Data Pre-processing: Case of Sensor Data Consistency Based on Bi-temporal Concepts

Faiza Allah Bukhsh[1],Patricio de Alencar Silva[2],Hans Wienen[1]
[1]Department of Computer Science,
University of Twente,
Enschede, the Netherlands

[2]Department of Computer Science
Federal University of Semi-arid Region(UFERSA),
Mossoro, Brazil
f.a.bukhsh@utwente.nl, patricio.alencar@ufersa.edu.br, h.c.a.wienen@utwente.nl

***Abstract*—The volume, velocity, variety, veracity and value of data currently produced and consumed by different types of information systems turned big Data into a phenomena of study. For data variety, temporal data commonly represents a source of potential inconsistency. This paper reports on a research endeavor for treating the problem of how to minimize inconsistencies in temporal databases due to unavailability of big data. This problem often occurs in situations where a same query is executed on the same data set at different points in time. To address this issue, we propose query optimization strategies based on query transformation and rewriting rules, to amend data consistency in temporal databases. We validate these strategies proposed via case scenario in sensor data analysis, and via manual data input, both for local and distributed query environments.**

I. Introduction

The nature of big Data, characterized by unprecedented volume, velocity, variety, veracity and value of data, poses not only new challenges, but also opportunities to the Database research community [1]. It may become specifically difficult to reconcile new demands for velocity in data sensing and storing with capabilities and strategies for maintaining temporal data consistency [2]. In the era of smart metering markets, this reconciliation is critical to generate value out of massive sensors data. For instance, the operations of many industrial segments rely on sensors data, the prospects of which are directly related to Enterprise value generation [3]. If time-sensitive data is not pre-processed, or if it is simply misinterpreted, it may lead to inefficient or even ineffective decision-making.

Therefore, the research scope considered here encompasses the business need for pre-processing an increasing inflow of sensors data, yet preventing from temporal data inconsistency. More specifically, the main research problem reported in this paper comprises what and how strategies for data pre-processing could be combined to prevent big data unavailability due to data inconsistency . The relevance of this problem is critical before basic requirements of ACID (atomicity, consistency, isolation and durability) for data management[4]. The problem-solving approach orienting this research is based on principles of Design Science, which provides guidelines for research on problems of practical relevance[5], [6].

The methodology used in this research comprises a combination of three research methods: (1) a real-world case scenario in sensors data, for elicitation of basic requirements for temporal consistency in big data production; (2) formal mechanism design, for specifying functional capabilities of query rewriting rules and query transformation; and (3) data analysis, for validating the strategies proposed for data pre-processing. Anonymized data used in this paper is inspired from data produced by sensors deployed at "Thur Valley" Switzerland. In this case scenario, chemical analysis of water samples is stored manually in a database. The structure of sensors measures the temperature of the water on different intervals of time, which is stored in the database as streaming data. The time-sensitive data is the point of possible inconsistency [7]. More specifically, querying over the

sensor sometimes produces different results from a same query at two different execution times. The database stores both manual input data and streaming data. These data input formats also pose different challenges for preserving temporal data consistency [8].

The following sections of this paper are organized as follows. In Section 2, we describe the case scenarios used to explain the need for preventing sensors' data management from temporal inconsistency. In Section 3, we elaborate on a formal specification of strategies of query rewriting rules and query transformation to deal with the research problem in focus. Moreover in the same section, we evaluate the proposed strategies through data analysis in case scenarios for both local and distributed query environments. The main conclusions and directions for future work are summarized in Section 4.

II. Problem Statement

A common big data scenario is when data flows continuously out of a domain application, either as streaming data or manual input data [9]. Let us suppose that one submits a query on the database requesting for data sampled at some point in the past time, but produces no results for manual input data. Thus, it is possible that there will be no results for that specific query. However, after a few days, one submits the same query again, which now yields a different result. This common situation raises the question of "why a same query with same constraints produces different results on the same database in different points in time".

Streaming and manual input data are annotated with sampling time and input time respectively. Let us consider that data sampling at 2015-12-02T10:30:00. Streaming data is committed in the database at 2015-12-02T10:30:00.

