Temporal Data Exchange

Ladan Golshanara
Supervised by Jan Chomicki
State University of New York at Buffalo
Buffalo, NY, USA
ladangol@buffalo.edu
Expected graduation date: September 2018

ABSTRACT

In this work, we study data exchange for temporal data. There are two views associated with temporal data: the concrete temporal view, which depicts how temporal data is compactly represented and on which implementations are based, and the abstract temporal view, which defines the semantics of temporal data. Based on the chase procedure, which is a fundamental tool in relational data exchange, two new kinds of chase are proposed in this paper: the abstract chase for the abstract temporal view and the concrete chase for the concrete temporal view. While labeled nulls are sufficient for relational data exchange, they have to be refined in temporal data exchange to keep the connection between the result produced by the concrete chase and the result of the abstract chase. We show that the concrete chase respects the semantics defined by the abstract chase and provides a foundation for query answering.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*

Keywords

Data exchange; Chase; Temporal database; Unknown values; Incomplete information

1. INTRODUCTION

A temporal database is a relational database where there is at least one time attribute associated with each relation schema. *Temporal data* is needed by many organizations such as insurance companies to support audit trails and store when what facts are true. Facts in temporal databases persist over intervals of time. For example, the fact that Ada worked for IBM between 2010 and 2013 can be represented as $(Ada, IBM, [2010, 2014))$ where $[2010, 2014)$ is a *clopen interval* consisting of the years 2010, 2011, 2012, and 2013. The fact that Ada has worked for Intel since then can be represented as $(Ada, Intel, [2014, \infty))$. Using an interval type for time attributes provides a compact way of storing the same data spanning multiple time points. Intervals can be finite or infinite. An infinite interval, such as $[2014, \infty)$, is a useful abstraction when the end point is not provided. Apart from storage concerns, it is simpler and more natural to view information at discrete time points. For these reasons, prior work on temporal databases [2, 7, 12] has provided two views of temporal data: *concrete temporal view* (or *concrete view* in short) and *abstract temporal view* (or *abstract view* in short). The concrete view provides a compact representation of temporal data over time intervals, such as Ada's employment history in the examples above. This view is an extension of the relational model where the time attributes take intervals as values. On the other hand, the abstract view shows temporal data at every time instant in order to give semantics to the concrete view. For example, there are four abstract facts, one for each year between 2010 to 2013, corresponding to the fact that Ada worked in IBM, and infinitely many abstract tuples $\{(Ada, Intel, 2014),\ (Ada, Intel, 2015), ...\}$, that correspond to the concrete temporal tuple $(Ada, Intel, [2014, \infty))$. The abstract view is based on the relational model with infinite relations. Note that SQL:2011 [9] implicitly supports both views. It provides features for defining tables with intervals, while the semantics of integrity constraints are defined with respect to time points.

Data exchange is the problem of translating data that conforms to one schema (called the *source schema*) into data that conforms to another schema (called the *target schema*), given a specification of the relationship between the two schemas. This relationship is specified by means of a schema mapping [4, 5]. The problem of data exchange has been extensively studied in the context where a schema mapping consists of *source-to-target tuple generating dependencies (st tgds)*, *target generating dependencies (tgds)* and *equality generating dependencies (egds)*. Given a schema mapping, the goal of data exchange is to determine an instance that conforms to the target schema and satisfies the specification. This target instance is called a *solution*. For a given source instance, there may be no solutions since there may not exist a target instance that satisfies the specification. On the other hand, there may be many solutions. It was shown in [4] that among all solutions of a given source instance, the *universal solutions* are the preferred solutions because they are the most general. In [4] the *chase procedure* is used to find a universal solution. Universal solutions can be

SIGMOD'16 PhD Symp, June 26–July 1, 2016, San Francisco, CA, USA.

ACM ISBN 978-1-4503-4192-9/16/06...$15.00

DOI: http://dx.doi.org/10.1145/2926693.2929900

used to determine *certain answers* of the union of conjunctive queries posed over a target schema. Certain answers of a query Q consist of all tuples that will be in the answer of Q over any arbitrary solution for a source instance and a schema mapping.

