Department of Informatics, University of Zürich

Facharbeit in Informatik

Provenance in Temporal Databases

Ekaterina Kuleshova

Matrikelnummer: 08-748-253

Email: `ekaterina.kuleshova@uzh.ch`

October 12 , 2011

supervised by Prof. Dr. M. Böhlen and A. Dignös

University of Zurich

Department of Informatics

Contents

Abstract

The purpose of this paper is to develop tracing of lineage and provenance techniques for temporal databases. Using the snapshot reducibility property of temporal databases we will define a pointwise lineage traceability for temporal databases. Merging time points with the same lineage in the result of temporal operators allows an interval-based model by still allowing lineage traceability. On examples we show the algebra and it lineage. To trace lineage we need to materialize intermediate results. Moreover, lineage tells us only about the tuples that contribute and not how they contribute to the result query. That is why we further define relations annotated with provenance semirings. To be able to to perform queries on such relations we generalize the algebra to operate on them, so that query execution propagates provenance information. Finally, we define positive relational algebra, which propagates how-provenance.

1 Introduction

Conventional databases consider the data stored in it to be valid at time instant now, they do not keep track of past or future database states. By attaching a time period to the data, it becomes possible to store different database states (*timestamping*). Such a database is refered to as a *temporal database*. We assume that time is linear, ordered and discrete set of time points $p \in O$. Time is considered as being orthogonal to data, hence one can take a slice of time-axis at the fixed time/ particular state. The obtained Database is called *snapshot database* and one of the important requirements temporal DB must satisfy is the equivalence between the snapshot database and the non temporal database at this time point. This requirement is called *snapshot reducibility*. Temporal DBMS must also satisfy some other requirements, that not only take migration concepts into consideration, but also ensure systematic and comprehensive built-in support: *temporal upward compatibility, syntactically similar, interval preservation, non-restrictiveness* (for more details see [Böh]).

Data provenance is an account of the derivation of a peace of data in a dataset that is typically the result of executing a database query against a source database [Bun]. A special field of data provenance is called *lineage tracing*. Lineage is a way of relating the tuples in query output to the tuples in the query input that "contribute" to them [Tan].

Typically, lineage traceability (and the provenance we consider) is obtained by carefully reasoning about the algebraic form of the database query and the underlying data model of the source and resulting databases. The formal description of how a relational database operates, i.e the mathematics which underpin SQL operations is defined by *relational algebra*. Using relational algebra we can hence formulate the purpose of lineage traceability and solve the problem in terms of the set of relation tuples (base tuples) that produce a given result. These base tuples are called the *derivation or tracing of lineage* of the result tuple. The problem of lineage tracing is defined and solved for non-temporal databases [Cui].

In the *first part* of the paper problem of data lineage tracing for temporal databases will be solved and the lineage preserving set based temporal relational algebra is defined. At the end of the first part we will see two drawbacks of this approach. First, lineage tracing forces intermediate results to be saved and that can be very expensive. Second, the possibility of tracing data lineage doesn't provide any information about how the result was got. That is why in the *second part* we will work with the annotation of the results of database transformations with provenance information (*annotated relations or K-relations*). V. Tan, G. Karvounarakis and T. Green in their paper "Provenance Semirings" have proposed a framework of semirings annotation [Tan]. The idea is that every tuple of the database is associated with an element of a semiring K, and to propagate the annotations through query evaluation. This means that query construct (of some expressivness) must be associated with operations in the semiring. The laws of commutative semirings are forced by certain expected identities in RA. The propagation of provenance through query evaluation is defined using only the semiring operations addition and multiplication (and the constants 0 and 1). Having identified commutative semirings as the right algebraic structure, they argue that a symbolic representation of semiring calculations is just what is needed to record, document, and track RA querying from input to output for applications which require rich *provenance* information. That is why provenance is represented by elements of a semiring of polynomials. Using these achievements, the why-provenance for positive algebra is defined.

The contributions of this paper are as follows:

- Using the snapshot reducibility property, we get a lineage for the temporal databases and define lineage preserving set-based temporal relational algebra.
- Using the theory of semirings provenance, the positive temporal relational algebra which includes annotation propagation will be defined.

2 Definitions and Notation

Temporal databases were defined above. Now we take a deeper look at time dimension. There are mainly two different notions of time which are relevant for temporal databases. One is called *valid time*, the other one is *transaction time*. Valid time denotes the time period during which a fact is true with respect to the real world. Transaction time is the time period during which a fact is stored in the database. In case of the relational model, data and time are recorded in a relation. Ordinary attributes of a conventional relation are termed *explicit*. Time interval itself is not considered to be data, it is considered as being orthogonal to data. Hence, this type of attributes often termed *implicit attributes* and are of the form $[begin, end)$. It is said that the data recorded in the explicit attributes is *stamped* by the time recorded in the implicit attributes. A relation which contains explicit data part, a valid time T or/and a transactional time is called *temporal relation*. Next we will only consider *valid time*.

To transform temporal relations we use *temporal relational algebra*. In terminology of temporal relational algebra:

- *Relation* is a set of tuples,
- *Tuple* is a collection of attributes which describe some real world entity and period during which this entity is valid.
- *Attribute* is a real world role played by a named domain
- *Domain* is a set of atomic values
- *Temporal set* has no value equivalent over overlapping time stamps (periods). Each snapshot is a set (see later).

In the Figure 2.1 you can see an example of a *temporal relation*. The relation "HotelBooking" is a set of 3 tuples. The first tuple (Knoth, 1) is valid on the time Interval [1,5). The time interval in the *Figure 2.1* is represented by horizontal line.

Using relational algebra we can hence formulate the problem of data lineage and data provenance and solve the problem in terms of the set of relation tuples (base tuples) that produce a given result. These base tuples in data lineage are called the *derivation or lineage* of the result tuple. I first review relational table semantics. Tuple derivations for operators and results follows in the next section.

We assume that a table (relation) *R* contains a set of tuples $t_1, ..., t_n$. A database instance *I* contains a list of *base tables* $< R_1, ..., R_m >$. *V* is a result of a query over the base tables in *I*. The query (or mapping from the base tables to the result table) is called the *result definition*, denoted by *v*. We say that $R_1, ..., R_m$ *derives V* if $V = v(R_1, ..., R_m)$. We consider *set sematics*, i.e. no dublicates.