Here, streaming data input time is the same as data sampling time. There is no delay for the streaming data input. We queried the data after an hour and found no results. After a day the manual data was entered into the database. After querying the database at 2015-12-04T12:30:00, we found entirely different results. Therefore, running the same query at two different times produces different results. At a first glance, we considered this difference of time as a possible cause of different results from the same query. Thus, we described the local case problem when we have all the data at a single location. Now, let us consider the case when we have data distributed at different locations and face the same problem of different results from a single query at different execution times. In another scenario, a query is submitted to Location B, where the results are stored. From a same query, different results are yielded at different points in time. The problem situation here seems to be the same of the one in a local scenario, i.e. data unavailability. For a certain time interval, there was no data in the result table, despite the fact that the database had both manual and streaming data stored. Considering that communication interruption might be the most possible cause of this situation, the problem in focus now seems to be how to retrieve the missing queried data.

Whenever there is no result after the execution of a query, we can consider that either the result of the query is null, or there is a problem with the data in the database. In a local scenario, data is not entered in the database, so the resultant tuple is empty. In a distributed scenario, either the local scenario condition exists, or the communication between the sites is interrupted. However, the same query may yield different results at another point in time. Overall, the problem here is how to prevent big data inconsistency to occur as a result of multi-temporal query execution over distributed databases [10], [11].

III. Solution Approach

Problem statement shows that we have two sub-problems one is for local and other is for distributed environment.These are interconnected so we will explore them in parallel. In the following, we will propose the solution to the problem highlighted in preceding section.

A. Approach for Local & Distributed Environment

Conceptually, we are dealing with time-sensitive data, thus it is critical to store valid sampling time in the database. A temporal database normally uses the notation of time for differentiating versions of data. More specifically, the temporal aspects include valid time and transaction time, the combination of which comprises Bi-Temporal data[12], [13]. In the solution approach proposed in this paper, we add transaction time to the database and analyze the results. By keeping the valid time and transaction time, we are actually recording the relationship between data from the real world and

data from the database world[14]. Transaction time validates consistency of data committed to the database. Therefore, our problem of yielding different results from executing a same query at two different execution times can be solved by using consistency control mechanisms based on transaction time. The scope of valid time is the real world, whereas the scope of transaction time is the database world. It can be noticed that valid time and transaction time for the streaming data is the same, while in case of manual data there is a time difference between valid time and transaction time.

Data storage and data processing through querying may locate at different sites and data might somehow migrate between sites [15]. In this case, we have to deal with the problem of yielding different results from a same query at different execution times. Major cause of different results is unavailability of data. There are two possible causes for unavailability of data in distributed scenario: either data is not entered into the database or the communication between the sites is interrupted, thereby stopping data flow. Adding transaction time and valid time is useful to verify if data is entered to the database, but it will not be sufficient to obtain information about the communication channel[14]. In order to monitor the communication channel between the sites, we need to maintain the record of all the events as a history table, whereby we can look at the time slots of disconnection, which will provide means to diagnose the actual cause of data unavailability. Besides, this information will also be used to maintain consistent results from the same query at two different execution times [12], [13].

In some cases, if some data is missing, instead of executing the same query and get the full data set again, we might select the information from the history table and set the constraints in the query only for the missing data only. If we add a history table at both considered sites. If we have two different results from the same query, then first it is necessary to check the transaction time to verify if data was committed on the database. If the corresponding data exist in the database, then it is also necessary to check the state of the communication channel. If a connection problem is identified, then the history table will be used to produce the same results out of a same query for two different points in time. If the query was executed at the time of disconnection, then there will be incomplete results. Whenever this query is executed again, we will get the information from the history table, so as to produce equivalent results[16]. Moreover, it is also possible to retrieve the starting and ending time of disconnection from the history table, in order to resubmit the query for that specific time frame. This strategy aims to save execution time and underlying resources.

