
ELSEVIER

Data & Knowledge Engineering 30 (1999) 121–134

DATA & KNOWLEDGE ENGINEERING

Making multiple views self-maintainable in a data warehouse

Weifa Liang [a,1], Hui Li [b,2], Hui Wang [c,3], Maria E. Orlowska [b,c,*]

[a] *Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia*
[b] *CRC for Distributed Systems Technology, University of Queensland, St. Lucia QLD 4072, Australia*
[c] *Department of Computer Science and Electrical Engineering, University of Queensland, St. Lucia QLD 4072, Australia*

Abstract

A data warehouse collects and maintains a large amount of data from several distributed and heterogeneous data sources. Often the data is stored in the form of materialized views in order to provide fast access to the integrated data, regardless of the availability of the data sources. In this paper we focus on the following problem: for a given set of materialized select-project-join (SPJ) views, how can we find and minimize the auxiliary data stored in a data warehouse in order to make all materialized views in the data warehouse self-maintainable? For this problem we first devise an algorithm for finding such an auxiliary view set by exploiting information sharing among the auxiliary views and materialized views themselves to reduce the total size of auxiliary views. We then consider how to make the data warehouse still self-maintainable by minor modifications when there is a view addition to or deletion from it by giving an algorithm for this incremental maintenance purpose.

Keywords: Multiple materialized view maintenance; Self-maintainability; Data warehousing; Data integration; Algorithm design

1. Introduction

The problem of materialized view maintenance has received increasing attention in the past few years [8,13,9,14] due to its application to data warehousing. Traditionally, a view is a derived relation defined in terms of base relations. A view is said to be *materialized* when its content is stored in a database, rather than computed from the base databases in response to user queries. Basically, the materialized view maintenance problem is to keep the content of the materialized views consistent with the content of the base relations when there are updates to the base relations.

Data warehouses usually store materialized views in order to provide fast access to the data that is integrated from several distributed data sources [4]. The data sources may be heterogeneous and/or remote from the data warehouse. Consequently, the problem of maintaining materialized views in a data warehouse is different from the traditional view maintenance problem

* Corresponding author. E-mail: maria@csee.uq.edu.au
[1] E-mail: wliang@cs.anu.edu.au
[2] E-mail: huili@csee.uq.edu.au
[3] E-mail: hwang@csee.uq.edu.au

0169-023X/99/$ – see front matter
PII: S0169-023X(99)00009-9

where the views and the base data are stored in the same database. In particular, when a remote data source has an update, it may be necessary to access the base data from the other data sources in order to maintain the views derived from these base relations. However, access to the data sources may not be available at that time, or it may be too expensive and/or time-consuming. Therefore, a trivial alternative would be to replicate all the source data to the data warehouse. However, such a solution will incur very large additional storage and maintenance cost. Sometimes, it may be impossible to do so, because the whole space of the data warehouse is not unlimited [13,9].

In this paper we show that, for a set of *select-project-join* (SPJ for short) views, it is not necessary to replicate the entire source databases to the data warehouse in order to maintain the views, especially if some additional data that support the views can be designed and placed in the data warehouse. We give an algorithm for determining what additional data, called *auxiliary views* can be used in the warehouse in order to maintain such a set of SPJ views. The algorithm takes both the definition of the views and view-sharing information into account to reduce the total space (sizes) of all auxiliary views. Furthermore, we also consider the self-maintainability issue associated with the evolution of the data warehouse, i.e., through addition to or deletion a materialized view from the data warehouse, how can we ensure that all the views in the data warehouse are still 'self-maintainable'? Here the self-maintainability of a view is defined as: when a view V together with a set of auxiliary views $\mathscr{A}'$ can be maintained at the data warehouse without accessing the base data from data sources, the view V, is called *self-maintainable*.

