Query Processing with Materialized Views in a Traceable P2P Record Exchange Framework

Fengrong Li[1] and Yoshiharu Ishikawa[2]

[1] Graduate School of Information Science, Nagoya University
[2] Information Technology Center, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan
lifr@db.itc.nagoya-u.ac.jp, ishikawa@itc.nagoya-u.ac.jp

Abstract. Materialized views which are derived from base relations and stored in the database are often used to speed up query processing. In this paper, we leverage them in a traceable peer-to-peer (P2P) record exchange framework which was proposed to ensure reliability among the exchanged data in P2P networks where duplicates and modifications of data occur independently in autonomous peers. In our proposed framework, the provenance/lineage of the exchanged data can be available by issuing tracing queries. Processing for tracing queries was based on the "pay-as-you-go" approach. The framework can achieve low maintenance cost since each peer only maintains minimum amount of information for tracing. However, the user must pay relatively high query processing cost when he or she issues a query. We consider that the use of materialized views allows more efficient query execution plans. In this paper, we focus on how to incorporate query processing based on materialized views in our framework.

1 Introduction

With the advance of high-performance of computer and the wide spread of high-speed network, *peer-to-peer* (*P2P*) network has become a new paradigm for information sharing. However, unlike the traditional client-server architecture, a P2P network allows a peer to publish information and share data with other peers without going through a separate server computer. It brings us a critical problem; since copies and modifications of data are performed independently by autonomous peers without a specific central server control, it is difficult to determine how data is exchanged among the peers and why the data is located in a peer.

To interpret database contents and to enhance the reliability of data, the notion of *data provenance* is considered very important. Practical and theoretical methodologies for describing, querying, and maintaining provenance information have been proposed, for example in [5, 6]. The importance of understanding the process by which a result was generated is fundamental to many real life applications, such as fields of bioinformatics and archaeology. Without such information, users cannot reproduce, analyze or validate processes or experiments.

Based on the background, to ensure the reliability of data exchanged in P2P networks, we have proposed a *traceable P2P record exchange framework* in [16, 18]. In the framework, a *record* means a tuple-structured data item that obeys a predefined schema globally shared in a P2P network. An important feature of the P2P record exchange framework is that it is based on the database technologies to support the notion of *traceability*. User can trace the lineage of target record by issuing a tracing query. Processing for tracing queries was described in [19].

In this paper, we focus on the issue on how to improve query processing performance by using materialized views. The remainder of this paper is organized as follows. Section 2 describes the fundamental framework of the proposed P2P record exchange system. Section 3 shows the current strategy for query processing and analyzes its problems. Section 4 explains how materialized views are used to improve efficiency in our context for query processing and discusses the maintenance of materialized views. Section 5 reviews the related work. Finally, Section 6 concludes the paper and addresses the future work.

2 Traceable P2P Record Exchange Framework

Figure 1 shows the overview of the traceable P2P record exchange framework proposed in [16, 18], but some terminologies are revised.

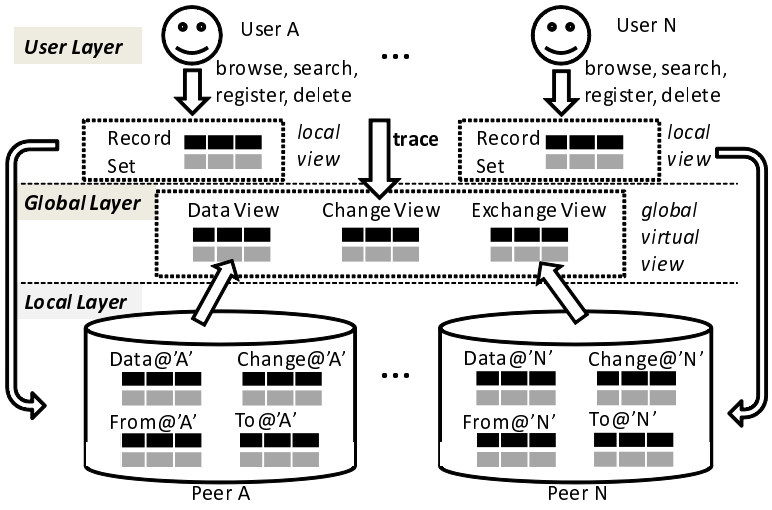

Fig. 1. Traceable P2P Record Exchange Framework

In the framework, we assume that each peer corresponds to a user and maintains the records owned by the user. Each record has the same structure, which is defined by a predefined schema that globally shared within the network. The framework has the following main features:

- In our P2P record exchange framework, every peer can act as a provider and a searcher. Records are exchanged between peers and peers can modify, store, and delete their records independently. Each peer has its own record set in the *user layer*, but their contents are not the same. Peers can behave autonomously and exchange records when required. A peer can find desired records from other peers by issuing a query.
- For reliable data sharing in a P2P network, we want to know, for example, the original creator of the given record and the path of the record in circulation before reaching the current peer. We assume that each peer maintains its own relational tables for storing record exchange and modification histories in the *local layer* and facilitates traceability. All the information required for tracing is maintained in distributed peers. When a tracing query is issued, the query is processed by coordinating related peers in a distributed and recursive manner.
- For ease of understanding and writing tracing queries, we provide an abstraction layer called the *global layer* which virtually integrates all distributed relations and a datalog-like query language [2] for writing tracing queries in an intuitive manner.

In the following, we briefly explain the three-layer model using an example.

User Layer For the ease of presentation, we assume that each peer in a P2P network maintains a `Novel` record set that has two attributes `title` and `author`. Figure 2 shows three record sets maintained by peers A to C. Each peer maintains its own records and wishes to incorporate new records from other peers in order to enhance its own record set. For example, the record `(t1, a2)` in peer A may have been copied from peer B and registered in peer A's local record management system.

Peer A

title	author
t1	a2
t7	a6

Peer B

title	author
t1	a2
t5	a5

Peer C

title	author
t1	a1
t3	a3

Fig. 2. Record Sets in Three Peers

Local Layer In the local layer, each peer maintains minimum amount of information that is required to represent its own record set and local tracing information. In our framework, every peer maintains the following four relations in its local record management system implemented using an RDBMS.

- `Data[Novel]`: It maintains all the records held by peer. Figure 3 shows `Data[Novel]` for peer A. Every record has its own record id for the maintenance purpose. Each record id should be unique in the entire P2P network. Note that there are additional records compared to Fig. 2; they are *deleted* records and usually hidden from the user. They are maintained for data provenance.

– `Change[Novel]`: It is used to hold the creation, modification, and deletion histories. Figure 4 shows an example for peer A. Attributes `from_id` and `to_id` express the record ids before/after a modification. Attribute `time` represents the timestamp of modification. When the value of the `from_id` attribute is the null value (−), it represents that the record has been created at the peer. Similarly, when the value of the `to_id` attribute is the null value, it means that the record has been deleted.

id	*title*	*author*
#A01	t1	a2
#A02	t6	a6
#A03	t7	a6
#A04	t3	a3

Fig. 3. `Data[Novel]`@'A'

from_id	to_id	time
−	#A02	...
#A02	−	...
#A02	#A03	...
−	#A04	...
#A04	−	...

Fig. 4. `Change[Novel]`@'A'

– `From[Novel]`: It records which records were copied from other peers. When a record is copied from other peer, attribute `from_peer` contains the peer name and attribute `from_id` has its record id at the original peer. Attribute `time` stores the timestamp information.

id	from_peer	from_id	time
#A01	B	#B02	...

Fig. 5. `From[Novel]`@'A'

id	to_peer	to_id	time
#A04	C	#C02	...

Fig. 6. `To[Novel]`@'A'

– `To[Novel]`: It plays an opposite role of `From[Novel]` and stores information which records were sent from peer A to other peers. Fig. 6 shows the `To[Novel]` relation of peer A.