B. Data Storage into Temporal Database

For storing the data into the temporal database we need to record the valid time. Valid time is usually being entered by the user but user is unaware of the transaction time. To store the data in Temporal database we need to store the transaction time in the database transparent from the user [17], [18]. For the manual data, database management system (DBMS) will add the transaction time to the incoming data transparently from the user with the help of following query.

```
Query 1
UPDATE tablename
SET transactiontime = systemtime
WHERE inputflag = ''T'';
```

The challenge is to handle large streams of sensor data and store it with transaction time. Streaming data management system (SDMS) is responsible for handling streaming data [9]. The transparent addition of transaction time with every tuple, is done by SDMS by using the continuous query [19] which is the building block for Complex Event Processing (CEP) systems and event correlation engines. Continuous queries let users get new results from the database without issuing the same command again and again. Moreover it is ideal for our situations in which an application repeats a particular query, and would benefit from always having instant access to the up-to-date result of that query. We are following the abstract approach of continuous query [20] for adding transactions time with a tuple into the database and trigger it by input into the database. According to the continuous query concepts format of the query will be as follows

```
Query2
SELECT valid.time
FROM tablename
WHERE inputflag= ''T''
For (No. of records==0 ; No. of records— )
{Insert into TableName values
( Transaction time= systime ) }
```

While entering the data in the database, query III-B will add the transaction time to the tuples of streaming data transparent from the user.

C. Query Transformation

For getting results we need to query the states. As we have added the transaction time transparent from the

user so we need to add constraints related to transaction time with the query. These addition of constraints needs to be transparent from user also. The results produced by the query can be needed later [21]. Instead of storing the results we should store the query because storing the results is not an efficient way. But before storing the queries we need to rewrite the queries for future use with updated constraints. The rewritten query will be stored in the query storage database for future usage. Results produced by the stored query should be same as produced by the rewritten query. As discussed in the previous section the problem of predicate can also handled by query rewriting. Storage of query in the database and replacement of predicate with transaction time is dependent on the type of predicate [22], [23]. In our scenario we mostly used the predicate "NOW".

D. Query Rewrite Rules

Transparent addition of transaction time to the inputted data has changed the structure of database. Now it is necessary to change the query because user is totally unaware of the addition of the transaction time in the database It is obvious that the user who is asking queries will ask query without constraints of transaction time[22], [23]. It is clear that the user who is executing queries will be ignorant from this change. So we need to transform the incoming query into database compatible format.

Selection of predicates "NOW" complicate the situation. If we store the queries with these predicates and execute that at later times the results will be different because value of predicate "NOW" will incorporate the current time into the query while executing it. So we have to find out a way to store the value of predicates. Either we use the continuous query or the SQL query we will face the same problem of predicate [19]. To resolve the problem of predicate we will define query rewrite rules which will help us in query rewriting for the follows purposes

- Provides us a way to add transaction time with the query transparent from user.
- Helps us to execute the query by replacing the predicate "NOW" with current value.
- Provide us a way to store the queries for future use by substituting the predicate values.

We can transform the query with the help of following rules, $q' = T_{NOW}(q)$.

1) Select Query: Select statement is used to select data from the database on the basis of some constraints. Attributes A_1 and A_2 will be selected from table t_1 and t_2 on the basis of conditions c_1, c_2 and c_3. As relations are at different locations so we need to manage the history table for keeping track of the action and disconnection. While query transformation we will check if the relation is at location A and transaction time is equal to current time then we will return the empty set else we will return the relation as it is as expressed in the following rule,

$$q' = T_{NOW}(R@A) \begin{cases} \begin{pmatrix} \text{if } \sum Loc = A \& TT = T_{NOW}, \\ History = EmptySet \end{pmatrix} \\ R \end{cases}$$

(Rule i.)

2) Insert Query: For query transformation we need to convert the insertion query by using following rule,

$$\text{Insert into } t_1 \text{ values}(t) = \text{Insert into } t_1(A_1, A_2, \ldots, A_n) \text{ values}(t)$$

As the transaction time will be added to the database transparent from the user so we need to transform the insertion query from user understandable format to database compatible format. Following rule will add the transaction time as the column name as well as value,

$$\begin{aligned} q' &= T_{NOW}(\text{Insert into } t_1(A_1, A_2, \ldots, A_n) \text{ values } (t)) \\ &= \text{Insert into } t_1(A_1, \ldots, A_n, TT) \text{ values } (T_{NOW}(t) \times T_{NOW}) \end{aligned}$$

(Rule ii)

Following rule will add the transaction time with the tuple,

3) Delete Query: Delete statement delete a row from the table on the basis of some constraints, following rule will add the transaction time in the constraints of the delete query.