1.1. Related work

The problem of a single view self-maintainability has been considered in [2,6,11,9,10]. For each modification type (insertions, deletions and updates), they identify subclasses of SPJ views that can be maintained using only the view and the modification content. In particular, [2] gives necessary and sufficient conditions on the view definition for the view to be self-maintainable for updates specified using a particular SQL modification statement. [10] considers maintaining a materialized view, without accessing the remote sources by finding a maximal test, that guarantees that the view is self-maintainable under a given update to the resources. [6] use the information about key attributes to determine the self-maintainability of a view with respect to all modifications of a certain type. [9] considers view self-maintenance by pushing down selection conditions and projections to the base relations and storing the results in the data warehouse. Later [11] extends the result of [9] by allowing a semi-join operator to be applied to the auxiliary views. They also exploit the key referential integrity constraints to further reduce the number of tuples in the auxiliary views. Based on the work by [11], recently [1] presents an algorithm to generate a minimal auxiliary view set $\mathscr{A}$ such that a data warehouse consisting of auxiliary view set $\mathscr{A}$ and a SPJ view V with aggregations, is self-maintainable. Note that all these previously known self-maintenance techniques are dealing with a single view only, while in this paper we will consider a set of views $\mathscr{V}$ which is much harder than the single case. The difference regarding the self-maintainability issue between the single view and a set of views is that: in the former case, all the auxiliary views built is for that single view only, but in the latter case, not only do we need to explore the information sharing among the auxiliary views, but also do we need to find the information sharing among the views.

1.2. Our contributions

Our contributions can be summarized as follows. We first devise an algorithm for finding an auxiliary view set $\mathscr{A}$ by exploiting information shared among the auxiliary views and views themselves to reduce the total size of auxiliary views. The algorithm runs in polynomial time. We then consider how to make the views in the data warehouse still self-maintainable by minor modifications to the set of auxiliary views, when there is a view addition to or deletion from the data warehouse by presenting an incremental algorithm for this incremetial maintenance purpose.

Remark. We should mention that, even we do use some techniques for the single view in the design of our algorithm for the multiple views, but the techniques used for single view self-maintainability cannot be extended for multiple views' self-maintainability purpose in an incremental way. The reason is that in the latter case we need to take the information sharing among the auxiliary views and the views into account. One may wonder the following naive approach may work. That is, for each view V_i in the materialized view set $\mathscr{V}$, apply the best efficient algorithm for a single view to find a minor auxiliary view set $\mathscr{A}_i$ for V_i. Then uniting all these auxiliary view sets to form an auxiliary view set $\mathscr{A} = \cup_{V_i \in \mathscr{V}} \mathscr{A}_i$ for all views in $\mathscr{V}$. Clearly, the views in $\mathscr{V}$ along the views in $\mathscr{A}$ now are self-maintainable. However, $\mathscr{A}$ is not the most economical one in terms of the space occupied by these auxiliary views because this approach does not explore possible information sharing between $\mathscr{A}_i$ and $\mathscr{A}_j$.

1.3. Paper outline

The paper is organized as follows. Section 2 introduces notations and assumptions. Section 3 first presents algorithms for finding the auxiliary view set that is sufficient to maintain a set of SPJ views, then shows how the SPJ views are maintained using the auxiliary views and the auxiliary views are self-maintainable. Section 4 deals with the self-maintenance of the views in the data warehouse by adding/deleting a view from the view set. Conclusions are given in Section 5.

2. Preliminaries

2.1. Self-maintainability

Let $\mathscr{V}$ be a set of SPJ views and each view consists of a single projection followed by a single selection followed by a single cross-product over a set of base relations. It is well known that any combination of selections, projections and joins can be expressed in this form. Now let us consider a view $V \in \mathscr{V}$ which is defined on a set of base relations $\mathscr{R} = \{R_1, R_2, \ldots, R_n\}$. If there is a change $\delta\mathscr{R}$ (here $\delta\mathscr{R}$ represents either a series of insertions $\Delta\mathscr{R}$ or a series deletions $\nabla\mathscr{R}$, or an update set to $\mathscr{R}$) to the base relations in $\mathscr{R}$, V may need to be updated accordingly, in order to keep V consistent with the base relations in $\mathscr{R}$. In doing so, we want to compute δV, the changes to V, using as little extra information as possible. If δV can be computed using only the contents of the existing V and $\delta\mathscr{R}$, then V is *self-maintainable*. Otherwise, V is not self-maintainable.

In order to make every V in $\mathscr{V}$ is self-maintainable, we are interested in finding a set of auxiliary views $\mathscr{A}$ which are defined on the relations in $\mathscr{R}$ such that every V along with the auxiliary views in $\mathscr{A}$ is self-maintainable. Clearly, $\mathscr{R}$ itself is such an $\mathscr{A}$. However, our objective is to find more "economical" auxiliary views that are much smaller than the base relations, i.e., we will find a

minimal $\mathscr{A}$. By "minimal", we mean that (i) if any auxiliary view $AV \in \mathscr{A}$ is removed from $\mathscr{A}$, then there exists at least a view $V \in \mathscr{V}$ along with the views in $\mathscr{A}-\{AV\}$ is not self-maintainable; and (ii) it is impossible to further reduce the number of tuples (therefore, the total space occupied by the tuples) in any auxiliary view by adding additional selection conditions.

