a	b
1	1
2	1
3	2

S

c	d
1	3
2	4
4	5

$$q_1 = \sigma_{a\ =\ ANY\ (\Pi_c(S))}(R)$$

result tuple	provenance
(1,1)	$R^{*} = \{(1,1)\}$, $S^{*} = \{(1,3)\}$
(2,1)	$R^{*} = \{(2,1)\}$, $S^{*} = \{(2,4)\}$

$$q_2 = \sigma_{c\ >\ ALL\ (\Pi_a(R))}(S)$$

result tuple	provenance
(4,5)	$R^{*} = \{(1,1),(2,1),(3,2)\}$, $S^{*} = \{(4,5)\}$

$$q_3 = \sigma_{(a=3)\vee\neg(a\ <\ ALL\ (\sigma_{c\neq 1}(\Pi_c(S))))}(R)$$

result tuple	provenance
(2,1)	$R^{*} = \{(2,1)\}$, $S^{*} = \{(2,4)\}$
(3,2)	$R^{*} = \{(3,2)\}$, $S^{*} = \{(2,4),(4,5)\}$

Figure 3: Examples for sublink provenance

2.3.2 ALL-sublinks in Selections

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 fullfiled for all tuples from T_{sub}. If C_{sub} is true then $C_{sub}(T)$ is true for all $T \subseteq T_{sub}$. It follows that T_{sub} fulfills conditions 1 and 2, if C_{sub} is *reqtrue* or *ind*. If C_{sub} is false then T_{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_{sub} is *reqfalse*, T_{sub} therefore fulfills condition 1, but only tuples from $T_{sub}{}^{false}$ fulfill condition 2. Thus $T_{sub}{}^{*} = T_{sub}{}^{false}$. The proof is analog to the proof for ANY-sublinks.

2.3.3 EXISTS-sublinks in Selections

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

2.3.4 T_{sub}-sublinks in Selections

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

2.3.5 Examples

The provenance of three example queries is given in Figure 3. For q_1, the sublink is *reqtrue* for all input tuples t from R, but for each t, only one tuple t' from S is in $R^{true}(t)$. In query q_2 the ALL-sublink is also *reqtrue*. 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 *reqfalse* for input tuple $(2, 1)$ and *reqind* 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_{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_{sub} and the sets ${T_{sub}}^{true}$ and ${T_{sub}}^{false}$ can be different. For each tuple t_i from T^*, the results established in the last section hold.

Let ${T_{sub}}^*(t_i)$ 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 $\bigcup {T_{sub}}^*$ be defined as follows:

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

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_{sub} in projection list A. The provenance of a result tuple t according to sublink C_{sub} is:*

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

PROOF.
To prove that a subset ${T_{sub}}'$ of T_{sub} is the provenance of a tuple t according to T_{sub}, we have to show that condition 2 holds for every t_i from T^* and for every t' from ${T_{sub}}'$. In addition, condition 1 must hold and ${T_{sub}}'$ 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: $\Pi_{A(\bigcup {T_{sub}}^*)}(t_i) = t$, where $A(T)$ is the projection list A with T substituted for T_{sub}.

For *EXISTS*- and T_{sub}-sublinks, $T_{sub}(t_i) = T_{sub}$ holds for every i and, thus, $\bigcup {T_{sub}}^* = T_{sub}(t_i)$. It follows that conditions 1 and 2 are fulfilled for ${T_{sub}}^* = T_{sub}$. To prove that ${T_{sub}}^* = \bigcup {T_{sub}}^*$ 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 analog.

[4]The *ind* role can appear if the sublink is used in an expression. For instance, $\Pi_{true \vee a = ANY\ (S)}(R)$.

