Revisit On View Maintenance In Data Warehouses

Weifa Liang[†] Jeffrey X. Yu[‡]

[†] Department of Computer Science,The Australian National University
Canberra, ACT 0200, Australia
[‡] Department Systems Engineering and Engineering Management
Chinese University of Hong Kong, Hong Kong

Abstract. The complete consistence maintenance of SPJ-type materialized views in a distributed source environment has been studied extensively in the past several years due to its fundamental importance to data warehouses. Much effort has been taken based on an assumption that each source site contains only one relation and no multiple appearances of a relation is allowed in the definition of views. In this paper a generalized version of the view maintenance problem that not only a relation may appear many times in the definition of the view but also a site may contain multiple relations is considered. Due to unpredictability of the communication delay and bandwidth between the data warehouse and the sources, the materialized view maintenance is very expensive and time consuming. Therefore, one natural question for this generalized case is whether there is an algorithm which not only keeps the view complete consistent with the remote source data but also minimizes the number of accesses to the remote sites. In this paper we first show that a known `SWEEP` algorithm is one of the best algorithms for the case where multiple relations are included in a site. We then propose a complete consistency algorithm which accesses remote sources less than $n-1$ times for the case where multiple appearances of a relation is allowed and n is the number of relations in the definition of the view.

1 Introduction

Data warehouses store materialized views in order to provide fast access to the integrated data from various local and/or remote sources [4]. They can be used as an integrated and uniform basis for decision support, data mining, data analysis, and ad-hoc querying across the source data. One of the fundamental problems in data warehousing is the view maintenance problem which aims at maintaining the content of a materialized view at a certain level of consistency with the source data when the updates commit at the remote sources.

Many incremental maintenance algorithms for materialized views have been introduced for centralized database systems [2, 7, 5]. There are also a number of similar studies in the distributed environments [3, 8, 12, 14, 15, 9]. These previous works have formed a spectrum of solutions ranging from a fully virtual approach at one end where no data is materialized and all user queries are answered by interrogating the source data [8], to a full replication at the other end where the whole databases at the sources are copied to the warehouse so that the view refreshments (updates) can be handled in the warehouse locally [6, 11]. In order to keep the content (data) of a view consistent with the source data, the approach most commonly used is to maintain the warehouse at night time when the warehouse is not available to users, while the user query can be processed in the day time when the maintenance transactions are not running. Following [14, 1], we here only consider maintaining materialized views through reducing the number of accesses to the remote site without using any auxiliary views. The complete consistence maintenance of a SPJ-type materialized view has been investigated in the past several years [14, 1, 10]. All these previous studies are based on the assumption that each source site contains only one relation and no multiple appearances of a relation are allowed in the definition of the view. In this paper we consider a generalized case which allows that (i) a relation may appear many times in the definition of the view; and (ii) a site may contain multiple relations. We first show that the `SWEEP` algorithm is one of the best such algorithms to make the view complete consistent. In other words, there is no maintenance algorithm which accesses the remote sites less than $n-1$ times but keeps the view complete consistent with the source data. We then propose a complete consistency algorithm which accesses the remote source less than $n-1$ times to evaluate the view update for the case with multiple identical relations in the definition of the view.

2 Preliminaries

A *data warehouse* is a repository of integrated data which collects and maintains a large amount of data from multiple distributed, autonomous and possibly heterogeneous sources. A typical data warehouse architecture is defined in [13]. On the source side, associated with each source there is a monitor/wrapper which collects the data of interest and sends the data to the warehouse. The monitor/wrapper is responsible for identifying changes and notifying the warehouse. On the data warehouse side, there is an integrator which receives the source data, performs necessary data integration and translation, adds any extra desired information such as the timestamps for historical analysis, and requests the warehouse to store the data. In effect, the warehouse caches a materialized view of the source data. The data is then readily available to user applications for querying and analysis. The communication between the source and data warehouse is assumed to be reliable FIFOs, i.e., messages are not lost on their way between the source and the warehouse and delivered in the order in which they originally are sent. No communication is imposed between any two sources. In fact, they may not be able to communicate with each other.