In doing so, we will increase the "sharing" information across the original and auxiliary views and take the restrictions in the definition of the views in $\mathscr{V}$ into account. Note that this problem was treated as an open problem in [11]. Here we only give a partial solution for it because we does not take the key and referential constraints used in [11] into consideration.

2.2. The gluing operation

Before we proceed, we fabricate a new operation on two views derived from the a single relation. We call this operation as *gluing operation* which will generate a new view. The detail of this operation is explained as follows.

Consider a relation R. Let $\overline{X}$ and $\overline{Y}$ be subsets of the attributes of R. The intersection of $\overline{X} \cap \overline{Y}$ may or may not be $\emptyset$. Let V_i, $i = 1, 2$, be a view on R where $V_1 = \pi_{\overline{X}}\sigma_{P_1}R$ and $V_2 = \pi_{\overline{Y}}\sigma_{P_2}R$. The *gluing operation* is to find a view V_{12} such that (i) V_{12} is also derived from R and V_i can be derived from V_{12}; and (ii) there is no other view $V'_{12} \subset V_{12}$ that V_i can be derived from V'_{12}, i.e., V_{12} is a view with the minimal number of tuples, $i = 1, 2$. Clearly, $V_{12} = \pi_{\overline{X}\cup\overline{Y}}\sigma_{P_{12}}R$ where $P_{12} = P_1 \vee P_2$. Then, $V_1 = \pi_{\overline{X}}\sigma_{P_1}V_{12}$ and $V_2 = \pi_{\overline{Y}}\sigma_{P_2}V_{12}$.

We use an example to illustrate the gluing operation. Let R be a relation on the attribute set $\{A, B, C, D\}$, and let $\overline{X} = \{A, B\}$ and $\overline{Y} = \{B, C\}$. Assume that $R = \{(1,2,3,1), (1,3,3,2), (2,1,3,3), (1,3,2,4), (1,2,4,2), (1,3,1,1), (1,2,3,4)\}$. Define $V_1 = \pi_{\{A,B\}}\sigma_{D=1}R$ and $V_2 = \pi_{\{B,C\}}\sigma_{(A=1\vee D=1)}R$. It is easy to see that $V_1 = \{(1,2), (1,3)\}$ and $V_2 = \{(2,3), (3,3), (3,2), (2,4), (3,1)\}$. Let V_{12} be the result by gluing V_1 and V_2. Then, $V_{12} = \pi_{\{A,B,C\}}\sigma_{(A=1\vee D=1)}R$. Clearly, $V_{12} = \{(1,2,3), (1,3,3), (1,3,2), (1,2,4), (1,3,1)\}$.

3. Algorithms for generating auxiliary views

In this section we first adopt a simple algorithm for finding an auxiliary view set for a single view from [9] which will be used as a subroutine in our algorithm. We then introduce our algorithm for finding an auxiliary view set for a given view set. In doing so, Firstly, we explore our algorithmic idea by considering the case where the view set contains only two views. Secondly, the algorithm for finding the auxiliary view set for more than 2 views is presented in Section 3.3. Finally, we show how to maintain each view in the view set using the auxiliary views only, we also show how to maintain the auxiliary views themselves.

3.1. Finding auxiliary views for a single view

Given a view V defined on a set of base relations $\mathscr{R}$, the objective is to find an auxiliary view set $\mathscr{A}$ such that $\{V\} \cup \mathscr{A}$ is self-maintainable upon changes to base relations in $\mathscr{R}$ without accessing the base relations. Certainly $\mathscr{R}$ is such an $\mathscr{A}$. But the cost in terms of space may be very high. It is possible to reduce the total storage space occupied by the auxiliary views in $\mathscr{A}$, using the following *local reduction rule* on the relations in $\mathscr{R}$.

Local Reduction Rule: This reduction refers to pushing projections and local selection conditions (i.e., select conditions involving attributes from a single relation as opposed to those

involving attributes from different relations that are called join conditions) to each base relation $R_i \in \mathscr{R}$ [1,9,11]. As a result, the number of attributes and tuples in the auxiliary view that replaces R_i is significantly reduced, compared with the number of attributes and tuples in R_i, because those tuples in R_i that do not pass local conditions cannot possibly contribute to the tuples in the view. Hence, they are not needed for the view maintenance and are therefore not needed to be stored in the auxiliary view.