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

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

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

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 $\mathbf{R} = (b)$, $\mathbf{S} = (c)$ and $\mathbf{U} = (a)$:

$$\sigma_{C_1 \vee C_2}(U)$$
$$C_1 = (a\ =\ ANY\ R) \qquad C_2 = (a\ >\ ALL\ S)$$

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 is 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	**solution 2**
$R^* = \{5\}$	$R^* = \{1, \ldots, 100\}$
$S^* = \{1, 5\}$	$S^* = \{1\}$

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_{sub} should include only tuples that produce the same result of the sublink C_{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 $C_{sub}(T', (t_1, \ldots, t_n))$ 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' is substituted for T_{sub}. 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^* fullfill conditions 1 and 2 of definition 1 and the following condition holds:*

$$\begin{aligned} &\forall tup = ({t_1}^*, \ldots, {t_n}^*) : \forall i \in \{1, \ldots, n\} : t_i \in {T_i}^* : \\ &\forall j \in \{1, \ldots, m\} : \forall t^* \in {T_{sub_j}}^* : \\ &\quad C_{sub_j}(T_{sub_j}, tup) = C_{sub_j}(t^*, tup) \end{aligned} \tag{3}$$

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 *reqtrue* and *reqfalse* 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 $\sigma_{a=2 \vee a = ANY\ S}(R)$ 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 poofs 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 n sublinks $C_{sub_1}, \ldots, C_{sub_n}$. 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 $C' = C({T_{sub_1}}', \ldots, {T_{sub_n}}')$ 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 $t^* \in {T_{sub_i}}'$. $C' = C({T_{sub_1}}', \ldots, t^*, \ldots, {T_{sub_n}}')$ is true iff $C_{sub_i}(t^*)$ is true, because each $C_{sub_j}({T_{sub_j}}')$ 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, $\sigma_{C'}(T^*) = t \neq \emptyset$.

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 $q = \Pi_{a = ALL\ (\sigma_{b=c}(S))}(R)$ on the relations from Figure 3, the sublink query $T_{sub} = \sigma_{b=c}(S)$ has three different results computed for the three tuples from R:

$$\begin{aligned} T_{sub}(1, 1) &= \{(1, 3)\} \\ T_{sub}(2, 1) &= \{(2, 4)\} \\ T_{sub}(3, 2) &= \emptyset \end{aligned}$$

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 $(true)$, 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:

$$R^* = \{(1, 1)\} \qquad S^* = \{(1, 3)\}$$

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 outmost sublink first and using the results of this computation in the following computations.

$$R^+ = \Pi_{\mathbf{R},\mathcal{R}}(R) \text{ with } \mathcal{P}(R^+) = \mathcal{R} \text{ where } \mathcal{R} \text{ is a unique renaming of the attributes from } \mathbf{R} \quad \text{(R1)}$$

$$(\overset{S/B}{\Pi}_A(T))^+ = \overset{S/B}{\Pi}_{A,\mathcal{P}(T^+)}(T^+) \text{ with } \mathcal{P}((\overset{S/B}{\Pi}_A(T))^+) = \mathcal{P}(T^+) \quad \text{(R2)}$$

$$(\sigma_C(T))^+ = \sigma_C(T^+) \text{ with } \mathcal{P}((\sigma_C(T))^+) = \mathcal{P}(T^+) \quad \text{(R3)}$$

$$(T_1 \times T_2)^+ = {T_1}^+ \times {T_2}^+ \text{ with } \mathcal{P}((T_1 \times T_2)^+) = \mathcal{P}({T_1}^+) \blacktriangleright \mathcal{P}({T_2}^+) \quad \text{(R4)}$$

$$(\alpha_{G,aggr}(T))^+ = \Pi_{G,aggr,\mathcal{P}(T^+)}(\alpha_{G,aggr}(T) \overset{\cdot}{\bowtie}_{G=\hat{G}} \Pi_{G\to\hat{G},\mathcal{P}(T^+)}(T^+)) \text{ with } \mathcal{P}((\alpha_{G,aggr}(T))^+) = \mathcal{P}(T^+) \quad \text{(R5)}$$

Figure 4: *Perm* provenance rewrite rules

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 *Perm* and the rewrite rules used for queries without sublinks. Then we present the *Gen* rewrite strategy that is applicable for all types of sublinks and the *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 *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 *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 $(\mathbf{q}, \mathcal{P}(R_1), \ldots, \mathcal{P}(R_n))$. $\mathcal{P}(T)$ 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 *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 $\mathbf{R} = (a,b)$ and $\mathbf{S} = (c)$ is:

a	**c**	**pa**	**pb**	**pc**
1	2	1	2	2
1	5	1	2	5
3	5	3	4	5

Note that the provenance representation used by *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 *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

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 *Perm* provenance computation is similar to annotation propagation approaches, such as DBNotes[6] or MONDRIAN [10].