A warehouse state (source state) ws (ss) represents the content of the data warehouse (sources) at a given time point. The warehouse state changes whenever one of the views in it is updated. A source state ss_j is a vector that contains n components, and the ith component, $ss_j[i]$ is the state of source i which represents the content of source i at that given time point. Let $ws_0, ws_1, \ldots, ws_f$ be the sequence of the warehouse states after a series of source updates $ss_0, ss_1, \ldots, ss_q$, V be a materialized view, and $V(ws_j)$ be the content of V at warehouse state

ws_j. Assume that V is derived from n sources, where each source has a unique identity i, $1 \leq i \leq n$. Further, assume that source updates are executed in serializable fashion across sources, and ss_q is the final state of the sources. $V(ss_j)$ is the result of computing the view V over the source state ss_j. That is, $V(ss_j)$ is evaluated over all R_i at the state $ss_j[i]$, $1 \leq i \leq n$. Given a materialized view V, V is *complete consistent* with the source data if, for every source state ss_i, there is a corresponding warehouse state ws_j such that $V(ss_i) = V(ws_j)$. Assume that initially $V(ws_0) = V(ss_0)$. That is, there is a complete order preserving mapping between the states of the view and the states of the sources.

In this paper we focus on the maintenance of SPJ materialized views such that they are completely consistent with the remote source data. Let V be a SPJ-type view derived from n relations $R_1, R_2, \ldots, R_n$, defined as follows.

$$V = \pi_X \sigma_P (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n), \tag{1}$$

where X is the set of projection attributes, P is the selection condition which is the conjunction of disjunctive clauses, and R_i is in a remote source j, $1 \leq j \leq K$ ($K \leq n$). Updates to the source data are either insertions or deletion of tuples. A modification is treated as a deletion followed by an insertion. Furthermore, the views in the warehouse are based on bag semantics.

3 Multiple Relations Within A Single Site

In this section we consider the complete consistency maintenance of materialized views when a site contains multiple relations. As mentioned by its authors, the `SWEEP` algorithm [1] can be easily extended to deal with this case, which treats a site containing d relations ($d > 1$) as d different sites virtually. Thus, the evaluation of an update to the materialized view is implemented through visiting the site at least d times. To avoid multiple visits to a site, one natural question is to ask whether it is possible to evaluate the update by visiting each site just once instead of d times as suggested by the `SWEEP` algorithm? In the following we dedicate ourselves to answer this question. Unfortunately, we show that there is no algorithm that keeps the view complete consistent with the source data by visiting each site just once. Therefore, the `SWEEP` algorithm is one of the best algorithms to achieve the complete consistency.

It is well known that each SPJ type view can be represented by an undirected join-graph $G(N, E)$, where each node in N represents a relation, and there is an edge between two nodes if there is a join condition which only involves two relations. Note that each appearance of a relation in the definition of the view is treated as a different node in the graph. Let $G[S]$ be an induced subgraph by a subset of nodes S ($S \subseteq N$). An auxiliary view $V_S = \bowtie_{R_{i_j} \in S} R_{i_j}$ can be defined using S. If $G[S]$ is disconnected, then V_S is a *direct* product of the $|S|$ relations, which is usually avoided because this leads to very high join cost. Let S_i be the set of relations used by V in site i with $|S_i| > 1$, $1 \leq i \leq K$.

Theorem 1. *Given a SPJ view with multiple relations from a site, there is no algorithm that visits the site just once and keeps the view completely consistent with the remote source data.*

Proof. Assume that there is a source update δR_i at source i. To respond the update, there is a view update δV to be evaluated. Without loss of generality, assume that the site j contains $|S_j|$ relations and $G[S_j]$ is connected. Otherwise, it can be applied to each connected component of $G[S_j]$.

Let $\delta V_{i,j-1}$ be the partial result of δV after having visited sites containing $R_{i+1}, R_{i+2}, \ldots$, and R_{j-1}, where δR is either an insertion ΔR or a deletion ∇R. Now it is about to visit site j at the first time. In order to visit this site just once, all evaluation involved the relations at site j must be done through this visit. For simplicity, let $S_j = \{R_{j_1}, R_{j_2}, R_{j_3}\}$. Assume that R_{j_1} and R_{j_3} have updated δR_{j_1} and δR_{j_3} during the period from t_1 to t_2, where t_1 is the moment that the data warehouse starts evaluating the source update δR_i and t_2 is the moment that site j received the query of evaluating the partial result of δV and $\delta V_{i,j-1}$. Clearly $t_1 < t_2$. Since site j is visited just once, the evaluation involving R_{j_2} and R_{j_3} must be considered too during this visit. Let $\delta V'_{i,j}$ be the partial result received by the data warehouse at t_3 after finishing the evaluation at site j, which has been contaminated, where