HotelBooking

client	hotel_id	period
Knoth	1	1-5
Smith	9	3-6
Eger	3	5-7

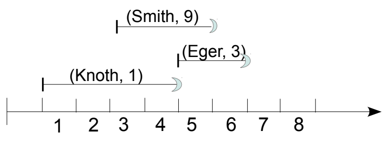


Figure 2.1: Example of *temporal relation*

In this report I work with a class of queries defined over base relations using the relational algebra operators *selection (σ), projection* (π)*, aggregation*(θ)*, difference* $(-)$*, union* $(\cup)$ *and cartesion product* $(\times)$ (for more details see for example [Cui]). We use the standard relational semantics, included here for completeness. For an attribute list $A = A_1, ..., A_n$, we use shorthand t.A to denote $< t.A_1, ..., t.A_n >$.

3 Lineage traceability for temporal data

3.1 Tuple derivation in nontemporal data model

Definition 3.1 (*Tuple derivation for an Operator*). Let Op be any relational operator over tables $R_1, ..., R_m$, and let $R = Op(R_1, ..., R_m)$ be the result of applying Op to $R_1, ..., R_m$. Given a tuple $t \in R$, we define *t's derivation in* $R_1, ..., R_m$ *according to Op* to be $Op^{-1}_{<R_1,...,R_m>}(t) =< R^*_1, ..., R^*_m >$, where $R^*_1, ..., R^*_m$ are *maximal* subsets of $R_1, ..., R_m$ such that:

(a) $Op(R^*_1, ..., R^*_m) = \{t\}$

(b) $\forall R^*_i : \forall t^* \in R^*_i : Op(R^*_1, ..., \{t^*\}, ..., R^*_m) \neq \emptyset$

We also say that $Op^{-1}_{R_i}(t) = R^*_i$ is *t's derivation in* R_i, and each tuple t^* in R^*_i *contributes to t*, for i = 1...m.

Requirement (a) says that the derivation tuple sets (the R^*_i's) derive exactly *t*. From relational semantics we know that for any result tuple *t* there must exist such tuple sets. Requirement (b) says that each tuple in the derivation does in fact contribute something to *t*. By defining the R^*_i's to be the maximal subsets that satisfy requirements (a) and (b), we make sure that the derivation contains exactly all the tuples that contribute to *t*. Thus, the derivation fully explains why a tuple exists in the result (see also [Cui]).

In the non temporal model Op can be extended to represent the derivation of a set of tuples: $Op^{-1}_{<R_1,..,R_m>}(R) = \bigcup_{t \in R} Op^{-1}_{<R_1,...,R_m>}(t)$ where $\bigcup$ represents the multiway union of relation lists, i.e. $< S_1, ..., S_m >=< (R_1 \cup S_1), ..., (R_m \cup S_m) >$. It can be shown that there is a unique derivation for any operator and result tuple.

3.2 Tuple derivation for a snapshot in temporal model

According to the snapshot reducibility property a snapshot database we get by slicing a temporal database on the time-axis is equal to the non temporal database at the given time point.

Lemma 3.1 Let τ_p be a *time slice operator* at the time point p. Then it holds true:
$\forall p \in O \; \tau_p(Op^T(R_1, ..., R_m)) = Op(\tau_p(R_1), ..., \tau_p(R_m))$

, where $R_1, .., R_m$ are relations (temporal if a temporal operator will be executed and non temporal relation in the case of non temporal operator); Op^T is a temporal Operator, Op is non temporal operator.

Remark Time slice operator selects all tuples in the argument relations with the timestamp that overlaps time point p.

The above Lemma means that for all time points in the time domain the execution of a temporal operator on the temporal relation gives the same result as first take a snapshot and then execute a non temporal Operator on the snapshot.

Definition 3.2 Let Op^T be a basic operator of temporal relational algebra over temporal relations $R_1, ..., R_m$ and $R = Op^T(R_1, ..., R_m)$. Similar to the Lemma 1, τ_p be a *time slice operator* at the time point p. For each tuple $z \in R$ valid at time point p we define the lineage set $L_{\{z,p\}} =< R_1^*, ..., R_m^* >$ of the operator Op^T as follows: $L_{\{z,p\}} = (Op^T_{<R_1,...,R_m>})^{-1}(\tau_p(\{z\}))$ iff:

(a) $\tau_p(Op^T(L_{\{z,p\}})) = \tau_p(\{z\})$

(b) $\forall R_i^* \, \forall z_{p,i} \in R_i^* : \tau_p(Op^T(R_1^*, ..., z_{p,i}, R_m^*)) \neq \emptyset$

(c) $L_{\{z,p\}} =< R_1^*, ..., R_m^* >$ is a maximal subset of $R_1, ..., R_m$

Remark The first two conditions in Theorem 1 can be written as follows:

(a) $\tau_p(Op^T(L_{\{z,p\}})) = \tau_p(Op^T(R_1^*, ..., R_m^*)) =$
$\overset{Lemma1}{=} Op(\tau_p(R_1^*), ..., \tau_p(R_m^*)) = \{z_p\}$

(b) $\forall R_i^* \, \forall z_{p,i} \in R_i^* : \tau_p(Op^T(R_1^*, ..., z_{p,i}, R_m^*))$
$= Op(\tau_p(R_1^*), ..., \tau_p(z_i), ..., \tau_p(R_m^*))) \neq \emptyset$

That means, that pointwise tuple derivation for an temporal Operator is equivalent to the tuple derivation for a non temporal operator on the snapshot database.

Lemma 3.2 (*Point based lineage for temporal databases*)
Let R and S be temporal relations, A and B are Attributes in the Relation R and S respectively. For difference and Union operators A and B are union compatible. c is a predicate over Attributes in R. p is a time point, z is a tuple s.t $z \in Op^T(R)$ or $z \in Op^T(R, S)$, where Op^T is a basic temporal operator. G is a group by attribute list from A. H is an attribute list from A. $aggr(H)$ is a set of aggregate functions applied to attributes H of R.

1. Selection:
   $(\sigma^T_{c<R>})^{-1}(z, p) = \{t | t \in R \wedge t.A = z.A \wedge p \in t.T\}$

2. Projection:
   $(\pi^T_{H<R>})^{-1}(z, p) = \{t | t \in R \wedge t.H = z.H \wedge p \in t.T\}$

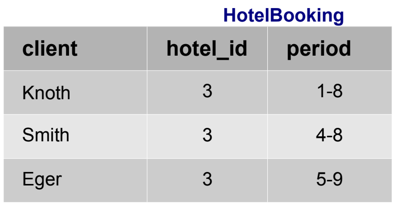
3. Aggregation:
$({}_G\theta^T_{aggr(H)<R>})^{-1}(z,p) = \{t | t \in R \wedge t.G = z.G \wedge p \in t.T\}$

4. Difference:
$(R -^T S)^{-1}(z,p) =$
$$\langle \{t|t \in R \wedge t.A = z.A \wedge p \in t.T\}, \{t|t \in S \wedge t.B \neq z.A \vee p \notin t.T\}\rangle$$

5. Union:
$(R \cup^T S)^{-1}(z,p) =$
$$\langle \{t \mid t \in R \wedge t.A = z.A \wedge p \in t.T\}, \{t \mid t \in S \wedge t.B = z.A \wedge p \in t.T\}\rangle$$

6. Cartesian Product:
$(R \times^T S)^{-1}(z,p) =$
$$\langle \{t|t \in R \wedge t.A = z.A \wedge p \in t.T\}, \{t|t \in S \wedge t.B = z.B \wedge p \in t.T\}\rangle$$

Thus, we got a pointwise lineage tracing for temporal databases. The next natural step to get the lineage on the intervals is *to make a coalescing of points with the same lineage into maximal intervals*. We consider table *HotelBooking* in the *Figure 3.1* and execute temporal projection $\pi^T_{hotel\_id}(HotelBooking)$.