In *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 $\blacktriangleright$ 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:

$${q_{ex}}^+ = \Pi_{a,c,pa,pb,pc}(\sigma_{a<c}(\Pi_{a,b,a\to pa,b\to pb}(R) \times \Pi_{c,c\to pc}(S)))$$

3.3 Gen Rewrite Strategy

We now present the *Gen* rewrite strategy for queries that contain sublinks and prove that the strategy computes provenance according to Definition 2. The *Gen* strategy uses the *Perm* rewrite rules for standard algebra 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

***Gen* strategy rewrite rules**

$$(\sigma_C(T))^+ = \sigma_{C \wedge C_{sub_1}{}^+ \ldots \wedge C_{sub_n}{}^+}(T^+ \times CrossBase(T_{sub_1}) \ldots \times CrossBase(T_{sub_n})) \quad \text{(G1)}$$

$$(\Pi_A(T))^+ = \sigma_{C_{sub_1}{}^+ \ldots \wedge C_{sub_n}{}^+}(\Pi_{A, \mathcal{P}(T^+)}(T^+) \times CrossBase(T_{sub_1}) \ldots \times CrossBase(T_{sub_n})) \quad \text{(G2)}$$

$$C_{sub_i}{}^+ = EXISTS\ (\sigma_{J_{sub_i} \wedge \mathcal{P}(T_{sub_i}{}^+) =_n T_{sub_i}{}'}(\Pi_{\mathcal{P}(T_{sub_i}{}^+) \rightarrow T_{sub_i}{}'}(T_{sub_i}{}^+)) \vee (\neg EXISTS(T_{sub}) \wedge \mathcal{P}(T_{sub}{}^+) =_n null)$$

***Left* strategy rewrite rules**

$$(\sigma_C(T))^+ = \Pi_{\mathbf{T}, \mathcal{P}(T), \mathcal{P}(T_{sub_1}), \ldots, \mathcal{P}(T_{sub_n})}(\sigma_C(T^+ \overline{\bowtie}_{J_{sub_1}} T_{sub_1}{}^+ \ldots \overline{\bowtie}_{J_{sub_n}} T_{sub_n}{}^+) \quad \text{(L1)}$$

$$(\Pi_A(T))^+ = \Pi_{A, \mathcal{P}(T), \mathcal{P}(T_{sub_1}), \ldots, \mathcal{P}(T_{sub_n})}(T^+ \overline{\bowtie}_{J_{sub_1}} T_{sub_1}{}^+ \ldots \overline{\bowtie}_{J_{sub_n}} T_{sub_n}{}^+) \quad \text{(L2)}$$

***Move* strategy rewrite rules**

$$(\sigma_C(T))^+ = \Pi_{\mathbf{T}, \mathcal{P}(T^+), \mathcal{P}(T_{sub_1}), \ldots, \mathcal{P}(T_{sub_n})}(\sigma_{C_{tar}}(\Pi_{\mathbf{T}, \mathcal{P}(T^+), C_{sub_1} \rightarrow C_1, \ldots, C_{sub_m} \rightarrow C_m}(T^+) \overline{\bowtie}_{J_{sub_1}} T_{sub_1}{}^+ \ldots \overline{\bowtie}_{J_{sub_n}} T_{sub_n}{}^+)) \quad \text{(T1)}$$

$$(\Pi_A(T))^+ = \Pi_{A'', \mathcal{P}(T), \mathcal{P}(T_{sub_1}), \ldots, \mathcal{P}(T_{sub_n})}(\Pi_{A'}(T^+) \overline{\bowtie}_{J_{sub_1}} T_{sub_1}{}^+ \ldots \overline{\bowtie}_{J_{sub_n}} T_{sub_n}{}^+) \quad \text{(T2)}$$