In this paper, we extend the standard framework of data exchange to the context of temporal data, studying *temporal data exchange*. To elicit the challenges that arise in temporal data exchange, we start by examining the most basic case where we assume that every relation in a source instance has exactly one temporal attribute and source instances are complete (i.e. have no unknown values). In our framework, the specification between the source and target schema consists of *s-t tgds*, and *temporal key constraints* [2] over the target schema. As a result of data exchange, unknown values may occur in target instances. One of the extensions we bring to the framework of temporal data exchange is the notion of *dynamic nulls* under the concrete view, which is used to model the unknown values of an attribute that may possibly vary over time. In data exchange, *named nulls* (or labeled nulls) are used to model unknown values in the solutions of a data exchange problem, where distinct named nulls denote possibly distinct unknown values. While named nulls continue to be sufficient in temporal data exchange under the abstract perspective, they are no longer sufficient under the concrete perspective as each concrete tuple represents multiple, possibly distinct, abstract tuples in general. To see this, consider the schema $Emp(Name, Position, Company, Time)$. The abstract tuple $(Ada, Programmer, N^{2011}, 2011)$ contains the named null N^{2011}. This tuple states that Ada was a programmer at an unknown company in 2011. On the other hand, the concrete tuple $(Ada, N^{[2008,\infty)}, IBM, [2008, \infty))$ contains the dynamic null $N^{[2008,\infty)}$, and the tuple denotes that the position of Ada at IBM is unknown and the position is possibly different every year since 2008. Though the idea of (named) nulls is not new and is used to model *incompleteness* in databases [6], the notion of a dynamic null, which is necessary to accurately model incompleteness in temporal data exchange under the concrete view is new to the best of our knowledge. Note that dynamic nulls are only used to show the correspondence between the result of data exchange on a concrete instance and the result of data exchange on an abstract instance. In other words, dynamic nulls are hidden from end users if the semantics of query answering is certain answers.

Every dynamic null $N^{[s,e)}$ in a concrete tuple has a *time context* $[s, e)$ which is inherited from the time interval of the tuple. Hence, dynamic nulls are *context-dependent*. Also, every dynamic null of the same concrete tuple will have identical time context as a consequence. Therefore, tuples such as $(Ada, Programmer, N^{[2008,\ 2012)}, [2000, \infty))$ are prohibited in our framework as it is unclear what the corresponding null for Ada's affiliation is in the abstract view of this tuple in the year 2000, or 2014.

A fundamental challenge of temporal data exchange is that the data exchange semantics over concrete source instance (which is the representation on which implementations are based) has to be carefully defined so that it corresponds to the standard semantics of data exchange over the corresponding abstract source instance. Here we provide an example of temporal data exchange:

Example 1. Suppose we have the following relation in the source schema that represents the employment histories of persons:

$$Employee(Name, Company, Time).$$

A concrete source instance is shown in figure 1(a), while the corresponding abstract instance is shown in figure 2(a). Due to space restrictions, only the last two digits of years are shown. We want to move the data to a bigger schema with two relations:

$$Emp(Name, Position, Company, Time),$$
$$Sal(Position, Salary, Time).$$

Suppose we have the following s-t tgd:

$$\forall n \forall c \forall t\ Employee(n, c, t) \rightarrow \exists p \exists s\ Emp(n, p, c, t) \wedge Sal(p, s, t)$$

and the temporal key constraint:

$$\forall p \forall s_1 \forall s_2 \forall t\ Sal(p, s_1, t) \wedge Sal(p, s_2, t) \rightarrow s_1 = s_2$$

A solution for the concrete source instance, that can be obtained using the concrete chase, is depicted in figures 1(b) and 1(c). A solution for the abstract instance, that can be constructed using the abstract chase, is shown in figures 2(b) and 2(c). Note that both solutions satisfy the above s-t tgd and temporal key constraint. Also note that the concrete and abstract solutions are semantically aligned. □

Research Problems and Results Here are our preliminary research problems and results :

1. How can we define the chase procedure on the abstract view where the relations can be infinite?
2. How can we define the chase procedure on the concrete view where the domain of the time attribute is the intervals?
3. Is the concrete chase correct? In other words, are the results of the concrete chase and the abstract chase semantically aligned?
4. Is the result of the concrete chase representative of the solution space?