$$q' = \begin{pmatrix} T_{NOW}(\text{Delete from } t_1 \text{ where } t) \\ = \text{Delete from } t_1 \text{ where } T_{NOW}(t) \end{pmatrix} \quad \text{(Rule iii)}$$

E. Representative Example

Continuing with the example of temperature and chemical data, we need to add transaction time with every inputted tuple. Addition of transaction time supposed to be transparent from the user. Query rewrite rules help us to translate the incoming query into a query having transaction time. Consider the data in the TemperatureData(Sr.No, ValidTime, Temp, Loc) and ChemicalData(Sr.No, ValidTime, NO, C, K, Loc) tables. We have to insert that data into the database with transaction time. Suppose current time is 2013-12-02T11:30:00.000 and we have following insert query,

q =insert into chemical (validtime, Temp, location)

TABLE I
TEMPERATURE DATA WITH TRANSACTION TIME

No.	ValidTime	TransactionTime	Temp	Loc
1	2013-12-01T11:30:00.000	2013-12-02T11:30:00.000	5	1
2	2013-12-01T17:50:00.000	2013-12-02T17:50:00.000	5.2	5
3	2013-12-02T12:10:30.000	2013-12-03T12:10:30.000	5	3
4	2013-12-04T11:30:00.000	2013-12-05T11:30:00.000	4	3
5	2013-12-05T11:22:00.000	2013-12-06T11:22:00.000	4.5	6

TABLE II
CHEMICAL DATA WITH TRANSACTION TIME

No.	ValidTime	TransactionTime	NO	C	K	Loc
1	2013-12-01T12:00:00.000	2013-12-02T12:00:00.000	3	5	6	3
2	2013-12-02T10:30:00.000	2013-12-03T10:30:00.000	4	3	5	3
3	2013-12-04T11:00:00.000	2013-12-05T11:00:00.000	5	6	6	5
4	2013-12-05T11:30:00.000	2013-12-06T11:30:00.000	3	5	2	6

values ((2013-12-01T11:30:00.000, 5, 1)

$q' = T_{NOW}$(Insert into chemical (validtime, Temp, location)
values (2013-12-01T11:30:00.000, 5, 1))

By applying Rule (ii)

$q' =$Insert into chemical (validtime, Temp, location, TT)
values (T_{NOW}(2013-12-01T11:30:00.000, 5, 1), T_{NOW})

By applying rule (iii)

$q' =$(Insert into chemical (validtime, Temp, location, TT)
values (2013-12-01T11:30:00.000, 5, 1 , T_{NOW}))

So in this way transaction time will be added with every tuple. The tuples of Table I and Table II are produced with the help of rules. Thus a query have to be transformed before execution, in order to obtain accurate results. User want to know the values of Temperature, NO, C and Validtime when validtime of chemical and temperature values are varying with a difference of 10 minutes as shown in the following query.

Query 1. Query 3.

```
Select Temp, NO,C, t.validtime
FROM chemical as c, Temp as t
WHERE c.validtime =>    t.validtime −10 min
AND       c.validtime =< t.validtime + 10min
AND       c.location = t.location
```

$$q = \pi\ Temp, NO, C, t.validtime(\sigma(c.validtime => t.validtime - 10min \wedge c.validtime <= t.validtime + 10min \wedge c.location = t.location)(CHEM \times TEMP))$$

The DBMS and SDMS will append required constraints in the query with the help of transformation rules.

$$q = (\pi(Temp, NO, C, t.validtime(\sigma c.validtime => t.validtime - 10min \wedge c.validtime <= t.validtime + 10min \wedge c.location = t.location(CHEM \times TEMP))))$$

$$q' = T_{NOW}(\pi\ Temp, NO, C, t.validtime(\sigma c.validtime => t.validtime - 10min \wedge c.validtime <= t.validtime + 10min \wedge c.location = t.location(CHEM \times TEMP)))$$