***Unn* strategy rewrite rules**

$$(\sigma_{\ EXISTS\ T_{sub}}(T))^+ = T^+ \times T_{sub}{}^+ \quad \text{(U1)}$$

$$(\sigma_{x\ =\ ANY\ (T_{sub})}(T))^+ = T^+ \bowtie_{x = \mathbf{T_{sub}}} T_{sub}{}^+ \quad \text{(U2)}$$

Figure 5: Provenance query rewrite rules for sublinks

how to determine the influence role of a sublink. We approach the first problem by joining the original query with the sublink query, and the second by restricting the join to filter out tuples according to the influence role of the 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:

$$CrossBase(T_{sub}) = \Pi_{\mathbf{R_1} \rightarrow \mathcal{P}(R_1)}(R_1 \cup null(R_1)) \times \ldots \times \Pi_{\mathbf{R_n} \rightarrow \mathcal{P}(R_n)}(R_n \cup null(R_n))$$

$CrossBase(T_{sub})$ 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:

$$J_{sub} = (C_{sub} \wedge C'_{sub}) \vee \neg C_{sub} \quad \text{(ANY-sublink)}$$

$$J_{sub} = C_{sub} \vee (\neg C_{sub} \wedge \neg C'_{sub}) \quad \text{(ALL-sublink)}$$

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

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

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

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

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 ${T_{sub}}^+$. In this comparison we have to consider *null* values as equal and therefore use the comparison operator $=_n$:

$$a =_n b \Leftrightarrow a = b \vee (a = null \wedge b = null)$$

The simulated join condition between *CrossBase* and ${T_{sub}}^+$ is defined as follows:

$$\begin{aligned}{C_{sub}}^+ = EXISTS\ (&\sigma_{J_{sub} \wedge \mathcal{P}({T_{sub}}^+) =_n {T_{sub}}'}(\\ &\Pi_{\mathcal{P}({T_{sub}}^+) \to {T_{sub}}'}({T_{sub}}^+)) \\ &\vee (\neg EXISTS(T_{sub}) \wedge \mathcal{P}({T_{sub}}^+) =_n null)\end{aligned}$$

The first *EXISTS* sublink in ${C_{sub}}^+$ checks that a tuple from the *CrossBase* actually belongs to the provenance of the sublink C_{sub}. To belong to the provenance a tuple has to be an element of ${T_{sub}}^+$. This is checked by the condition $\mathcal{P}({T_{sub}}^+) =_n {T_{sub}}'$. In addition, the tuple has to fulfill the condition J_{sub}. The second *EXISTS* sublink is needed to handle the special case of an empty sublink query result. In this case the provenance attributes are filled with *null* values.

${C_{sub}}^+$ enables us to define the *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

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.*

PROOF.
To prove the correctness of the *Gen* rewrite rules, we have to show that each 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.

Result Preservation:
To show that the normal attributes $\mathbf{T}$ of a rewritten query q^+ contain exactly the tuples from q we have to show that $\overset{S}{\Pi}_{\mathbf{T}}(q^+) = \overset{S}{\Pi}_{\mathbf{T}}(q)$. For rewrite rule *G1* the input to the outer most selection is: $inn = T^+ \times CrossBase(T_{sub_1}) \ldots \times CrossBase(T_{sub_n})$. Trivially $\overset{S}{\Pi}_{\mathbf{T}}(inn) = \overset{S}{\Pi}_{\mathbf{T}}(q)$, because none of the *CrossBases* generates the empty set. Therefore, $\overset{S}{\Pi}_{\mathbf{T}}(q^+) = \overset{S}{\Pi}_{\mathbf{T}}(q)$ iff the selection condition C is fulfilled for ever tuple from T^+ and for each condition ${C_{sub_i}}^+$ there is at least one tuple t_i from $CrossBase(T_{sub_i})$ that fulfills condition ${C_{sub_i}}^+$. The first requirement is always meet, because every tuple from the result of q fulfills the selection condition C. The selection condition ${C_{sub_i}}^+$ filters out tuples from $CrossBase(T_{sub_i})$ that do not belong to the provenance of C_{sub_i}. If no tuples belong to the provenance of C_{sub_i}, the second *EXISTS* condition selects a tuple with all attributes set to *null*. Thus the second requirement is fulfilled too.