HotelBooking

client	hotel_id	period
Knoth	3	1-8
Smith	3	4-8
Eger	3	5-9

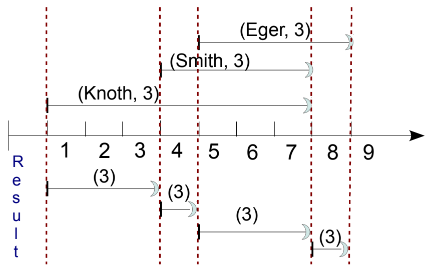


Figure 3.1: Lineage traceability for temporal projection: $\pi^T_{hotel\_id}(HotelBooking)$

Taking lineage traceability pointwise we get following results:
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\},1) =< \{(Knoth,3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\},2) =< \{(Knoth,3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\},3) =< \{(Knoth,3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\},4) =< \{(Knoth,3),(Smith,3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\},5) =< \{(Knoth,3),(Smith,3),(Eger,3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\},6) =< \{(Knoth,3),(Smith,3),(Eger,3)\} >$

$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\}, 7) =< \{(Knoth, 3), (Smith, 3), (Eger, 3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3\}, 8) =< \{(Eger, 3)\} >$

Now we can coalesce points with the same lineage to intervals:
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3|[1-4)\}) =< \{(Knoth, 3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3|[4-5)\}) =< \{(Knoth, 3), (Smith, 3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3|[5-8)\}) =< \{(Knoth, 3), (Smith, 3), (Eger, 3)\} >$
$(\pi^T_{hotel\_id<HotelBooking>})^{-1}(\{3|[8-9)\}) =< \{(Eger, 3)\} >$

We we can now define a lineage preserving temporal relational algebra.

Definition 3.3 (*Lineage preserving set based temporal algebra*)
Let R and S be a temporal relations, A and B are Attributes in the Relation R and S respectively. For difference and Union operators A and B are union compatible. c is a predicate over Attributes in R. p is a time point, z is a tuple s.t $z \in Op^T(R)$ or $z \in Op^T(R, S)$, where Op^T is a basic temporal operator. G is a group by attribute list from A. H is an attribute list from A. $aggr(A)$ is a set of aggregate functions applied to attributes A of R. t.T is a maximal time interval over which all argument tuples with the A-values are hold. Small letters like k, x, t, r, s mean tuples.

1. $\sigma^T_c(R) = \{t | t \in R \wedge c(t)\}$,

2. $\pi^T_H(R) = \{t|$
   $\exists r \in R(t.H = k.H \wedge t.T \subseteq r.T) \wedge$
   $\forall x \in R(x.H = t.H \Rightarrow x.T \supseteq t.T \vee x.T \cap t.T \neq \emptyset) \wedge$
   $\forall T' \supset t.T \exists x \in R(x.H = t.H \wedge T' \nsubseteq x.T \wedge T' \cap x.T \neq \emptyset)\}$,

3. ${}_G\theta^T_{aggr(H)}(R) = \{t|$
   $\exists r \in R(t.G = r.G \wedge t.T \subseteq r.T \wedge t.aggr(H) = g.aggr(H)) \wedge$
   $g = \{x | x \in R \wedge x.G = t.G \wedge x.T \cap t.T \neq \emptyset\} \wedge$
   $\forall x \in R(x.G = t.G \Rightarrow r.T \supseteq t.T \vee x.T \cap t.T \neq \emptyset) \wedge$
   $\forall T' \supset t.T \exists x \in R(x.G = t.G \wedge T \nsubseteq x.T \wedge T' \cap t.T \neq \emptyset)\}$

4. $R -^T S = \{t|\}$
   $\exists r \in R(t.A = k.A \wedge t.T \subseteq k.T \wedge$
   $\forall s \in S(s.B = t.A \Rightarrow s.T \cap t.T = \emptyset) \wedge$
   $\forall T' \supset t.T \exists s \in S(s.B = t.A \wedge T' \cap s.T \neq \emptyset \vee T' \nsubseteq r.T))\}$

5. $R \cup^T S = \{t|$
   $\exists r \in R(t.A = r.A \wedge t.T = r.T \wedge$
      $\forall s \in S(s.B = t.A \Rightarrow s.T \cap t.T = \emptyset) \wedge$
      $\forall T' \supset t.T \exists s \in S(s.B = t.A \wedge T' \cap s.T \neq \emptyset \vee T' \nsubseteq r.T)) \vee$
   $\exists r \in R(t.A = r.A \wedge$
      $\exists s \in S(r.A = s.B \wedge t.T = r.T \cap t.T \wedge t.T \neq \emptyset)) \vee$
   $\exists s \in S(t.A = s.B \wedge t.T \subseteq s.T) \wedge$

$$\forall r \in R(r.A = t.A \Rightarrow r.T \cap t.T = \emptyset) \wedge$$
$$\forall T' \supset t.T \exists s \in S(s.B = t.A \wedge T' \cap s.T \neq \vee T' \nsubseteq r.T))\}$$

6. $R \times^T S = \{t|$
$$\exists r \in R \exists s \in S(t.A = r.A \wedge t.B = s.B \wedge t.T = r.T \cap s.T \wedge t.T \neq \emptyset)\}$$

3.3 Example

We will now show results of some lineage preserving temporal operators execution. The following temporal tables represent a part of temporal relational DB for a travel agency: "Car-Sharing" and "HotelBooking" (See Figure 3.2). We want to compute the temporal unary operator projection and a binary operator cartesian product:

- $V_1 = \pi^T_{client}(CarSharing)$,
- $V_2 = \pi^T_{client}(HotelBooking)$,
- $V = V_1 \times^T V_2$

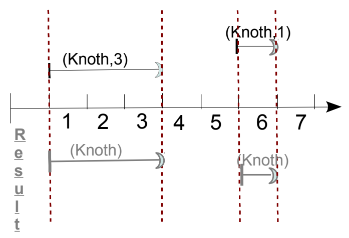
On the *Figure 3.2* you can see the calculation of a projection on the Attribute client for temporal relations HotelBooking and CarSharing for a *client= "Knoth"*. For other clients projection can be calculated analogously. In the *Table 3.1 and 3.2* you can see the result of applying the temporal projection on the tables "CarSharing" and "HotelBooking" respectively.