$$\begin{aligned}\delta V'_{i,j} &= \delta V_{i,j-1} \bowtie V_{S_j} = \delta V_{i,j-1} \bowtie (R_{j_1} + \delta R_{j_1}) \bowtie R_{j_2} \bowtie (R_{j_3} + \delta R_{j_3}) \\ &= \delta V_{i,j-1} \bowtie (R_{j_1} \bowtie R_{j_2} \bowtie R_{j_3}) + \delta V_{i,j-1} \bowtie (R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}) \\ &+ \delta V_{i,j-1} \bowtie (\delta R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}) + \delta V_{i,j-1} \bowtie (\delta R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}). \quad (2)\end{aligned}$$

Note that the contamination can be cleaned up using the logs of R_{j_1}, R_{j_2} and R_{j_3} in UMQ from t_1 to t_3. One important fact is the data warehouse does not have the knowledge of the subsequent updates δR_{j_1} and δR_{j_3} at the moment it issues the query $\delta V_{i,j-1} \bowtie R_{j_1} \bowtie R_{j_2} \bowtie R_{j_3}$ to site j. So, it is impossible at that time for it to send queries to the site in order to evaluate $R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}$, $\delta R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}$, and $\delta R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}$, while $\delta V_{i,j} = \delta V'_{i,j} - \delta V_{i,j-1} \bowtie (R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}) - \delta V_{i,j-1} \bowtie (\delta R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3}) - \delta V_{i,j-1} \bowtie (\delta R_{j_1} \bowtie R_{j_2} \bowtie \delta R_{j_3})$. Thus, there is no way to remove the contaminated data if the algorithm allows to visit each site just once. If $G[S_j]$ is disconnected, then for each connected component $G[S_{j_i}]$, an auxiliary view $V_{S_{j_i}}$ can be derived, and the above argument can be applied. Therefore, the theorem follows. ■

4 An SPJ View With Multiple Appearance of Relations

In this section we consider the complete consistency view maintenance for the case where multiple appearances of a relation are allowed in the definition of the view. We start with a simple algorithm for it. Basically, the simple algorithm is similar to the `SWEEP` algorithm except some minor modifications. Assume that the source update is δR_i, the update δV to V will be evaluated by visiting the other sources (except R_i) one by one. Let $\delta V_{j-1,i}$ be the partial result of δV after visiting distinct remote relations (sources) $R_{i-1}, R_{i-2}, \ldots$, and R_{j-1}. Now the simple algorithm is about to visit next source R_j. If R_j is a copy of R_i, then the partial result $\delta V_{j,i}$ of the update δV becomes $\delta V_{j,i} = \delta V_{j-1,i} \bowtie \delta R_i$, which can be evaluated at the data warehouse without querying the remote source containing R_i because δR_i which is located in the queue UMQ, is available locally. Otherwise, assume that a relation $R_{j'}$ has been visited before ($j < j' <$

i) and R_j is a copy of $R_{j'}$ ($R_{j'} = R_j$), the evaluation of the partial result $\delta V_{j,i} = \delta V_{j-1,i} \bowtie R_j$ is done exactly as it is done in the SWEEP algorithm, i.e., the algorithm treats R_j as a different $R_{j'}$ and visits the site containing R_j again.

Having finished the evaluations of $\delta_{1,i}V$ and $\delta_{i,n}V$, $\delta V = \delta_{1,i}V \bowtie \delta_{i,n}V$ is evaluated locally and merged to V, and the updated view is complete consistent with the remote source data, which is stated in the following lemma. The proof of this lemma is similar to that for SWEEP, omitted.

Lemma 1. *For a given SPJ-type view with multiple appearances of a relation in its definition, the view maintained by the simple algorithm is complete consistent with the remote source data.*

4.1 An improved algorithm

Unlike the case where multiple relations are located in the same site for which we have shown that $O(n)$ accesses to remote sites is needed, we here show that the number of accesses to remote sources can be reduced by visiting each site only once. If V consists of K distinct relations, then the simple algorithm still requires $O(n)$ accesses to the remote sources. In the following an improved algorithm is presented, which aims to improve the number of accesses to the remote sources.