Global Layer Three virtual global views are constructed by unifying all the relations in distributed peers. Relation `Data[Novel]` in Fig. 7 expresses a view that unifies all the `Data[Novel]` relations in peers A to C shown in Fig. 2. The `peer` attribute stores peer names. Relation `Change[Novel]` shown in Fig. 8 is also a global view which unifies all `Change[Novel]` relations in a similar manner.

`Exchange[Novel]` shown in Fig. 9 unifies all the underlying `From[Novel]` and `To[Novel]` relations in the local layer. Attributes `from_peer` and `to_peer` express the origin and the destination of record exchanges, respectively. Attributes `from_id` and `to_id` contain the ids of the exchanged record in both peers.

Since recursive processing is required to collect historical information, our framework provides a modified version of *datalog* query language [2]. For ex-

peer	id	*title*	*author*
A	#A01	t1	a2
A	#A02	t6	a6
A	#A03	t7	a6
A	#A04	t3	a3
B	#B01	t1	a1
B	#B02	t1	a2
B	#B03	t5	a5
C	#C01	t1	a1
C	#C02	t3	a3

Fig. 7. View `Data[Novel]`

peer	from_id	to_ id	time
A	–	#A02	...
A	#A02	–	...
A	#A02	#A03	...
A	–	#A04	...
A	#A04	–	...
B	#B01	–	...
B	#B01	#B02	...
B	–	#B03	...
C	–	#C01	...

Fig. 8. View `Change[Novel]`

from_peer	to_peer	from_id	to_id	time
C	B	#C01	#B01	...
B	A	#B02	#A01	...
A	C	#A04	#C02	...

Fig. 9. View `Exchange[Novel]`

ample, the following query detects whether peer X copied the record `(t1, a2)` owned by peer A or not:

```
Reach(P, I1) :- Data[Novel]('A', I2, 't1', 'a2'),
                Exchange[Novel]('A', P, I2, I1, _)
Reach(P, I1) :- Reach(P, I2), Change[Novel](P, I2, I1, _), I1 != NULL
Reach(P, I1) :- Reach(P1, I2), Exchange[Novel](P1, P, I2, I1, _)
Query(I) :- Reach('X', I)
```

Datalog is so flexible that we can specify various types of queries using the three global views; please refer to [16, 18] for the detail.

3 Query Processing Approach and Problem Statement

In our original framework, every peer only maintains the minimum amount information for tracing in the local layer. In order to process tracing queries which are described in datalog using virtual global views, it is necessary to transform the given query to suit the organization of the local layer.

According to the mapping rules [16], the example query in Section 2 can be mapped as follows. The symbol @ is a *location specifier* which indicates the location (peer id) of relation in the local layer.

```
Reach(P, I1) :- Data[Novel]@'A'(I2, 't1', 'a2'),
                To[Novel]@'A'(I2, P, I1, _)
Reach(P, I1) :- Reach(P, I2), Change[Novel]@P(I2, I1, _), I1 != NULL
Reach(P, I1) :- Reach(P1, I2), To[Novel]@P1(I2, P, I1, _)
Query(I) :- Reach('X', I)
```

In [17], we compared two major strategies for datalog query execution, the *seminaive method* and the *magic set method*, in our context. Both of them are

based on the "pay-as-you-go" approach [14] for tracing. It means that we need to aggregate the required historical information from the distributed peers when a tracing query is issued from a user; the user should pay the cost when he or she traces information.

The advantage is that this method is simple and there is no wastefulness in respect of the storage cost. However, when we perform the query processing, since it is necessary to spread a requirement to all the related distributed peers. We should trace the path along the process that the records were exchanged. Generally, the cost for query processing is relatively large. To solve this problem, we consider to construct materialized views which are often used to speed up query processing.

4 Query Processing with Materialized Views

4.1 Definitions of Materialized Views

Materialized views play important roles in databases [12]. In our case, all of the materialized views do not store all of the information in the whole P2P network. They are only used to store the information at the peers in the target scope. A *target scope* is determined by a materialized view maintenance policy. In this paper, we assume that materialized views at each peer store the related information to k hops. Hops means the number of peers involved in a record exchange. For example, if peer A received a record from peer B and peer B received the record from peer C, peer C is in two hops from peer A in terms of the record.