TABLE III
HISTORY FOR LOCATION A– DC=DISCONNECT, RC=RECONNECT A

#	Validtime	Event	Comment	TransactionTime	Loc
1	2015-11-29T12:00:00.000	No data	DC	2015-11-30T12:01:00.000	A
2	2015-12-01T12:00:00.000	data	RC	2015-12-01T12:15:00.000	A
3	2015-12-01T15:00:00.000	No data	DC	2015-12-01T15:10:00.000	A
4	2015-12-03T15:00:00.000	data	RC	2015-12-03T15:10:00.000	A
5	2015-12-05T09:00:00.000	No data	DC	2015-12-05T10:00:00.000	A
6	2015-12-05T15:00:00.000	data	RC	2015-12-05T15:10:00.000	A

TABLE IV
HISTORY FOR LOCATION B

#	Validtime	Event	Comment	TransactionTime	Loc
1	2015-11-27T12:00:00.000	no data	DC	2015-11-30T12:01:00.000	B
2	2015-12-30T15:00:00.000	data	RC	2015-12-02T15:30:00.000	B

When we execute the above query at 2013-12-02T12:30:00.000 it displayed no results because there were no data in the database. Same query when executed at 2013-12-05T12:00:00.000 come up with following tuples

Temp	NO	C	t.Validtime
5	4	3	2013-12-02T10:30:00.000
4.5	3	5	2013-12-05T11:30:00.000

Instead of storing the results we can apply the query transformation rules to store the query. These rules will automatically store the query with the value instead of predicate. Consider the transformation of following query in which we are replacing the value of predicate with time 2013-12-05T12:00:00.000.

$$q' = T_{NOW}(\pi Temp, NO, C, t.validtime(\sigma c.validtime => t.validtime - 10min \wedge c.validtime <= t.validtime + 10min \wedge c.location = t.loaction \wedge c.transactiontime <= NOW - 1sec \wedge t.transactiontime <= NOW - 1sec(CHEM \times TEMP))$$

By applying rule (i) following query has been produced,

$$q' = \pi Temp, NO, C, t.validtime(\sigma c.validtime => t.validtime - 10min \wedge c.validtime <= t.validtime + 10min \wedge c.location = t.loaction \wedge c.transactiontime <= NOW - 1sec \wedge t.transactiontime <= NOW - 1sec(\sigma_{TT<T_{NOW}}(CHEM) \times \sigma_{TT<T_{NOW}}(TEMP))$$

(Query 4)

We can conclude that we need to maintain a history table(III, IV) at each location.

By applying the rule (i) on query 4 we will get following query,

$$q' = \sigma_{Locatio_A}(\boldsymbol{\pi} Temp, NO, C, t.validtime (\boldsymbol{\sigma} c.validtime => t.validtime - 10min \wedge$$

$$c.validtime <= t.validtime + 10min \wedge$$
$$c.location = t.loaction \wedge c.transactiontime$$
$$<= 2015-12-05T12:00:00.000 - 1sec \wedge$$
$$t.transactiontime <= 2015-12-05T12:00:00.000$$
$$- 1sec(TNOW(CHEM)\ X\ TNOW(TEMP)))x(y@LocA)$$
(Query 5)

Maintaining the history also helps us to save time and resources by helping us to maintain the constraints only for missing data. History table tells us that there was some disconnection for site A for certain period of time.

IV. Conclusion & Future Work

There is a large volume of work related to the streaming data, continuous query, temporal states, and decentralization of data [21][18] [10] . Data streams are always dealt with continuous queries. Our approach differs in a way that we use the bi-temporal concepts in the streaming data and then with an application of continuous query and its transformation. We have found that explicitly designed continuous query languages can be very helpful in sceneries similar to ours. In this paper we have explored how to maintain consistency in sensor data. Different results from the same query at different execution times cause inconsistency. By applying the bi-temporal concepts, we are able to achieve the consistency. Query transformation rules are designed and evaluated to transform the query according to our database design. We are also able to store the query for future use. Alteration for storing has been successfully achieved by applying transformation rules. Properties affecting the consistency have be considered and maintained by introducing a few table design alterations. We have defined a detailed approach for maintaining consistency in the distributed data. Implementation and details protocol suits need to be decided yet. In is paper we have discussed how to solve expressed problem but still there is room for improvement. How to handle error data input from the sensor , We have found that detail of each step to achieve consistency introduces new challenges, therefore validation of proposed solutions through a prototype on the sample data is big task ahead. Prototype will raise many implementation related issues and then we can improve proposed approach in multiple design cycles.