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 $\forall i \in \{1, \ldots, m\} : \overset{S}{\Pi}_{\mathcal{P}(T_{sub_i}) \to \mathbf{T_{sub_i}}}(\sigma_{\mathbf{T}=t}(q^+)) = \overset{S}{\Pi}_{\mathbf{T_{sub_i}}}({T_{sub_i}}^*)$ holds. This is equivalent to proving ${C_{sub_i}}^+(t') \Leftrightarrow t' \in {T_{sub_i}}^*$ for all tuples t' from $CrossBase\ (T_{sub_i})$. We ignore the fact that ${T_{sub_i}}^*$ and q^+ have different schemas to omit extra projections and thus keep the proof readable.

$\underline{C_{sub_i} \text{ is } reqtrue}$: ${T_{sub_i}}^* = {T_{sub_i}}^{true}$. Let t' be a tuple from $CrossBase(T_{sub_i})$ that belongs to ${T_{sub_i}}^+$. The sub-condition $\mathcal{P}({T_{sub_i}}^+) =_n {T_{sub_i}}'$ is fulfilled only for such tuples, because it compares t' with tuples from ${T_{sub_i}}^+$. Iff $t' \in {T_{sub_i}}^{true}$ then ${C_{sub_i}}'(t')$ is fulfilled. From C_{sub_i} is true we can deduce that J_{sub_i} is fulfilled and hence ${C_{sub_i}}^+(t') \Leftrightarrow t' \in {T_{sub_i}}^*$ holds.

$\underline{C_{sub_i} \text{ is } reqfalse}$: ${T_{sub_i}}^* = T_{sub_i}$. Let t' be a tuple from $CrossBase(T_{sub_i})$ that belongs to ${T_{sub_i}}^+$. The sub-condition $\mathcal{P}({T_{sub_i}}^+) =_n {T_{sub_i}}'$ is fulfilled only for such tuples. C_{sub_i} is false and from ${T_{sub_i}}^* = T_{sub_i}$ we know that ${C_{sub_i}}'(t')$ is false too. Thus the condition J_{sub_i} is fulfilled and it follows that ${C_{sub_i}}^+(t') \Leftrightarrow t' \in {T_{sub_i}}^*$ holds. □

3.5 Example of the Gen strategy

As an example for an application of the *Gen* strategy consider the query $q = \sigma_{a\ =\ ANY\ (\sigma_{c=b}(S))}(R)$ on relations R and S with schemas $\mathbf{R} = (a, b)$ and $\mathbf{S} = (c)$. The rewritten query produced by the rewrite rule *G1* is as follows:

$$\begin{aligned}q^+ =&\sigma_{C_{sub} \wedge {C_{sub}}^+}(\Pi_{a,b,a \to pa,b \to pb}(R) \times \\ &\Pi_{c \to pc}(S \cup null(S))) \\ C_{sub} =&(a\ =\ ANY\ (\sigma_{c=b}(S))) \\ {C_{sub}}^+ =& EXISTS\ (\sigma_{(a=c \vee \neg C_{sub}) \wedge pc =_n c'}(\\ &\Pi_{c,pc \to c'}(\sigma_{c=b}(\Pi_{c,c \to pc}(S)))) \\ &\vee (\neg\ EXISTS\ (\sigma_{c=b}(S)) \wedge pc =_n null)\end{aligned}$$

3.6 Rewrite Strategies for Uncorrelated Sublinks

The *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 ${T_{sub}}^+$ 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 *Gen* strategy.

3.6.1 Left Strategy

As mentioned above, uncorrelated sublinks enable us to directly join rewritten sublink queries. The *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_{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 $q = \sigma_{a\ =\ ALL\ (S)}(R)$. After applying rewrite rule *L1*, the resulting query q^+ is:

$$\begin{aligned}q^+ =& \Pi_{a,b,pa,pb,pc}(\\ &\sigma_{C_{sub}}(\Pi_{a,b,a \to pa,b \to pb}(R)) \rightarrowtail\!\!\bowtie_{C_{sub} \vee \neg C_{sub}'} \Pi_{c,c \to pc}(S)) \\ C_{sub} =& (a\ =\ ALL\ (S)) \\ C'_{sub} =& (a = c)\end{aligned}$$