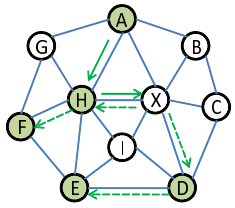

Fig. 10. Target Scope for Peer X ($k = 2$)

For instance, Fig. 10 shows the target scope of the materialized views for peer X in case of $k = 2$. A solid line arrow shows the route of the record that has been copied. A dotted line arrow shows the copy route of the offered records. Peer A, D, E, F, and H are the peers in the scope of the materialized views at peer X since there were record exchanges between them and peer X directly or indirectly in two hops. In this paper, we assume that each peer maintains four materialized views: MVData, MVChange, MVFrom, and MVTo. Each of them corresponds to Data, Change, From, and To relations in the local layer, respectively.

Like a tracing query, materialized views are expressed in datalog and using Data, Change and Exchange virtual views in the global layer. In the following, we show the representations of them in case of $k = 2$.

MVData@X: MVData is a materialized view that stores the exchanged records owned by peers which locate in the target scope. MVData stored at peer X can be described as below:

```
RData1(P, I1, T, A, H) :- Data[Novel]('X', I2, T, A),
                          Exchange[Novel](P, 'X', I1, I2, _), H=1
RData1(P, I1, T, A, H) :- RData1(P, I2, T, A, H), Change[Novel](P, I1, I2, _)
RData1(P, I1, T, A, H) :- RData1(P1, I2, T, A, H1),
                          Exchange[Novel](P, P1, I1, I2, _), H=H1+1, H<=2
RData2(P, I1, T, A, H) :- Data[Novel]('X', I2, T, A),
                          Exchange[Novel]('X', P, I2, I1, _), H=1
RData2(P, I1, T, A, H) :- RData2(P, I2, T, A, H), Change[Novel](P, I1, I2, _)
RData2(P, I1, T, A, H) :- RData2(P1, I2, T, A, H1),
                          Exchange[Novel](P1, P, I2, I1, _), H=H1+1, H<=2
RData(P, I, T, A) :- RData1(P, I, T, A, H)
RData(P, I, T, A) :- RData2(P, I, T, A, H)
MVData@X(P, I, T, A) :- RData(P, I, T, A)
```

The variable H is used to count the number of hops. The maximum value of H should be set to be equal to k. `RData1` is the collection of the records related to the copied record owned by peer X in two hops. `RData2` stores the information that which peer copied the records owned by peer X in two hops and also stores the contents of records in these peers. `RData1` and `RData2` are finally combined into `MVData@X`. Peer X executes the program and stores DataMV@X as a materialized view.

MVChange@X: The following is the definition of the materialized view for storing the change histories of the exchanged records:

```
RPeer1(P, I1, T, A, H) :- Data[Novel]('X', I2, T, A),
                          Exchange[Novel](P, 'X', I1, I2, _), H=1
RPeer1(P, I1, T, A, H) :- RPeer1(P, I2, T, A, H), Change[Novel](P, I1, I2, _)
RPeer1(P1, I1, T, A, H) :- RPeer1(P2, I2, T, A, H1),
                           Exchange[Novel](P1, P2, I1, I2, _), H=H1+1, H<=2
RChange1(P, I1, I2, _, H) :- RPeer1(P, _, _, _, H), Change[Novel](P, I1, I2, _)
RPeer2(P, I1, T, A, H) :- Data[Novel]('X', I2, T, A),
                          Exchange[Novel]('X', P, I2, I1, _), H=1
RPeer2(P, I1, T, A, H) :- RPeer2(P, I2, T, A, H), Change[Novel](P, I1, I2, _)
RPeer2(P, I1, T, A, H) :- RPeer2(P1, I2, T, A, H1),
                          Exchange[Novel](P1, P, I2, I1, _), H=H1+1, H<=2
RChange2(P, I1, I2, _, H) :- RPeer2(P, _, _, _, H), Change[Novel](P, I1, I2, _)
RChange(P, I, I1, _) :- RChange1(P, I, I1, _, H)
RChange(P, I, I1, _) :- RChange2(P, I, I1, _, H)
MVChange@X(P, I, I1, _) :- RChange(P, I, I1, _)
```

Both MVData and MVChange will increase the cost for storage and management in the operation and maintenance of materialized views. But they not only are used to improve query processing efficiency, but also can be used for the recovery of the lost data when some peer disappeared suddenly.