References

[1] S. Madden, "From databases to big data," *IEEE Internet Computing*, vol. 16, no. 3, pp. 4–6, 2012.

[2] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Communications of the ACM*, vol. 57, no. 7, pp. 86–94, 2014.

[3] B. Cui, H. Mei, and B. C. Ooi, "Big data: the driver for innovation in databases," *National Science Review*, vol. 1, no. 1, pp. 27–30, 2014.

[4] B. Wu, D. Lin, D. Chaudhary, L. P. Petrov, and S. Volkov, "Predicting data unavailability and data loss events in large database systems," Nov. 8 2016, uS Patent 9,489,379.

[5] A. Dignös, M. H. Böhlen, J. Gamper, and C. S. Jensen, "Extending the kernel of a relational dbms with comprehensive support for sequenced temporal queries," *ACM Transactions on Database Systems (TODS)*, vol. 41, no. 4, p. 26, 2016.

[6] J. Langseth, F. Aref, J. Alarcon, and W. Lindner III, "Real-time data visualization of streaming data," Jun. 21 2016, uS Patent App. 15/188,975.

[7] H. Qi, X. Chang, X. Liu, and L. Zha, "The consistency analysis of secondary index on distributed ordered tables," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017 IEEE International*. IEEE, 2017, pp. 1058–1067.

[8] A. Margara, J. Urbani, F. Van Harmelen, and H. Bal, "Streaming the web: Reasoning over dynamic data," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 25, pp. 24–44, 2014.

[9] M. Garofalakis, J. Gehrke, and R. Rastogi, "Data stream management: A brave new world," in *Data Stream Management*. Springer, 2016, pp. 1–9.

[10] J. Liu, J. Li, W. Li, and J. Wu, "Rethinking big data: A review on the data quality and usage issues," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 115, pp. 134–142, 2016.

[11] U. Sivarajah, M. M. Kamal, Z. Irani, and V. Weerakkody, "Critical analysis of big data challenges and analytical methods," *Journal of Business Research*, vol. 70, pp. 263–286, 2017.

[12] S. Kumar and R. Rishi, "A relative analysis of modern temporal data models," in *Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on*. IEEE, 2016, pp. 2851–2855.

[13] L. Anselma, P. Terenziani, and R. T. Snodgrass, "Valid-time indeterminacy in temporal relational databases: Semantics and representations," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 12, pp. 2880–2894, 2013.

[14] D. Ryu, "Information content of inter-transaction time: A structural approach," *Journal of Business Economics and Management*, vol. 16, no. 4, pp. 697–711, 2015.

[15] M. Kvet, K. Matiako, and M. Kvet, "Transaction management in fully temporal system," in *Computer Modelling and Simulation (UKSim), 2014 UKSim-AMSS 16th International Conference on*. IEEE, 2014, pp. 148–153.

[16] X. L. Dong and W.-C. Tan, "A time machine for information: Looking back to look forward," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 2044–2045, 2015.

[17] L. Han, L. Huang, X. Yang, W. Pang, and K. Wang, "A novel spatio-temporal data storage and index method for arm-based hadoop server," in *International Conference on Cloud Computing and Security*. Springer, 2016, pp. 206–216.

[18] C. Coronel and S. Morris, *Database systems: design, implementation, & management*. Cengage Learning, 2016.

[19] S. Krishnamurthy, N. Thombre, N. Conway, W. H. Li, and M. Hoyer, "Addition and processing of continuous sql queries in a streaming relational database management system," Jun. 3 2014, uS Patent 8,745,070.

[20] Q. Chen and M. Hsu, "Cut-and-rewind: Extending query engine for continuous stream analytics," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXI*. Springer, 2015, pp. 94–114.

[21] Z. Bao, B. Kimelfeld, and Y. Li, "Automatic suggestion for query-rewrite rules," May 24 2016, uS Patent 9,348,895.

[22] P. Leyshock, D. Maier, and K. Tufte, "Minimizing data movement through query transformation," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 311–316.

[23] J. F. Sequeda, M. Arenas, and D. P. Miranker, "Obda: query rewriting or materialization? in practice, both!" in *International Semantic Web Conference*. Springer, 2014, pp. 535–551.