We define two chase procedures, one for each view (abstract vs. concrete) of temporal data. We establish that the result of the abstract chase (*a-chase*) over the abstract view, is semantically aligned to the result of the concrete chase procedure (*c-chase*) over the concrete view. We also show that the result of c-chase provides a foundation for query answering.

The remainder of this paper is organized as follows. Section 2 provides an overview of the related work. In section 3 some preliminaries are given. The abstract chase is defined in section 4, while the concrete chase is defined in section 5. Section 6 discusses the universality property of the results of the proposed chases. We report our future plans in section 7. Section 8 provides some concluding remarks.

2. RELATED WORK

Here we overview the previous relevant work on data exchange and temporal databases.

The formal foundations of data exchange was developed by Fagin et al. in [4]. The authors applied the chase algorithm, previously used for checking implication of data

Name	Company	Time
Ada	IBM	[08, 10)
Ada	Intel	[10, 13)

(a)Employee(Source)

Name	Position	Company	Time
Ada	$N^{[08,10)}$	IBM	[08, 10)
Ada	$N^{[10,13)}$	Intel	[10, 13)

(b)Emp(Target)

Position	Salary	Time
$N^{[08,10)}$	$M^{[08,10)}$	[08, 10)
$N^{[10,13)}$	$M^{[10,13)}$	[10, 13)

(c)Sal(Target)

Figure 1: Concrete source and target instances

Name	Company	Time
Ada	IBM	08
Ada	IBM	09
Ada	Intel	10
Ada	Intel	11
Ada	Intel	12

(a)Employee(Source)

Name	Position	Company	Time
Ada	N^{08}	IBM	08
Ada	N^{09}	IBM	09
Ada	N^{10}	Intel	10
Ada	N^{11}	Intel	11
Ada	N^{12}	Intel	12

(b)Emp(Target)

Position	Salary	Time
N^{08}	M^{08}	08
N^{09}	M^{09}	09
N^{10}	M^{10}	10
N^{11}	M^{11}	11
N^{12}	M^{12}	12

(c)Sal(Target)

Figure 2: Abstract source and target instances

dependencies, for data exchange and query answering. The original chase, which we call the standard chase throughout the paper, assumes finite relational instances. Further extensions to the standard chase were surveyed in [10]. In [3] a parallel chase was proposed for s-t tgds, that can be used for infinite relations, but the details in the case of egds were missing. Our work is different from previous work on data exchange in two aspects: the abstract view requires infinite sets of tuples and the concrete view is an extension of relational model with intervals as attributes values.

The formal foundations of temporal data models and query languages were studied in [2, 13]. Abstract versus concrete temporal views were first developed in the context of semantics of temporal query languages. The papers [2, 13] did not discuss incomplete temporal information and its possible semantics. Koubarakis proposed a unified framework for both finite and infinite, definite and indefinite temporal data [8, 7]. His suggested framework extends c-tables [6] and can be used to store facts such as room1 is booked from 2 to *sometime between 5 to 8*. He used global conditions to store the constraints on the start point or end point of a time interval. In his framework, the temporal attribute can be unknown. On the other hand, we are dealing with unknown values for non-temporal attributes in the concrete view (i.e., dynamic nulls).

3. PRELIMINARIES

A *named null* is a pair N^s where N is a label and the superscript s is a time point which shows the temporal context. A *dynamic null* is a pair $N^{[s,e)}$ where the base is a label and the superscript $[s, e)$ is a time interval carrying the temporal context. Two dynamic nulls (resp., named nulls) are equal, if they are syntactically identical.

An *incomplete abstract relation* has the schema $\{A_1, ..., A_m, T\}$, where the domain of T is a set of time points and the domain of attributes $A_1, ..., A_m$ is a set of constants and named nulls. A complete abstract relation is a special case of an incomplete relation where the attributes $A_1, ..., A_m$ take only constant values. An abstract instance is a set of abstract relations. If it is a complete instance all relations in it are complete abstract relations; otherwise, it is an incomplete instance. Similarly, a concrete relation has the same schema, but the domain of T is a set of time intervals. A concrete relation is incomplete, if the attributes $A_1, ..., A_m$ take values from a set of constants and dynamic nulls. In case of a complete concrete relation, the values for $A_1, ..., A_m$ are taken only from a set of constants.