MVFrom@X: This materialized view stores the information of copied records from other peers in two hops.

```
FromH(I2, P, I1, H) :- Data[Novel]('X', I2, T, A),
                       Exchange[Novel](P, 'X', I1, I2, _), H=1
FromH(I2, P, I3, H) :- FromH(I1, P, I2, H), Change[Novel](P, I3, I2, _)
FromH(I2, P, I1, H) :- FromH(I1, P, I2, H1),
                       Exchange[Novel](P, P1, I1, I2, _), H=H1+1, H<=2
MVFrom@X(I, P, I1, H) :- FromH(I, P, I1, H)
```

MVFrom is effective for tracing the record retrospectively. Management cost is negligible though the storage cost is additionally needed. Path information caching and the record insertion to the materialized view can be executed one when the record is exchanged.

MVTo@X: A similar idea can be applied to the side of the To relation. This materialized view stores the information that which peer copied records from peer X in the scope of two hops .

```
ToH(I2, P, I1, H) :- Data[Novel]('X', I2, T, A),
                     Exchange[Novel]('X', P, I2, I1, _), H=1
ToH(I2, P, I3, H) :- ToH(I1, P, I2, H),
                     Change[Novel](P, I2, I3, _), I3 != NULL
ToH(I2, P, I1, H) :- ToH(I1, P1, I2, H1),
                     Exchange[Novel](P1, P, I2, I1, _), H=H1+1, H<=2
MVTo@X(I, P, I1, H) :- ToH(I, P, I1, H)
```

The management cost for the MVTo is not negligible. For example, in Fig. 10, when the record is copied from peer D to peer E, it is necessary to pass on the information of the copy event to peer X. In other words, not only peer D and E but also peer X should be involved in the transaction of the copy of the record from peer D to peer E. This becomes an additional overhead to some extent.

4.2 Query Processing with Materialized Views

Materialized views stored locally as base relations can improve query performance through query rewrites. We illustrate their use in query processing using our example.

Based on the materialized views, we can rewrite the example query in Section 2 as below:

```
Reach(P, I1) :- Data[Novel]@'A'(I2, 't1', 'a2'),
                To[Novel]@'A'(I2, P, I1, _)
Reach(P, I1) :- Reach(P1, I2), MVTo[Novel]@'A'(I2, P, I1, _)
Reach(P, I1) :- Reach(P, I2), MVChange[Novel]@P(I2, I1, _), I1 != NULL
Reach(P, I1) :- Reach(P1, I2), MVTo[Novel]@P1(I2, P, I1, _)
Query(I) :- Reach('X', I)
```

In this example, peer A wants to detect that whether peer X copied the record (t1, a2) or not. In the following, we describe the query processing referring to the Fig. 10.

In our original approach without materialized views, the query processing which is based on the seminaive method starts at peer A and the query fragments generated at peer A are first forwarded to peer H, and then peer H forwards them to Peer X. The query is executed in this way until it reaches the fixpoint.

With the materialized views, peer A can do the execution locally since peer X copied the record (t1, a2) in the target scope of peer A. Assuming that materialized view MVTo about the record (t1, a2) shown in Fig. 11 is stored at peer A. At peer A, notice that when the first rule is executed, the result (H, #H01) will become a tuple in `Reach` which is shown in Fig. 12. When the second rule is executed, a new tuple (X, #X01) should be inserted in `Reach`. Finally, when the last rule is executed, the #X01 as a result should return to the query. That is to say, peer X copied the record (t1, a2) from peer A.

id	to_peer	to_id	#hop
#A01	H	#H01	1
#H01	X	#X01	2

Fig. 11. Relation `MVTo` at Peer A

P	I
H	#H01
X	#X01

Fig. 12. Relation `Reach`

The example indicates that the materialized views speed up the query processing. Like this case, with the materialized views, processing for queries which include recursive operation does not have to do the forward processing and queries can be answered immediately at the local peer. It is thought that the materialized views work well effectively in the query processing. Of course, processing for many tracing queries still needs forwarding operations until it reaches the fixpoint based on the seminaive method. If the query processing can not be done using materialized views, then it will be executed ordinarily as before.