Following the assumption that there are only K distinct relations among the n relations in the definition of V, let a distinct relation R_{j_i} appear k_{j_i} times in the definition of V. Clearly $\sum_{i=1}^{K} k_{j_i} = n$. Let $Predicate(R_{j_i})$ be the set of join condition clauses in the definition of V, associated with R_{j_i}. It is obvious that $|Predicate(R_{j_i})| = k_{j_i} \leq K$. Let $All_{R_{j_i}}$ be the set of attributes in R_{j_i} and $C_{R_{j_i}} = \{A \mid A$ is an attribute of R_{j_i}, A is appeared in Q, and $Q \in Predicate(R_{j_i})\}$ be a subset of attributes of R_{j_i} which includes the attributes of R_{j_i} in the join condition clauses of V. Denote by $P_{R_{j_i}} = (All_{R_{j_i}} \cap X) \cup C_{R_{j_i}}$, the set of projection attributes of R_{j_i} which must be included in the schema definition of the incremental update expression δV of V.

Assume that k_i is the number of appearances of R_{i_1} in the definition of V. Let $R_{i_1} = R_{i_2} = \ldots = R_{i_{k_i}}$. The key idea behind the proposed algorithm is that one of the k_i copies of R_{i_1} will serve as the *representative* of its other $k_i - 1$ copies. To evaluate an update δV due to a source update, the remote site is visited by only accessing the representative of a relation. For the subsequent evaluation of δV, if the relation is met again, the algorithm does not need to access the remote site containing the relation again, and evaluates the update, based solely on the previously partial result in the data warehouse. As results, only K accesses to the remote sites are needed in order to update V.

The proposed algorithm is presented as follows. If a relation is appeared only once in the definition of V, its treatment during evaluation is exactly the same as that in the SWEEP algorithm. Otherwise, let a relation R_j with multiple copies have been scanned first during the evaluation of δV. Then, R_j will become the representative of the other copies of the relation. The role of a representative is as follows. When dealing with the joining of R_j and the previous partial result of δV, it is required that the schema of the partial result of δV is not only just including those attributes of R_j in X (i.e., the attributes in $X \cap All_{R_j}$ based on the current join condition) but also including the other attributes of R_j in P_{R_j}. With the update evaluation continues, assume that the next scanning source is R_k which is a copy of the representative R_j. This time the data warehouse is able to evaluate the partial result of the update to V, using only the current partial result in the warehouse and the join condition associated with R_k without querying the remote site containing R_k. Thus, the number of accesses to the remote sites will be determined by the number of representatives of relations in the definition of V, instead of $n-1$ in the simple algorithm. In the following we show that the view maintained by the improved algorithm is complete consistent with the source data.

Theorem 2. *Let V be the joins of n relations with some of the relations are identical. Assume that $V = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$ and an update δR_i at a remote site R_i, then the evaluation an update δV to V to respond the source update proceeds. Let $\delta V_{k,i} = \delta R_i \bowtie \ldots \bowtie R_j \bowtie \ldots \bowtie R_k$ be the partial result of the update δV. During the evaluation of δV, assume R_k is being scanned and R_k is a copy of R_j that had been visited before, then $\pi_{A_{R_j}} \rho_{P_{R_j}} \delta V_{k,i} = \pi_{A_{R_j}} \rho_{P_{R_j}} (\delta V_{k-1,i} \bowtie R_k)$, i.e., $\delta V_{k,i} = \pi_{A_{R_j}} \rho_{P_{R_j}} \delta V_{k-1,i}$.*

Proof. For the simplicity, we assume that there is only a duplicate relation R_j in the definition of V. If there are more than 2 duplicates of R_j or multiple identical relations, it can be proven inductively. To respond a source update δR_i, the update δV to the view is as follows.