CarSharing

client	car_id	period
Knoth	1	6-7
Smith	5	4-5
Knoth	3	1-4
Smith	9	2-3

HotelBooking

client	hotel_id	period
Knoth	3	5-10
Schweizer	4	3-6
Knoth	4	1-3
Schweizer	3	1-2
Knoth	5	3-11



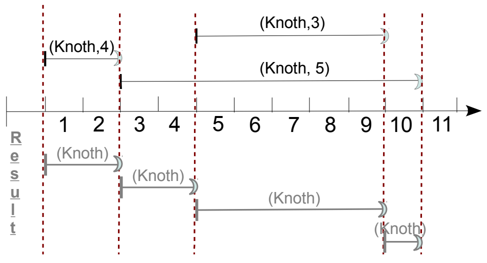


Figure 3.2: $\pi^T_{client}(HotelBooking)$, $\pi^T_{client}(CarSharing)$ for $client = "Knoth"$

The result of the projection of the relation $HotelBooking$ on the Attribute $client$ for $client = "Knoth"$ contains for example tuples $(Knoth|[1-3))$ and $(Knoth|[3-5))$ (see the right part on the Figure 3.2). We could coalesce these two result tuple into $(Knoth|[1-5))$. According the *Definition 3.3* we don't coalesce these two result tuple since the trace of data lineage for these two tuples is different ($(Knoth, 4|[1-3))$ for the result tuple $(Knoth|[1-3))$ and $(Knoth, 5|[3-11))$ for the result tuple $(Knoth|[3-5))$),

Table 3.1: V1

	client	period
1	Knoth	1-4
2	Knoth	6-7
4	Smith	2-3
5	Smith	4-5

Table 3.2: V2

	client	period
1	Knoth	1-3
2	Knoth	3-5
3	Knoth	5-10
4	Knoth	10-11
5	Schweizer	1-2
6	Schweizer	3-6

Figure 3.3 shows the result of applying the temporal Cartesian Product.

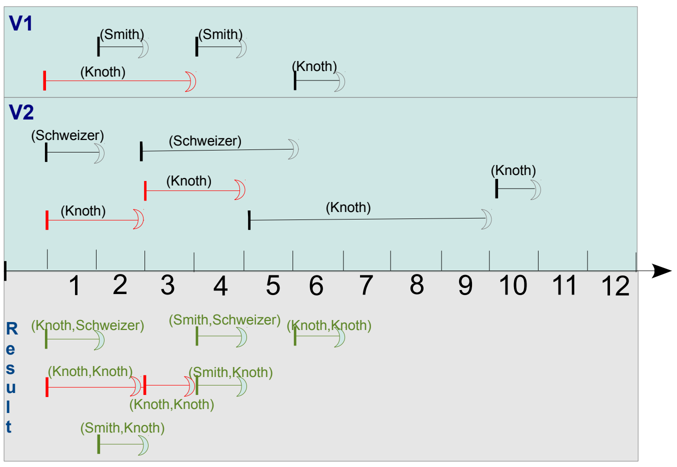


Figure 3.3: $V_1 \times^T V_2$

The result of applying Temporal Cartesian Product $V_1 \times V_2$ is represented in the *Table 3.3*. If we look at the red tuples in V_1 and V_2, the result of applying temporal Cartesian product consists of 2 equal tuples: $(Knoth, Knoth|[1-3))$ and $(Knoth, Knoth|[3-4))$. We could coalesce these two result tuples to $(Knoth, Knoth|[1-4))$. According to *Definition 3.3* we could do it only if they had the same lineage and this is not the case here (the lineage of $(Knoth, Knoth|[1-3))$ is $\langle(Knoth|[1-4)),(Knoth|[1-3))\rangle$ and the lineage of $(Knoth, Knoth|[3-4))$ is $\langle(Knoth|[1-4)),(Knoth|[3-4))\rangle$). You can see above that $(Knoth|[1-3)$ and $(Knoth|[3-4)$ have different trace of lineage and that is why are also written as two different tuples.

Table 3.3: $V_1 \times^T V_2$

	$client_{V_1}$	$client_{V_2}$	period
1	Knoth	Schweizer	1-2
2	Knoth	Knoth	1-3
3	Knoth	Knoth	3-4
4	Smith	Knoth	2-3
5	Smith	Schweizer	4-5
6	Smith	Knoth	4-5
7	Knoth	Knoth	6-7

3.4 Lineage on composite temporal queries

For complex queries that are composed of a sequence of more than one relational operator, lineage is defined inductively. A composite temporal query can be thought of as an operator tree that is evaluated buttom-up. Intuitively, if a base tuple t^* contributes to a tuple t' in the intermediate result of a composite query evaluation, and t' further contributes to a view tuple t, then t^* contributes to t.

Definition 3.4. We define a *composite query tuple's derivation* to be the set of all base tuples that contribute to the result tuple.

The *Definition 3.4* means that if we for example have a composite query:

$$\sigma_c(\pi_B(R)), \text{ i.e} \begin{cases} V_1 = \pi_B(R) \\ V_2 = \sigma_c(V_1) \end{cases},$$

then the *trace of lineage* consists not of the tuples in V_1, but rather of tuples from the temporal relation R.

A straightforward way to compute *tuple derivation for composite temporal queries* is thus to compute the intermediate results for all operators, store the results as temporary tables, then trace the tuple's derivation in the temporary tables recursively until reaching the base table.

Example 3.4.1 (*Composite query with lineage traceability direct from result*).
Let us consider a composite query from *Example 3.3*:

$$V = V_1 \times^T V_2, \text{ where} \begin{cases} V_1 = \pi^T_{client}(CarSharing) \\ V_2 = \pi^T_{client}(HotelBooking) \end{cases}$$

We *trace the lineage* recursively for the result relation, represented in the *Table 3.4*. The *intermediate results* are materialized in *Table 3.1* and *Table 3.2* for V_1 and V_2 respectively. We first trace the lineage from the result *Table 3.4* to the intermediate result in the *Table 3.1* and *3.2*.