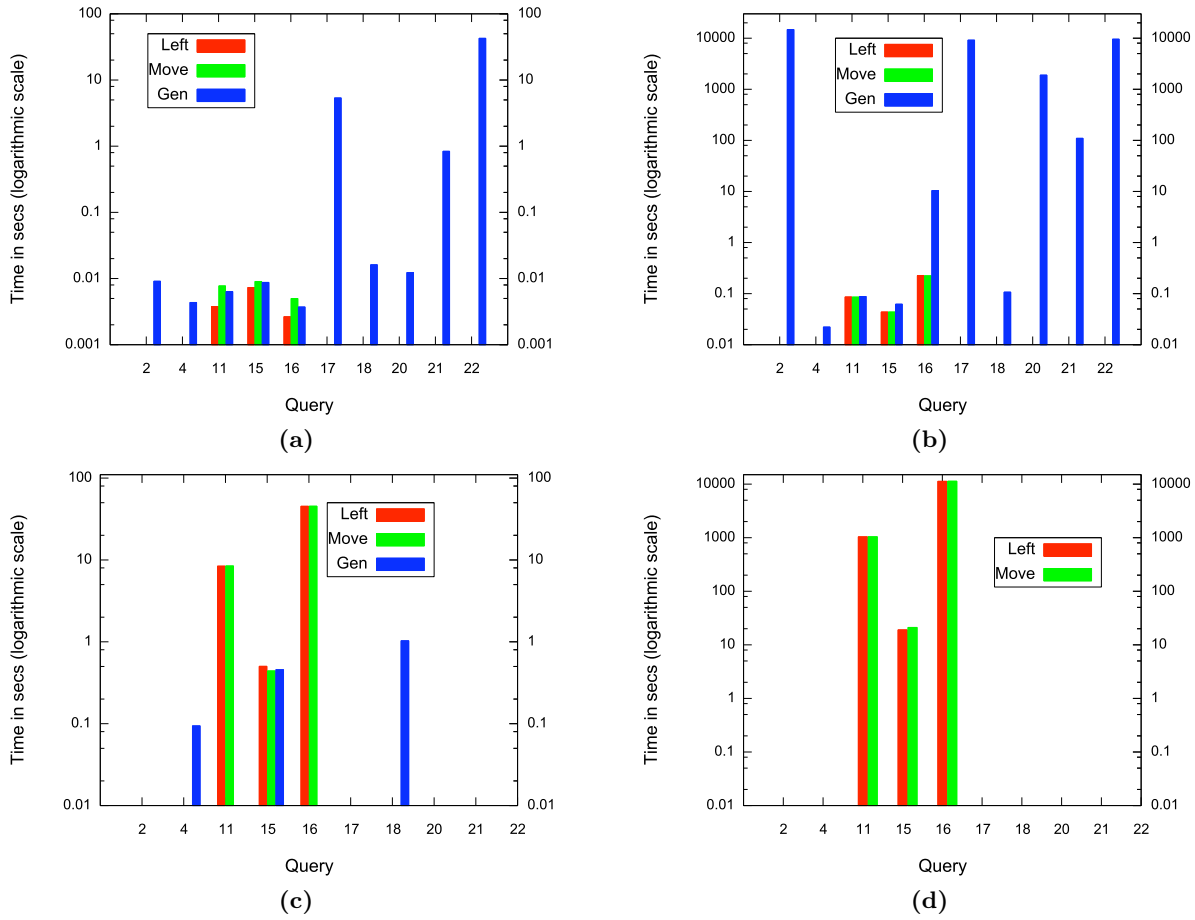

Figure 6: Performance of queries with sublinks in TPC-H for database sizes 1MB (a), 10MB (b), 100MB (c), and 1GB (d)

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 $T1$ 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:

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

Rewrite rule $T2$ 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 $q = \Pi_{a,S}(R)$:

$$q^+ = \Pi_{a,C_1,pa,pb,pc}(\Pi_{a,S\rightarrow C_1}(R) \rtimes_{true} (S))$$

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 $U1$ and $U2$ (see Figure 5) for specific types of sublinks. Rule $U1$ is a specific rewrite for a selection with a condition $C = \ EXISTS\ T_{sub}$. 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 $U2$ rewrites a selection with a condition that is a sin-

gle ANY-sublink with an equality comparison. The sublink condition is true if the comparison condition is true. Therefore, C_{sub} is always *reqtrue* 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]. *Perm* is realized as an extension of the PostgreSQL-DBMS [15]. The *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 *PROVENANCE* is used in the select clause to mark a query for provenance rewriting.