4.3 Maintenance for Materialized Views

View maintenance which means the processing of updating a materialized view in response to changes to the underlying data. As we described above, materialized views can speed up query processing greatly. But they have to be kept up to date. If some of the base relations are changed, materialized views must be recomputed to ensure correctness of results to query processing. For maintaining general recursive views, [13] proposed the *DRed* (Delete and Rederive) algorithm that can handle incremental updates. However, the algorithm assumes a centralized environment, and it is quite costly to apply the algorithm in our context because the maintenance process is propagated among distributed peers. *Materialized view maintenance* problem in deductive databases was described in [12, 21].

In our case, we can utilize a feature of our framework. Every update can be handled as a tuple insertion. Assume that the related peer in the first hop

for peer X is peer Y, and peer Z is related peer in the second hop. Of course, related peers for peer Z in two hops are peer Y and peer X. Depending on the update types, a record is inserted in each of the following local relations and materialized views:

- record update in peer X: `Data@X`, `Change@X`, `MVData@Y`, `MVData@Z`, `MVChange@Y`, and `MVChange@Z`
- record modification in peer X: `Data@X`, `Change@X`, `MVData@Y`, `MVData@Z`, `MVChange@Y`, and `MVChange@Z`
- record deletion in peer X: `Change@X`, `MVChange@Y`, and `MVChange@Z`
- record copy from peer X to peer Y: `To@X`, `From@Y`, `Data@Y`, `MVData@X`, `MVData@Z`, `MVTo@X`, and `MVFrom@Y`

It means that there only exists *insertion* in the database updates in our framework. Materialized view maintenance for insertion can use the seminaive method for computation easily. When the fixpoint is found, the current incremental maintenance should be finished [13].

5 Related Work

Understanding provenance of documents is not a new problem. The importance of provenance goes well beyond verification. It is used in a wide range of fields, including data warehousing [7], uncertain data management [3, 22], curated databases [5], and other scientific fields such as bioinformatics [4]. In this area, one of the well-known projects would be the *Trio* project at Stanford University, in which both of the uncertainty and lineage issues are considered [22]. Our research is devoted to the data provenance issue in P2P information exchange, where data provenance is important but there is few proposals for this topic.

There are a variety of research topics regarding P2P databases, such as coping with heterogeneities, query processing, and indexing methods [1]. One related project with our problem is the ORCHESTRA project [9, 15], which aims at collaborative sharing of evolving data in a P2P network. In contrast to their approach, our research focuses on a simple record exchange scenario and does not consider schema heterogeneity. One of the features of our framework is to employ database technologies as the underlying foundation to support reliable P2P record exchange.

As proved in the *declarative networking* project [20], declarative recursive queries are very powerful in writing network-oriented database applications such as sensor data aggregation. In contrast to their approach, our focus is compact and understandable tracing query specifications.

Materialized views can be used to summarize, pre-compute, and replicate data. Maintenance for them is very important in database. We can find the recent survey about maintenance of materialized views in [11]. The incremental maintenance of views has received a lot of attention in database research, many incremental methods have been already proposed in the literature[8, 10, 21, 23].

In all papers, only [12, 21] described materialized view maintenance problem in deductive databases. In our research, for tracing queries, especially for the queries asking past histories, materialized views [12] are quite helpful to reduce query response time. For that purpose, we develop a query processing method which effectively uses materialized views and a view selection and maintenance method which considers the trade-off of cost and benefit.