Table 3.4: $V_1 \times^T V_2$

	$client_{V_1}$	$client_{V_2}$	period
1	Knoth	Schweizer	1-2
2	Knoth	Knoth	1-3
3	Knoth	Knoth	3-4

We can do it using the *Figure 3.3*. On the figure we can see that trace of lineage for the tuple:

$$(Knoth, Knoth|[1-3))$$

is the set of tuples from V1 and V2:

$$\{(Knoth|[1-4)), (Knoth|[1-3))\}$$

and knowing these intermediate lineage result we can compute the lineage for this tuple using *Figure 3.2*:

$$\{(Knoth, 3|[1-4)), (Knoth, 4|[1-3))\}$$

In the same way we get *trace of lineage* for other tuples in the result relation in the *Table 3.4*. The *intermediate lineage* is the following:

$$\begin{array}{l}\langle\{(Knoth|[1-4)), (Schweizer|[1-2))\},\\ \{(Knoth|[1-4)), (Knoth|[1-3))\},\\ \{(Knoth|[1-4)), (Knoth|[3-4))\}\rangle.\end{array}$$

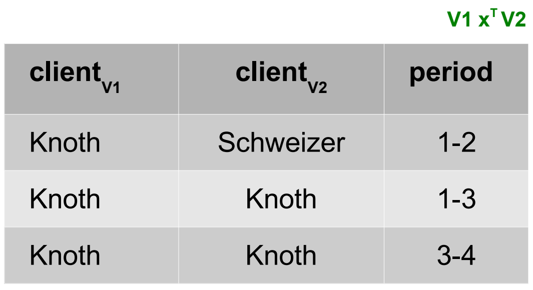
And the final **trace of lineage** for the result represented in the *Table 3.4* is given by:

$$\begin{array}{l}\langle\{(Knoth, 3|[1-4)), (Schweizer, 3|[1-2))\},\\ \{(Knoth, 3|[1-4)), (Knoth, 4|[1-3))\},\\ \{(Knoth, 3|[1-4)), (Knoth, 5|[3-11))\}\rangle.\end{array}$$

In this example there is also a direct way to trace the lineage of data, i.e. direct from result table without the intermediate results. We just need to look for tuples in the base relations $HotelBooking$ and $CarSharing$, which attributes are equal to attributes in the result tuple and the time interval contains or equal the time interval of the result tuple (see *Figure 3.4*).

Example 3.4.2 (*Composite query which requires storage of intermediate results to trace the lineage*).
Now we want to derive lineage for the result table, given by the *relation*:

$$< \{Knoth|[1-3)\}, \{Knoth|[3-11)\} >$$



V1 x^T V2

client$_{V1}$	client$_{V2}$	period
Knoth	Schweizer	1-2
Knoth	Knoth	1-3
Knoth	Knoth	3-4

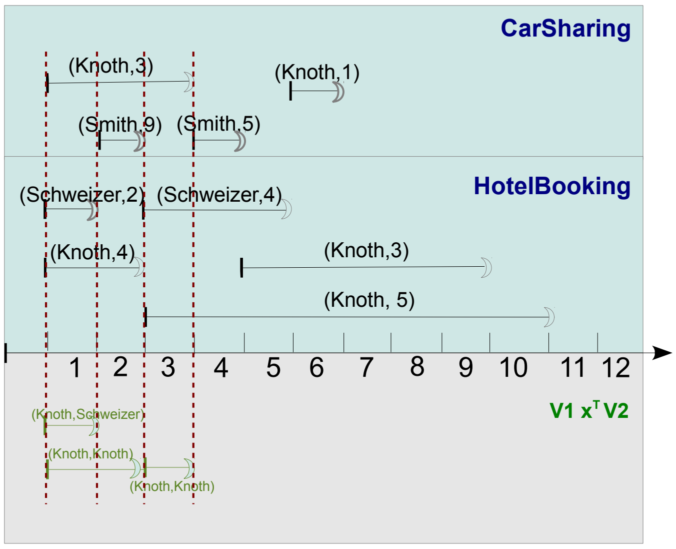


Figure 3.4: $V_1 \times^T V_2$

We know that this result we got applying following operators of temporal algebra on the temporal relation:

$$\pi_{client}(\sigma^T_{hotel\_id>3,client="Knoth"}(HotelBooking)),$$

where HotelBooking is a temporal relation given on the *Figure 3.2*. But we do not have any intermediate results. If we try to get the lineage direct from the result, we get the following:

	client	*hotel_id*	period
1	Knoth	3	5-10
2	Knoth	4	1-3
3	Knoth	5	3-11

To get this result (like in example above) we looked for tuples in the base relation *HotelBooking*, which attributes are equal to attributes in the result tuple and the time interval contains or equal the time interval of the result tuple.

Now, if we derive tuples which contribute to result table using intermediate result, we get:

	$client$	$hotel\_id$	period
1	Knoth	4	1-3
2	Knoth	5	3-11

Thus, we can conclude that the existence of the operator *selection* makes lineage derivation on the composite queries direct from the result (without materializing of intermediate results) impossible.

Lemma 3.3 For composite queries involving only θ^T, π^T, $\times^T$, $\cup^T$ we do not need the intermediate results to trace the lineage.

Remark Since temporal operators int he *Lemma 3.3* doesn't exclude any base tuples, one can conclude that the composite queries based on that operators are traceable.

Trace of lineage is *impractical* due to the computation and storage required for all the intermediate results which can be extremely expensive. That is why we are interested in a *question* of lineage traceability for composite temporal queries in the case when we only have *base tables, operators tree and result table*, i.e. where we do not need to materialize the intermediate results.

4 Provenance Semirings

As mentioned, storage or recompilation of the intermediate results can be expensive. The alternative way to get lineage is to use annotated relations to store provenance information. To be able to perform queries on annotated relations we need to extend relational algebra to operate on these, so that the operators execution propagates provenance information.

A.Tannel etc. el. proposed a general data model (referred to as *K-relations*) for annotated relations [Tan]. The generalization of *positive relational algebra* to perform on *K-relations* was introduced. It turned out that basic temporal operators performed on "usual" relations can be naturally extended to operations on annotated relations. More specifically, operations on tuples can be naturally translated into the algebraic operations sum and $product$ in K. Consideration of properties in *positive* RA like associativity and commutativity for operator $union$ and the associative, commutative and the distribution property of $join$ forces $(K, +, \cdot, 0, 1)$ to be a *semiring*. This led to the definition of the positive algebra on *K-relations*. Then the autors of the paper [Tan] noticed that symbolic representation of semirings calculations is just what is needed to record, document, and track *RA* querying from input to output for applications which require rich *provenance* information. Thus, the provenance is represented by elements of a *semiring of polynomials*. In the next section key definitions and results from the paper [Tan] are repeated.