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

```
SELECT PROVENANCE *
  FROM R
  WHERE a = 3 AND b = ANY (SELECT * FROM S);
```

For a more detailed explanation of the SQL extension used by *Perm* and the adaptation of the rewrite rules for query trees the interested reader is refered 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 *Gen* strategy was used for all 9 queries with sublinks. For the queries with uncorrelated sublinks (11,15 and 16) the *Left* and *Move* strategy are also applied (none of the queries fulfills the requirements to apply the *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 *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 *Left* and *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 *Move* strategy showed no significant improvement in comparison to the *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.

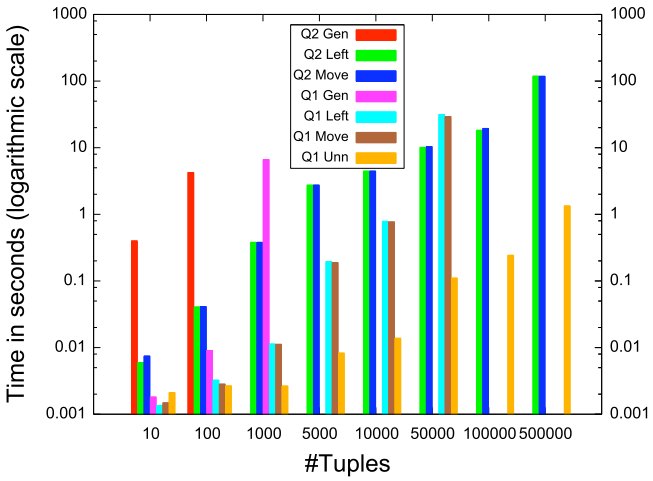

Figure 7: (Synthetic) Varying size of input relation

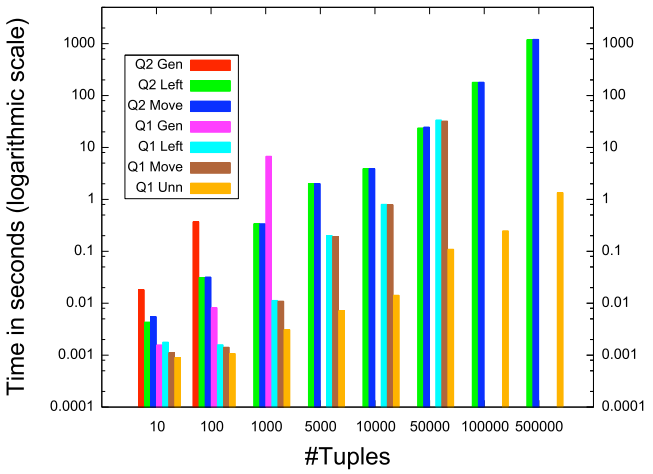

Figure 8: (Synthetic) Varying size of sublink relation

4.2.2 Synthetic Data

In addition to the TCP-H experiments, we also tested *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 ANY-sublink and equality

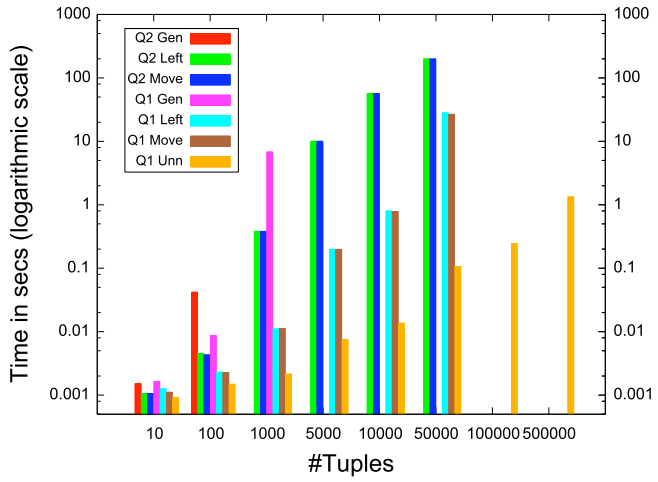

Figure 9: (Synthetic) Varying size of both relations

comparison:

$$q_1 = \sigma_{range \wedge a = ANY\ (\sigma_{range2}(R_2))}(R_1)$$

The *range* and *range*2 conditions restrict the input tables on a random range (with a fixed size) of values from attribute *b*. This query was used to test the performance of a simple sublink with an equality comparison.

The second query is an inequality condition in an *ALL*-sublink.

$$q_2 = \sigma_{range \wedge a < ALL\ (\sigma_{range2}(R_2))}(R_1)$$

Like for query $q1$, the *range*-conditions were used to select a random subset of the tuples from the input tables. All four rewrite strategies were used for both query types, except *Unn* that provides only a rewrite rule for query q_1.

We used the synthetic data to run three sets of experiments. For the first set of experiments we employed a fixed size of 1000 tuples for the relation used in the sublink and varied the size of the input relation of the selection. In the second set of experiments, we fixed the size of the input relation and varied the size of the sublink relation. For the last set of experiments the sizes of both relations were varied. All experiments were run for 100 queries of type q_1 and q_2. 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 to 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

[1] M. O. Akinde et al. Efficient computation of subqueries in complex OLAP. *ICDE '03*, pages 163–174, 2003.

[2] P. Buneman et al. Why and where: A characterization of data provenance. In *ICDT '01*, pages 316–330, 2001.

[3] P. Buneman et al. On the expressiveness of implicit provenance in query and update languages. *TODS*, 33(4), 2008.

[4] B. Cao et al. SQL query optimization through nested relational algebra. *TODS*, 32(3):18, 2007.

[5] S. Chaudhuri. An overview of query optimization in relational systems. *PODS' 98*, pages 34–43, 1998.

[6] L. Chiticariu et al. Dbnotes: a post-it system for relational databases based on provenance. In *SIGMOD '05*, pages 942–944, 2005.

[7] Y. Cui et al. Tracing the lineage of view data in a warehousing environment. *TODS*, 25(2):179–227, 2000.

[8] U. Dayal. Processing queries with quantifiers a horticultural approach. *PODS '83*, pages 125–136, 1983.

[9] M. Elhemali et al. Execution strategies for SQL subqueries. *SIGMOD '07*, pages 993–1004, 2007.

[10] F. Geerts et al. MONDRIAN: Annotating and querying databases through colors and blocks. Technical Report EDIINFRR0243, The University of Edinburgh, 2005.

[11] B. Glavic et al. Data provenance: A categorization of existing approaches. In *BTW '07*, pages 227–241, 2007.

[12] B. Glavic et al. Perm: Processing provenance and data on the same data model through query rewriting. In *ICDE '09*, 2009.

[13] T. Green et al. Provenance semirings. *PODS '07*, pages 31–40, 2007.

[14] W. Kim. On Optimizing an SQL-like Nested Query. *TODS*, 7(3):443–469, 1982.

[15] B. Momjian. *PostgreSQL: introduction and concepts*. Boston, MA: Addison-Wesley, 2001.

[16] M. Muralikrishna et al. Improved Unnesting Algorithms for Join Aggregate SQL Queries. *VLDB '92*, pages 91–102, 1992.

[17] M. Mutsuzaki et al. Trio-One: Layering uncertainty and lineage on a conventional DBMS. *CIDR '07*, pages 269–274, 2007.

[18] Y. L. Simmhan et al. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, 2005.

[19] W. Tan et al. Provenance in Databases: Past, Current, and Future. *IEEE Data Eng. Bull.*, 30(4):3–12, 2007.

[20] Transaction Processing Performance Council. TPC-H Benchmark Specification. *http://www. tpc. org/hspec. html*, 2008.