If R_i is the duplicate ($R_j = R_i$), δR_i will be used to replace all appearances of R_i in the definition of V, and there is no difference from the SWEEP algorithm, so, the theorem holds. Otherwise, the duplicate R_j is different from R_i. Following the SWEEP algorithm, to evaluate δV, it scans the other sources from both sides. One is from its current position to the left, and scans $R_{i-1}, R_{i-2}, \ldots, R_1$; another is from the current position R_i to the right, and scans $R_{i+1}, R_{i+2}, \ldots, R_n$. During the scanning, the two copies of R_j will appear in one of the following three positions. (i) Both copies are appeared at the left side. In this case, the first appearance of R_j will be its representative. Let R_k be the second appearance of R_j, then the evaluation of $\delta V_{j,i}$ is similar to its corresponding counterpart in the SWEEP algorithm except it includes more attributes of R_j by our definition, and the evaluation of $\delta V_{k,i}$ can be carried out in the data warehouse using $\delta V_{k-1,i}$ and the current condition associated with R_k. (ii) One copy of R_j is appeared at the left side and another is appeared at the right side. For this case, assume that the evaluation of δV starts from the left, then the representative of R_j is at the left side. The evaluation of δV then moves to the right, and the second appearance R_k of R_j will be at the right side. (iii) Both copies are appeared at the right side, which is similar to case (i), omitted.

We now focus on case (i). To evaluate $\delta V_{k,i}$, it can be done by sending a query $\delta V_{k-1,i} \bowtie R_j$ to the site containing R_j. Let $\delta V'_{k,i}$ be the partial result returned from the site after evaluating the query, which may contain the contaminated data. To obtain $\delta V_{k,i}$, the algorithm will offset the contaminated data by evaluating $\delta V_{k,i} = \delta V'_{k,i} - (\delta V_{k-1,i} \bowtie \cup_{j_i=j} \delta R^x_{j_i})$ at the data warehouse, where $\delta R^x_{j_i}$

is one of the updates to R_j occurred from the moment of δR_i occurred to the moment that $V'_{k,i}$ is received by the data warehouse. Let $\delta V''_{k,i} = \pi_{c_k(R_k)}\delta V_{k-1,i}$, where $c_k(R_k)$ is the current join condition associated with R_k. Then we have $\delta V_{k,i} = \delta V''_{k,i}$. To prove this claim, we first show that $t \in \delta V_{k,i}$ for any tuple $t \in \delta V''_{k,i}$. Let $t_{c_j}[R_j]$ be the set of attributes of R_j associated with the current join condition $c_j(R_j)$. From now on we abuse this concept further, $t_{c_j}[R_j]$ is also used to represent the values of these related attributes in tuple $t \in \delta V''_{k,i}$. Thus, for any tuple $t \in \delta V''_{k,i}$, both $t_{c_j}[R_j]$ and $t_{c_k}[R_k]$ are true. Then, we say that tuple t must be in $\delta V_{k,i}$, because $t_{c_j}[R_j]$ is true in $\delta V_{k-1,i}$ and $t_{c_k}[R_k]$ is true by the current join condition. Therefore, $\delta V''_{k,i} \subseteq \delta V_{k,i}$.

We then show that for any tuple $t' \in \delta V_{k,i}$, it must be in $\delta V''_{k,i}$ too. Since $t' \in \delta V_{k,i}$, $t'_{c_k}[R_k]$ must be true. While t' is obtained through the join of $\delta V_{k-1,i}$ and R_k, $t'_{c_j}[R_j]$ must be true due to that it is in $\delta V_{k-1,i}$. Thus, $t' \in \delta V''_{k,i}$. We therefore have $\delta V_{k,i} \subseteq \delta V''_{k,i}$. The theorem then follows. ■

We now can see that the maintenance time of V is improved dramatically because the number of remote accesses is $O(K)$, not $O(n)$. It must also be mentioned more space for the storage of the partial result is needed due to the fact that all attributes in $\cup_{R\in V} C_R$ must be included in the schema definition of the partial result, while they may not be included in the schema of δV in the `SWEEP` algorithm. Apart from that, in the `SWEEP` algorithm, the evaluation of $\delta V_{1,i}$ and $\delta V_{i,n}$ can be done concurrently. The proposed algorithm, however, cannot achieve this parallelism because there is only one representative for the multiple copies of a relation. In the following we extend our algorithm and make it have such kind of parallelism.