4.1 Positive algebra for K-relations and provenance semirings

Definition 4.1 In relational model a *tuple* can be considered as functions $t : U \to \mathbb{D}$ with U a finite set of attributes and $\mathbb{D}$ a domain of values.

We fix the time being and we denote the set of all such U-tuples by $\mathcal{U}$*-tup*.

Definition 4.2 A *relation* over *U* is a subset of $\mathcal{U}$*-tup*.

We model a tagged-relation by a function on all possible tuples. For example a usual set-theoretic relation corresponds to a function R : $\mathcal{U}$*-tup* $\to \mathbb{B}$, where $\mathbb{B}$ = *{true, false}*. That means tuples in the relation tagged by *true* and those not in a relation by *false*. This coincides with the definition of a relation as a subset of tuples.

Definition 4.3 Let K be a set containing a distinguished element 0. A *K-relation* over a finite set of attributes $\mathcal{U}$ is a function R : $\mathcal{U}$*-tup* $\to \mathcal{K}$ such that its *support* defined by

$supp(\mathcal{R}) \xlongequal{def} \{t|R(t) \neq 0\}$ is finite.

Thus, we have a space of annotations *K* and *K-relations* are relations where each tuple is annotated with some element from K. Now we need to define some operations on K in order to be able to perform queries that "annotate". We will now see that operations that queries in the relational algebra perform on tuples can be naturally extended to operations on annotated tuples.

To deal with *selection* we assume that the set K contains two distinct values $0 \neq 1$ which denote "out of" and "in" the relations, respectively. To deal with $union$ and $projection$ and therefore to combine different tags of the same tuple into one tag we assume that K is equipped with a binary operation "+" (*alternative use of data*). To deal with natural $join$ and therefore combine tags of joinable tuples we assume that K is equippped with another binary operation "·" (*joint use of data*).

Definition 4.4 Let $(K, +, \cdot, 0, 1)$ be an algebraic structure with two distinguished elements. The operations of the *positive algebra* are defined as follows:

empty relation For any set of attributes *U*, there is $: \mathcal{U} - tup \to K$ such that $\emptyset(t) = 0$.

union If R : $\mathcal{U}$*-tup* $\to \mathcal{K}$ then $R_1 \cup R_2$: $\mathcal{U}$*-tup* $\to \mathcal{K}$ is defined by

$$(R_1 \cup R_2)(t) \xlongequal{def} R_1(t) + R_2(t)$$

projection If R : $\mathcal{U}$*-tup* $\to \mathcal{K}$ and $V \subseteq U$ then $\pi_V R$: $\mathcal{U}$*-tup* $\to \mathcal{K}$ is defined by

$$(\pi_V R)(t) \xlongequal{def} \sum_{t=t' \text{ on } V \text{ and } R(t')\neq 0} R(t')$$

(here $t = t'$ on V means t' is a $\mathcal{U}$-tup whose restriction to V is the same as the V-tuple t; note also that the sum is finite since *R* has finite support).

selection If R : $\mathcal{U}$*-tup* $\to \mathcal{K}$ the selection predicate **P** maps each *U-tuple* to either 0 or 1 then $\sigma_{\mathbf{P}} R$: $\mathcal{U}$*-tup* $\to \mathcal{K}$ is defined by

$$(\sigma_{\mathbf{P}} R)(t) \xlongequal{def} R(t) \cdot \mathbf{P}(t)$$

Which {0,1}-valued functions are used as selection predicates is left unspecified, except that we assume that *false*-the constantly 0 predicate, and *true*- the constantly 1 predicate, are always available.

natural joint If R_i : $\mathcal{U}_i$*-tup* $\to \mathcal{K}$ *i=1,2*, then and $R_1 \bowtie R_2$ is the *K-relation* over $U_1 \cup U_2$ defined by

$$(R_1 \bowtie R_2)(t) \xlongequal{def} R_1(t_1) \cdot R_2(t_2)$$

where $t_1 = t$ on U_1 and $t_2 = t$ on U_2 (recall that *t* is a $U_1 \cup U_2$ *-tuple*).

renaming If $R : \mathcal{U}\textit{-tup} \to \mathcal{K}$ and $\beta : U \to U'$ is a bijection then $\rho_\beta R$ is a *K-relation* over U' defined by

$$(\rho_\beta R)(t) \stackrel{def}{=\!=} R(t \circ \beta)$$

Remark. Join is equal to Cartesian product followed by Selection:

$$r \bowtie_c s = \sigma_c(r \times s)$$

Lemma 4.1 The operations of RA^+ preserve the finiteness of supports, therefore they map *K-relations* to *K-relations*. Hence, *Definition 4.1* gives us an algebra on *K-relations*.

Lemma 4.2 The following *RA* identities:

- *union* is associative, commutative and has identity
- *join* is associative, commutative and distribute over *union*
- *projections* and *selections* commute with each other as well as with unions and joins (when applicable)
- $\sigma_{false}(R) =$ and $\sigma_{true}(R) = R$

hold for positive algebra on *K-relations* if and only if $(K, +, \cdot, 0, 1)$ is a *commutative semiring*.

Definition 4.5 Let *X* be the set of tuple ids of a (usual) database instance *I*. The *positive algebra provenance semirings* for *I* is the semiring of polynomials with variables from *X* and coefficients from $\mathbb{N}$, with the operations defined as usual: $(\mathbb{N}[X], +, \cdot, 0, 1)$.

4.2 Trace of Lineage/Where-provenance

We show that in Chapter 3 considered *trace of lineage* can be now modeled propagating lineage using annotated relations. As you already know *Lineage/where-provenance* is defined as a way of relating the tuples in a query output to the tuples in the query input that "contribute" to them. The *where-provenance* of a tuple t in a query output in fact the *set* of all contributing input tuples. Propagations of *where-provenance* for queries in RA^+can be defined using *Definition 4.4* for the semiring:

$$(\mathcal{P}(X), \cup, \cup, \emptyset, \emptyset),$$

where X consists of the ids of the tuples in the input instance.

Example 4.2.1 Let us consider the same temporal relations $CarSharing$ and $HotelBooking$ as in the *Example 3.3* but now we tag each tuple in that relations with their own ids $c_1, c_2, c_3, c_4, h_1, h_2, h_3, h_4, h_5$, as shown in *Table 4.1* and *Table 4.2*.