Definition 1. Semantics mapping:
Each concrete tuple $(a_1, ..., a_k, [s, e))$ is mapped to a set of abstract tuples under the semantics mapping, denoted by $[\![.]\!]$. That is, $[\![(a_1, ..., a_k, [s, e))]\!] = \{ (a'_1, ..., a'_k, t_0) \mid s \leq t_0 < e$ where $a'_i = a_i$ if a_i is a constant, and a'_i is the named null N^{t_0} if a_i is a dynamic null $N^{[s,e)}\}$. Similarly, $[\![R]\!] = \bigcup_{u \in R}([\![u]\!])$, where R denotes a concrete relation.

If the named nulls were not introduced, one had to generate a fresh labeled null for each time point in a dynamic null's context $[s, e)$. To clarify, suppose we have two facts in two different relations $Emp(Ada, N^{[s,e)}, IBM, [s, e))$ and $Sal(N^{[s,e]}, 20000, [s, e))$. When we take the semantics mapping of these two facts, in order not to lose the connection between the concrete and the corresponding abstract facts, we need to keep track of the labeled nulls we generate for $N^{[s,e)}$ in $[\![Emp]\!]$ to produce the same set in $[\![Sal]\!]$. Named nulls, on the other hand, keep the base identical to the corresponding dynamic null and carry the context with their superscript.

Definition 2. Source-to-Target Dependency (s-t tgd) :
An s-t tgd is a dependency of the form:

$$\forall \bar{x}, t\ \alpha(\bar{x}, t) \rightarrow \exists \bar{y}\ \beta(\bar{x}, \bar{y}, t)$$

where α (resp.,β) is a conjunction of relational atoms (i.e., $\alpha = \alpha_1 \wedge ... \wedge \alpha_m$ (resp., $\beta = \beta_1 \wedge ... \wedge \beta_n$)) over the source schema (resp., target schema), $\bar{x}$ and $\bar{y}$ are non-temporal variables and t denotes the temporal variable.

Definition 3. Temporal Key Constraint :
A *temporal key constraint* over a relational schema $R(\bar{A}, \bar{B}, T)$ is of the form:

$$R(\bar{x}, \bar{y}, t) \wedge R(\bar{x}, \bar{y'}, t) \rightarrow \bar{y} = \bar{y'}$$

In this case, we say the set $\{\bar{A}, T\}$ of attributes is the *temporal key*[1] of relation R.

Throughout the paper, σ is used to refer to a single s-t tgd or temporal key constraint, while Σ_{st} and Σ_k refer to a set of s-t tgds and temporal key constraints, respectively.

[1]Temporal key attributes can have named null and dynamic null values in our framework.

Definition 4. Homomorphism:
A homomorphism h from a formula $\alpha(\bar{x}, t)$, denoting an s-t tgd or a temporal key constraint, to an abstract instance (resp., concrete instance) I is a mapping from the variables $\bar{x}$ to constants and named nulls (resp., dynamic nulls) and from t to time points (resp., time intervals) such that for every atom $R_i(\bar{x}, t)$ in α, the fact $R_i(h(\bar{x}, t))$ is in I.

Definition 5. Solution:
Let R_S be a source schema and R_T be a target schema. Let Σ_{st} be a set of s-t tgds and Σ_k a set of temporal key constraints. Let I be a source instance over R_S, then an instance J over R_T is called a *solution* if (I, J) satisfies $\Sigma_{st} \cup \Sigma_k$.

4. ABSTRACT CHASE

An abstract instance is very similar to a relational instance, but it can be infinite. Hence, the standard chase cannot be used because it is a *sequential* procedure. This motivates us to define the chase in *parallel* and as the union of individual chase steps. Indeed, it is much easier to reason about an infinite union of single steps rather than an infinite sequence of them where one needs to define appropriate notions of fix-point and convergence. The *abstract chase procedure* consists of a *parallel abstract chase step with s-t tgds* followed by a *parallel abstract chase step with temporal key constraints*.