4.2 An extension with concurrency

To respond a source update δR_i, the proposed algorithm first evaluates $\delta V_{1,i} = R_1 \bowtie \ldots \bowtie R_{i-1} \bowtie \delta R_i$, then evaluates $\delta V_{i,n} = \delta R_i \bowtie R_{i+1} \bowtie \ldots \bowtie R_n$, and finally evaluates $\delta V = \delta V_{1,i} \bowtie \delta V_{i,n}$ and merges the result δV to V. Now, for the given δR_i, let V_L (V_R) be the set of distinct relations in the sequence of $R_1, R_2, \ldots, R_{i-1}$ ($R_{i+1}, R_{i+2}, \ldots, R_n$). We now extend the proposed algorithm to make it have the parallelism. That is, the algorithm proceeds the evaluation of $\delta V_{1,i}$ and $\delta V_{i,n}$ independently. Thus, there are two representatives for each $R \in V_L \cap V_R$. One is used for the evaluation of $\delta V_{1,i}$; another is used for the evaluation $\delta V_{i,n}$. It is not difficult to show that the updated view after merging $\delta V = \delta V_{1,i} \bowtie \delta V_{i,n}$ to V is completely consistent with the source data. We therefore have the following lemma.

Lemma 2. *Assume that there are K distinct relations in the definition of V consisting of n relations. To evaluate the update δV to V concurrently due to a source update δR_i, the number of accesses to the remote sources is as follows. If $V_L \cap V_R = \emptyset$, then the number of remote source accesses is no more than K; otherwise, the number of remote source accesses is at most $K + |V_L \cap V_R| \leq 2K$.*

Proof. If $V_L \cap V_R = \emptyset$, then, there is no relation with multiple appearances at the both sides (the left and the right sides) when evaluating the update to the view. Thus, all the other copies of a relation must be at the same side, and one of the copies will serve as the representative of the relation. Due to $V_L \cap V_R = \emptyset$, the evaluation at the left and the right sides can be carried out independently.

If $V_L \cap V_R \neq \emptyset$, then for each relation $R \in V_L \cap V_R$ with multiple copies in the left side, its first appearance will serve as the representative of its other copies in the left side. Similarly, its first appearance in the right side will serve as the representative of its other copies in the right side. Thus, there are two representatives for each $R \in V_L \cap V_R$, which are located in both sides respectively. While the number of accesses to the remote sources is determined by the number of representatives in the definition of V, thus, the number of accesses to the remote sources is $K + |V_L \cap V_R| \leq 2K$. ■

Acknowledgment. The work was partially supported by both a small grant (F00025) from Australian Research Council and the Research Grants of Council of the Hong Kong Special Administrative Region (Project No. CUHK4198/00E).

References

1. D. Agrawal et al. Efficient view maintenance at data warehouses. *Proc. of the ACM-SIGMOD Conf.*, 1997, 417–427.
2. J.A. Blakeley et al. Efficiently updating materialized views. *Proc. of the ACM-SIGMOD Conf.*, 1986, 61–71.
3. L. Colby et al. Algorithms for deferred view maintenance. *Proc. of the 1996 ACM-SIGMOD Conf.*, 1996, 469–480.
4. *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing.* 18(2), June, 1995.
5. T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. *Proc. of the ACM-SIGMOD Conf.*, 1995, 328–339.
6. A. Gupta, H. Jagadish, and I. Mumick. Data integration using self-maintainable views. *Proc. 4th Int'l Conf. on Extending Database Technology*, 1996, 140–146.
7. A. Gupta, I. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. *Proc. of the ACM-SIGMOD Conf.*, 1993, 157–166.
8. R. Hull and G. Zhou. A framework for supporting data integration using the materialized and virtual approaches. *Proc. of the ACM-SIGMOD Conf.*, 1996, 481–492.
9. W. Liang et al. Making multiple views self-maintainable in a data warehouse. *Data and Knowledge Engineering*, Vol.30, 1999, 121–134.
10. W. Liang et al. Maintaining materialized views for data warehouses with the multiple remote source environments. *Proc of WAIM'00.*, LNCS, Vol.1846, 2000, 299–310.
11. D. Quass et al. Making views self-maintainable for data warehousing. *Proc. of Int'l Conf. on Parallel and Distributed Information Systems*, 1996, 158–169.
12. A. Segev and J. Park. Updating distributed materialized views. *IEEE Trans. on Knowledge and Data Engineering*, 1(2), 1989, 173–184.
13. J. Wiener et al. A system prototype for warehouse view maintenance. *Proc. of Workshop on Materialized Views*, 1996, 26–33.
14. Y. Zhuge et al. View maintenance in a warehousing environment. *Proc. of the ACM-SIGMOD Conf.*, 1995, 316–327.
15. Y. Zhuge et al. Multiple view consistency for data warehousing. *IEEE ICDE'97*, Birmingham, UK, 1997, 289–300.