Table 4.1: $CarSharing$

$client$	car_id	$period$	
Knoth	1	6-7	c_1
Smith	5	4-5	c_2
Knoth	3	1-4	c_3
Smith	9	2-3	c_4

Table 4.2: $HotelBooking$

$client$	car_id	$period$	
Knoth	3	5-10	h_1
Schweizer	4	3-6	h_2
Knoth	4	1-3	h_3
Schweizer	3	1-2	h_4
Knoth	5	3-11	h_5

The above relations can be seen as $\mathcal{P}(\{c_1, c_2, c_3, c_4\})$, reps. $\mathcal{P}(\{h_1, h_2, h_3, h_4, h_5\})$-relation, by replacing c_1 by $\{c_1\}$ etc. After applying the modified composite query from the *Example 3.4.1*

$$V = V_1 \bowtie^T V_2, \text{ where } \begin{cases} V_1 = \pi^T_{client}(CarSharing \\ V_2 = \pi^T_{client}(HotelBooking) \end{cases}$$

to the above annotated relations we obtain according to *Definition 4.4* the $\mathcal{P}(\{c_1, c_2, c_3, c_4, h_1, h_2, h_3, h_4, h_5\})$-relation shown in *Table 4.3*

Table 4.3: $Where\text{-}provenance$

$client$	$period$	
Knoth	1-3	$\{c_3, h_3\}$
Knoth	3-4	$\{c_3, h_5\}$
Knoth	6-7	$\{\{c_1\}, \{h_1, h_5\}\}$

Using $Where\text{-}provenance$ on *annotated relations* provenance propagation happens in annotations automatically executing relational algebra operators. Thus we do not need to materialize the intermediate results anymore to know *which tuples contribute*. If one needs to know *how they contribute*, *provenance semiring* must be applied on $K\text{-}relations$, represented in the *Definition 4.5*.

4.3 Provenance semiring/How-provenance

As mentioned above, one uses *provenance semirings* to get information about *how the result was derived*. As in *Definition 4.5* was defined, one uses the semiring of polynomials with variables from X and coefficients from $\mathbb{N}$. The used operation are given from *Definition 4.4*: "+" and "-". "+" references to *alternative use of data*, "$\cdot$" to *joint use of data*.

Example 4.3.1 We consider the same example as in *Section 4.3*. The result is the $\mathbb{N}[c_1, c_2, c_3, c_4, h_1, h_2, h_3, h_4, h_5]\text{-}relation$ represented in *Table 4.4*.

Table 4.4: $How\text{-}provenance$

$client$	$period$	
Knoth	1-3	$c_3 \cdot h_3$
Knoth	3-4	$c_3 \cdot h_5$
Knoth	6-7	$c_1 \cdot h_1 + c_1 \cdot h_5$

We consider the tuple $(Knoth|[6-7))$ in the *Table 4.4*. It can be computed in two different ways. One of them uses tuples c_1 and h_1; the second one uses tuples c_1 and h_5.

This approach assumes queries involve select, project, join, and union only. The where-provenance and why-provenance of an output location are described through a set of propagation rules, one for each relational operator (i.e., select, project, join, union).

4.4 Provenance-propagating temporal relational algebra

Let notations be like in the *Definition 3.3*. In the Definition 3.3 each Schema consists of (A, T), where A is attribute list and T represent time intervals. Up now each Schema consists of (A, T, tag), where *tag* is an additional attribute that captures the provenance. Now we extend the lineage preserving set-based temporal algebra to provenance-propagating temporal relational algebra.

Definition 4.6 (*Provenance-propagating positive temporal set-based relational algebra*)

1. $\sigma_c^T(R) = \{t|\exists r \in R(t.A = r.A \wedge c(r) \wedge t.T = r.T \wedge t.tag = r.tag)\}$

2. $\pi_H^T(R) = \{t|$
$$\exists r \in R(t.H = k.H \wedge t.T \subseteq r.T \wedge t.tag = \sum_{r \in R, r.A = t.A, t.T \subseteq r.T} r.tag) \wedge$$
$$\forall x \in R(x.H = t.H \Rightarrow x.T \supseteq t.T \vee x.T \cap t.T \neq \emptyset) \wedge$$
$$\forall T' \supset t.T \exists x \in R(x.H = t.H \wedge T' \not\subseteq x.T \wedge T' \cap x.T \neq \emptyset)\}$$

3. $R \cup^T S = \{t|$
$$\exists r \in R(t.A = r.A \wedge t.T = r.T \wedge t.tag = r.tag \wedge$$
$$\forall s \in S(s.B = t.A \Rightarrow s.T \cap t.T = \emptyset) \wedge$$
$$\forall T' \supset t.T \exists s \in S(s.B = t.A \wedge T' \cap s.T \neq \emptyset \vee T' \not\subseteq r.T)) \vee$$
$$\exists r \in R(t.A = r.A \wedge$$
$$\exists s \in S(r.A = s.B \wedge t.T = r.T \cap t.T \wedge t.T \neq \emptyset \wedge t.tag = r.tag + s.tag)) \vee$$
$$\exists s \in S(t.A = s.B \wedge t.T \subseteq s.T \wedge t.tag = s.tag) \wedge$$
$$\forall r \in R(r.A = t.A \Rightarrow r.T \cap t.T = \emptyset) \wedge$$
$$\forall T' \supset t.T \exists s \in S(s.B = t.A \wedge T' \cap s.T \neq \ \vee T' \not\subseteq r.T))\}$$

4. $R \bowtie^T S = \{t|$
$$\exists r \in R \exists s \in S(t.A = r.A \wedge t.A = s.B \wedge t.T = r.T \cap s.T \wedge t.T \neq \emptyset \wedge$$

$t.tag = r.tag \cdot s.tag)\}$

Example 4.4.1 (*Application of provenance-propagating positive temporal algebra to composite query*) Let two temporal relations $"HotelBooking"$,$"CarSharing"$ are given (*Table 4.5* and *Table 4.6* respectively).

Table 4.5: $HotelBooking$

$client$	car_id	$period$	tag
Knoth	3	1-9	h_1
Knoth	4	1-5	h_2
Knoth	5	5-9	h_3

Table 4.6: $CarSharing$

$client$	$period$	tag
Knoth	5-7	c_1
Smith	1-3	c_2

We execute the following composite query:

$$(\sigma_{client="Smith"}(CarSharing)) \times^T (CarSharing \cup^T \pi_{client}(HotelBooking))$$

Execution of the above query includes the following steps:

Step 1: $V_1 = \pi_{client}(HotelBooking)$
Step 2: $V_2 = CarSharing \cup^T V_1$
Step 3: $V_4 = V_3 \times^T V_2$, where $V_3 = (\sigma_{client="Smith"}(CarSharing))$

Step1 is represented in the *Figure 4.1* and the result in the *Table 4.7*

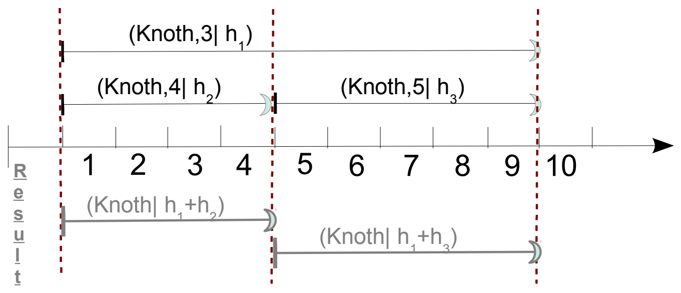


Figure 4.1: $V_1 = \pi_{client}(HotelBooking)$

Step 2 you can see on the *Figure 4.2* and in *Table 4.8*.
Finaly, *Step 3* can be found on the *Figure 4.3* and the result in th *Table 4.9*
We see that in the result Table 4.3 in the Attribute $"tag"$ we got provenance propagated automatically during operators execution.