Let I_a be a complete abstract instance. Let Σ_{st} be a set of s-t tgds. We define J_a to be the result of a *parallel abstract chase step(or a-chase$^*_{\Sigma_{st}}$ in short)* on I_a, obtained by union of the results of all chase steps, applied on the tuples of I_a with all dependencies in Σ_{st}. Note that J_a is an incomplete abstract instance and (I_a, J_a) satisfies Σ_{st}. In other words, J_a is a solution for I_a with respect to Σ_{st}. Next, we apply the parallel abstract chase step with temporal key constraints on J_a. If this application is successful, then the result is a solution that satisfies both s-t tgds and temporal key constraints; otherwise, the abstract chase procedure fails and no solution is produced.

Defining the parallel application of chase steps for temporal key constraints (or egds in general) is trickier. For these constraints we cannot apply chase steps independently, because we might equate one null value with two distinct constants in two chase steps. The following example clarifies this point.

Example 2. Consider the following Emp relation :

Name	Position	Company	Time
Ada	N^{2008}	IBM	2008
Ada	DBA	IBM	2008
David	N^{2008}	Intel	2008
David	Manager	Intel	2008

and the temporal key constraint:

$$Emp(n, p, c, t) \wedge Emp(n, p', c, t) \rightarrow p = p'$$

Notice that the sequential application of chase steps on this instance fails because DBA $\neq$ Manager. Indeed, there is no solution for this instance with the above temporal key constraint. However, the parallel application of chase steps will not fail because one of the chase steps results in replacing N^{2008} with DBA in the first tuple and the other replaces N^{2008} with Manager in the third tuple, independently of each other. Therefore, we define a chase step with a temporal key constraint σ on an incomplete abstract instance J_a such that, it identifies a set of equalities, representing the modifications, instead of modifying the relation. Then, in the *parallel abstract chase step with temporal key constraints (a-chase$^*_{\Sigma_k}$ in short)* we take the union of all equalities obtained in individual chase steps and reason about the equalities that can be deduced by considering their symmetric transitive closure. If we can deduce an equality between two distinct constants, then the abstract chase procedure fails. Otherwise, the named nulls in J_a are replaced by other constants or named nulls based on the set of equalities. The final result is a solution w.r.t. $\Sigma_{st} \cup \Sigma_k$.

5. CONCRETE CHASE

Like a relational instance, a concrete instance is finite. However, we cannot apply the standard chase because we have intervals as values for the time attribute. In order to find a homomorphism from an s-t tgd or a temporal key constraint to a concrete instance I_c, the tuples in that instance should be *normalized*. A set of concrete tuples (that might be over different schemas) is normalized if the time intervals of the tuples are either equal or disjoint (i.e., no overlap in the intervals). Intuitively, normalization makes time intervals behave like time points.

To be aligned with the abstract chase, the *concrete chase procedure* consists of two parallel concrete chase steps which are applied successively. The *parallel concrete chase step with s-t tgds (c-chase$^*_{\Sigma_{st}}$ in short)* is defined as the union of individual c-chase steps. For the *parallel concrete chase step with temporal key constraints Σ_k (c-chase$^*_{\Sigma_k}$ in short)*, the approach is to generate a set of equalities for each c-chase step and modify the instance after reasoning on all the equalities, same as the abstract case.

We have shown that we can simulate the abstract chase procedure by the concrete chase procedure for both s-t tgds and temporal key constraints. In other words, given a concrete instance I_c, the proof for Σ_{st} shows that $[\![c\text{-}chase^*_{\Sigma_{st}}(I_c)]\!]$ is homomorphically equivalent to $a\text{-}chase^*_{\Sigma_{st}}([\![I_c]\!])$. This is depicted in figure 3.

6. UNIVERSAL SOLUTION

For a source instance and a set of dependencies in $\Sigma_{st} \cup \Sigma_k$ there might be infinitely many solutions. The class of universal solutions consists of good candidates to be materialized, because there is a homomorphism from them to any other solution for the source instance. We have shown that the concrete instance obtained by the concrete chase is a good candidate to be materialized and used to answer queries posed over it. In order to show this, we have proved that the result of the abstract chase is a universal solution and then used our previous result that the semantics mapping of the result of the concrete chase is homomorphically equivalent to the result of the abstract chase.