6 Conclusions and Future Work

For the efficient query processing, data replication and caching are popular techniques. Considering practical requirements of tracing, we added some incorporate additional features and constructs to our fundamental P2P record exchange system. Although the storage and maintenance cost will increase, the query processing cost can be reduced.

In this paper, we described how to define materialized views and how to use them to improve query processing in our proposed P2P record exchange system. The maintenance of materialized views was also discussed. Nevertheless much work remains to be done. We need to consider how to take trade-off considering the total cost reduction. Several future research issues are summarized as follows:

- Full specification of complete query processing strategies: We need to enhance the strategies to handle more complex tracing queries. The effectiveness and limitation of the declarative language-based approach will become more clear.
- Fault-tolerance: In this paper, we omitted the issue of fault tolerance, but it is important for supporting P2P networks in which failure occurs frequently. We need to consider that how to use materialized views to do the data recovery for left peer in detail.
- Prototype system implementation and experiments: We are currently developing a prototype system of our P2P record exchange framework. Also, we started to construct a P2P network simulator that can be used for simulating our prototype system in a virtual P2P network. Their developments will have positive feedbacks to improve our fundamental framework.

Acknowledgments

This research was partly supported by the Grant-in-Aid for Scientific Research (#21013023, #22300034) from the Japan Society for the Promotion of Science (JSPS).

References

1. K. Aberer and P. Cudre-Mauroux. Semantic overlay networks. In *VLDB*, 2005. (tutorial notes).

2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. VLDB*, pp. 953–964, 2006.
4. D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. VLDB*, pp. 900–911, 2004.
5. P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *Proc. ACM PODS*, pp. 1–12, 2008.
6. P. Buneman and W.-C. Tan. Provenance in databases (tutorial). In *Proc. ACM SIGMOD*, pp. 1171–1173, 2007.
7. Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *Proc. VLDB*, pp. 471–480, 2001.
8. J. Goldstein and P.-A. Larson. Optimizing queries using materialized views: A practical, scalable solution. In *Proc. ACM SIGMOD*, pp. 331–342, 2001.
9. T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: Facilitating collaborative data sharing. In *Proc. ACM SIGMOD*, pp. 1131–1133, 2007.
10. T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. ACM SIGMOD*, pp. 328–339, 1995.
11. A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, 1995.
12. A. Gupta and I. S. Mumick eds. *Materialized Views*. MIT Press, 1999.
13. A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *Proc. ACM SIGMOD*, pp. 157–166, 1993.
14. A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *Proc. ACM PODS*, pp. 1–9, 2006.
15. Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *Proc. Conf. on Innovative Data Systems Research (CIDR 2005)*, pp. 107–118, 2005.
16. F. Li, T. Iida, and Y. Ishikawa. Traceable P2P record exchange: A database-oriented approach. *Frontiers of Computer Science in China*, 2(3):257–267, 2008.
17. F. Li, T. Iida, and Y. Ishikawa. 'Pay-as-you-go' processing for tracing queries in a P2P record exchange system. In *Proc. DASFAA*, Vol. 5463 of *LNCS*, pp. 323–327, 2009. A long version is available from http://www.db.itc.nagoya-u.ac.jp/papers/2009-dasfaa-li-long.pdf.
18. F. Li and Y. Ishikawa. Traceable P2P record exchange based on database technologies. In *Proc. APWeb*, Vol. 4976 of *LNCS*, pp. 475–486, 2008.
19. F. Li and Y. Ishikawa. Query processing in a traceable P2P record exchange framework. *IEICE Transactions on Information and Systems*, E93-D(6), 2010. (accepted for publication).
20. B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: Language, execution and optimization. In *Proc. ACM SIGMOD*, pp. 97–108, 2006.
21. M. Staudt and M. Jarke. Incremental maintenance of externally materialized views. In *Proc. VLDB*, pp. 75–86, 1996.
22. J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. Conf. on Innovative Data Systems Research (CIDR 2005)*, pp. 262–276, 2005.
23. Y. Zhuge, H. García-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proc. ACM SIGMOD*, pp. 316–327, 1995.