Table 4.7: V_1

client	*period*	*tag*
Knoth	1-5	$h_1 + h_2$
Knoth	5-9	$h_1 + h_3$

Table 4.8: V_2

client	*period*	*tag*
Knoth	1-5	$h_1 + h_2$
Knoth	5-7	$h_1 + h_3 + c_1$
Knoth	7-9	$h_1 + h_2$
Smith	1-3	c_2

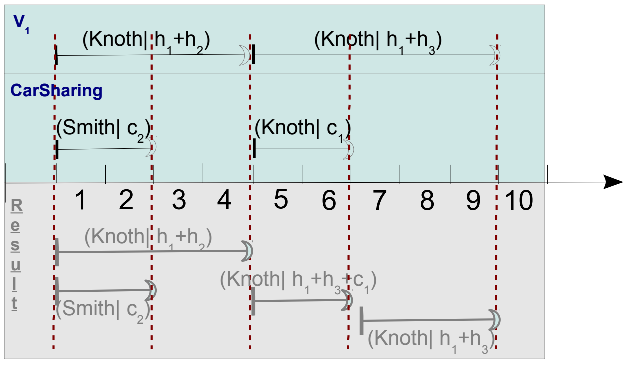


Figure 4.2: $V_2 = CarSharing \cup^T V_1$

Table 4.9: $V_4 = V_3 \times^T V_2$

$client_{V_3}$	$client_{V_2}$	*period*	*tag*
Smith	Smith	1-3	c_2^2
Smith	Knoth	1-3	$h_1 \cdot c_2 + h_2 \cdot c_2$

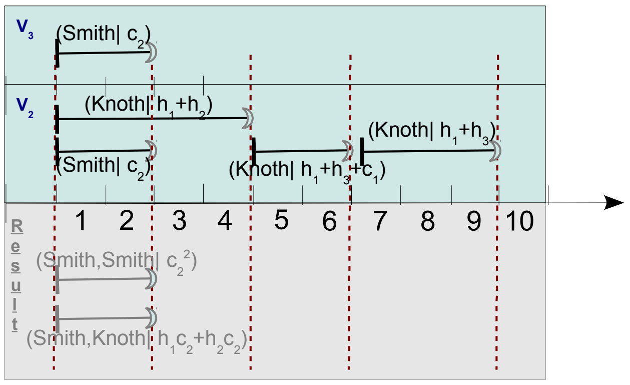


Figure 4.3: $V_4 = V_3 \times^T V_2$

5 Conclusion and future work

Relational algebra, lineage and provenance on the nontemporal databases can be in natural way extended to temporal model which must coincide with non temporal on points. Thus, using nontemporal models we could define temporal set-based lineage preserving relational algebra, trace of lineage, provenance semirings and provenance-propagating positive temporal set-based relational algebra for temporal models.

In Chapter 3 defined *trace of lineage* has some drawbacks: it forces intermediate results to be materialized and it does not provide any information about how the result was obtained. In Chapter 4 introduced annotated relations allow to capture the provenance and propagate it through execution of temporal operators of relational algebra. At the end of Chapter 4 defined *Provenance-propagating positive temporal set-based relational algebra* captures automatically provenance. In the Chapter 4 represented generalized algebra which operates on annotated relations is only defined for positive relational algebra, i.e. only for operators $selection$, $projection$, $union$ and $natural\ join$. That is why at the moment we cannot replace trace of lineage by provenance-propagating set-based temporal algebra. Different attempts was made to define the mapping of operators $difference$ and $aggregation$ to the operators on annotated relations ([Diff], [Gee], [Aggr]). In the paper [Diff] was proved that operator $difference$ doesn't satisfy the universal property, which one needs for example by deletion propagation. In the paper [Aggr] the authors was tried to define operator $aggregation$ and through aggregation operator $difference$ for m-semirings. The attempt to define the aggregation and difference on annotated relations using semirings forced a difficult construction, which becomes too unnatural. In the future the missing operators on m-semirings must be defined. Maybe even another mathematical structures should be found to make the provenance propagation for all relational algebra operators possible. And of course provenance for temporal databases should be implemented.

Bibliography

[Aggr] Y. Amsterdamer, D. Deutch, Val Tannen. *Provenance for Aggregate Queries*, Paper. Jan. 2011

[Böh] M. H. Böhlen, C. S. Jensen and R.T. Snodgrass *Temporal Statements Modifiers*. ACM Trans. Database Syst., 25(4):407-456, 2000.

[Bun] Buneman P. and Tan W.C. *Provenance in databases.* . ACM Trans. Database Syst., 25(2):179-227, 2000

[Cui] Y. Cui, J. Widom, and J.L. Wiener. *Tracing the lineage of view data in a warehousing environment*. ACM Trans. Database Syst., 25(2): 179-227, 2000

[Diff] D. Deutch, Y. Amsterdamer, Val Tannen. *On the Limitations of Provenance for Queries With Difference*, Paper. 2011

[Fos] J.N. Foster, Todd J. Green, G. Karvounarakis, Val Tannen, Z. Ives. *Provenance for Database Transformations*, in EDBT Keytone, Lausanne. March 2010.

[Gee] F. Geerts, A. Poggi. *On Database Query Language for K-relations*, Paper.

[Liu] L. Liu and M. T. Özsu, editors. Encyclopedia of Database Systems. Springer US, 2009

[Lor] N. A. Lorentzos. *Period-stamped temporal models*, in Liu and Özsu, pages 2094-2098.

[Tan] Todd J. Green, G. Karvounarakis, Val Tannen. *Provenance Semirings*, In L.Libkin, editor, POD, pages 31-40. ACM, 2007

[TanProv] W.C. Tan *Provenance*, in Liu and Özsu, page 2202.