For proving that the solution obtained by abstract is universal, we need to show that there is a homomorphism from it to any other solution. This was proved for the result of the standard chase in [4]. However, we cannot use this result directly because our chase steps are applied in parallel on the source instance (and not sequentially as in [4]). For s-t tgds, we proved that there is a homomorphism from partial

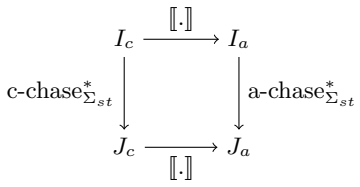

Figure 3: Simulation of a-chase and c-chase for s-t tgds

results of a single a-chase step to any arbitrary solution. Using this property, we have shown there is a homomorphism from the union of these partial results which is obtained by *a-chase*$^*_{\Sigma_{st}}$ to any arbitrary solution. For temporal key constraints, we showed that there is a homomorphism from the result of *a-chase*$^*_{\Sigma_k}$ to any instance that satisfies Σ_k.

7. RESEARCH PLAN

This PhD project is at the end of its second year. We came up with a formal framework for temporal data exchange and introduced two novel chase procedures: the abstract chase and the concrete chase. It is still open whether named/dynamic nulls can be simulated by labeled nulls.

We have started to investigate the problem of query answering over concrete instances. We plan to choose certain answers as the semantics of query answering in our framework because it is the prevailing semantics in data exchange. Our goal is to find tractable query classes and explore *naïve evaluation* [1] on a concrete instance as a technique to evaluate queries. Two classes of queries that are of our interest are the unions of conjunctive queries and queries involving comparison operators on the temporal variable. A good starting point to deal with the latter queries is the paper [11] which explores arithmetic operations in s-t tgds and tgds.

We also plan to evaluate the practical applicability of what we have proposed by implementing a temporal data exchange system for real use cases.

8. CONCLUSIONS

In this paper, we proposed a framework for temporal data exchange. We considered a basic case where the source instance is complete and has a single temporal dimension. The schema mappings consist of a set of s-t tgds and temporal key constraints. We proposed two chase procedures: one for a concrete temporal instance and one for an abstract temporal instance. The abstract chase makes it possible to define the semantics of the concrete chase and to prove that the concrete chase is correct. We have defined the abstract chase as the parallel application of chase steps to avoid reasoning over an infinite sequence of chase steps. We also discussed that application of chase steps for temporal key constraints requires more care as not to equate one unknown value with two different constants in two chase steps. Then we defined the concrete chase and proved that its result is semantically aligned to the one achieved by the abstract chase. Finally, we showed the result of the abstract chase is a universal solution and since we can simulate the abstract chase by the concrete chase, the result of the concrete chase is a good candidate to be materialized and used for answering queries.

9. ACKNOWLEDGMENTS

This work is supported in part by NSF grant IIS-1450590. We thank our collaborator Dr. Wang-Chiew Tan for her fruitful discussion and support.

10. REFERENCES

[1] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Foundations of Data Exchange*. Cambridge University Press, New York, NY, USA, 2014.

[2] J. Chomicki. Temporal query languages: A survey. In *Temporal Logic, First International Conference, ICTL '94, Bonn, Germany, July 11-14, 1994, Proceedings*, pages 506–534, 1994.

[3] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, pages 149–158, 2008.

[4] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, May 2005.

[5] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, March 2005.

[6] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, September 1984.

[7] M. Koubarakis. Database models for infinite and indefinite temporal information. *Inf. Syst.*, 19(2):141–173, March 1994.

[8] M. Koubarakis. Foundations of indefinite constraint databases. In *Principles and Practice of Constraint Programming, Second International Workshop*, pages 266–280. Springer, 1994.

[9] K. G. Kulkarni and J. Michels. Temporal features in SQL: 2011. *SIGMOD Record*, 41(3):34–43, October 2012.

[10] A. Onet. The chase procedure and its applications in data exchange. In *Data Exchange, Information, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 1–37. 2013.

[11] B. ten Cate, P. G. Kolaitis, and W. Othman. Data exchange with arithmetic operations. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 537–548, 2013.

[12] D. Toman. Point vs. interval-based query languages for temporal databases. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '96, pages 58–67, 1996.

[13] D. Toman. SQL/TP: A temporal extension of SQL. In *Constraint Databases*, pages 391–399, 2000.