Without loss of generality, let $V = \pi_{\overline{X}}\sigma_P(R_{i_1} \bowtie R_{i_2} \bowtie \ldots \bowtie R_{i_k})$ is a SPJ view where $R_{i_l} \in \mathscr{R}$, $1 \leqslant l \leqslant k$. Then, V can be rewritten as $V = \pi'_{\overline{X}}\sigma'_{P'}(\pi_{\overline{X}_1}\sigma_{P_1}R_{i_1} \bowtie \pi_{\overline{X}_2}\sigma_{P_2}R_{i_2} \bowtie \ldots \bowtie \pi_{\overline{X}_k}\sigma_{P_k}R_{i_k})$, where $\pi'_{\overline{X}}$ is the global projection and $\sigma'_{P'}$ is the join conditions, $\pi_{\overline{X}_l}$ is local projection including both the preserved projection attributes and the attributes in the join conditions, and σ_{P_l} is the local selection condition, $1 \leqslant l \leqslant k$. Let

$$A_{R_{i_l}} = \pi_{\overline{X}_l}\sigma_{P_l}R_{i_l}. \tag{1}$$

Define $A_{R_{i_l}}$ as an *auxiliary view* of V which is a selection and a projection on relation R_i. Thus, there is an auxiliary view set $\mathscr{A}$ for V which is defined as follows: $\mathscr{A} = \{A_{R_{i_l}} | 1 \leqslant l \leqslant k\}$. It is not hard to show that $\mathscr{A} \cup \{V\}$ is a minimal self-maintainable set [9].

3.2. Finding an auxiliary view set for two views

Having the preparation in the previous section, we start by considering a case where $\mathscr{V} = \{V_1, V_2\}$ and $\mathscr{R} = \{R_1, R_2, \ldots, R_n\}$. Recall that our objective is to generate an auxiliary view set $\mathscr{A}$ such that (i) $\mathscr{A} \cup \{V_1, V_2\}$ is self-maintainable; and (ii) the total space occupied by all auxiliary views is minimal.

The basic idea of our algorithm for it is to decide whether two corresponding auxiliary views derived from a single base relation are needed to be merged into an auxiliary view or just keep both the auxiliary views in $\mathscr{A}$. This decision is made by a space cost function.

Procedure `Find_Auxiliary_View_Set(`V, A`)`

/* Input: $\mathscr{V}$ is the set of two materialized views. */

/* Output: $\mathscr{A}$ is the auxiliary view set such that $\mathscr{A} \cup \{V_1, V_2\}$ is self-maintainable. */

1. Find two separate auxiliary view sets $\mathscr{X}_1 = \{A^1_{R_1}, A^1_{R_2}, \ldots, A^1_{R_n}\}$ and $\mathscr{X}_2 = \{A^2_{R_1}, A^2_{R_2}, \ldots, A^2_{R_n}\}$ for V_1 and V_2 by local reduction rule.
2. $\mathscr{A} := \emptyset$; /* the initial auxiliary view set */
3. **for** $i = 1$ **to** n **do**
   Let A_{R_i} be the resulting view by applying the gluing operation to $A^1_{R_i}$ and $A^2_{R_i}$
   and let N_i be the number of tuples of A_{R_i}.
   Let n_{ij} and s_{ij} be the number of tuples and bytes per tuple in $A^j_{R_i}$,
   let S_i be the total number of bytes by the attributes in $\overline{X} \cap \overline{Y}$,
   where $\overline{X}$ and $\overline{Y}$ are the projection attributes of $A^1_{R_i}$ and $A^2_{R_i}$.
4. **if** $N_i(s_{i1} + s_{i2} - S_i) \leqslant (n_{i1}s_{i1} + n_{i2}s_{i2})$
   then $\mathscr{A} := \mathscr{A} \cup \{A_{R_i}\}$
   else $\mathscr{A} := \mathscr{A} \cup \{A^1_{R_i}, A^2_{R_i}\}$
   endif

endfor.

We now make some comments about the condition at Step 4 in `Find_Auxiliary_View_Set2`. Actually, this condition can be further generalized as $N_i(s_{i1}+s_{i2}-S_i) \leqslant \alpha(n_{i1}s_{i1}+n_{i2}s_{i2})$ where α ($\geqslant 1$) is a constant which is determined by the CPU processing speed and I/O transfer speed. Because our aim is to minimize the total space occupied by all auxiliary views, if we use separate auxiliary views $A^1_{R_i}$ and $A^2_{R_i}$ for V_1 and V_2, then there is a possibility that the sum of the space occupied by both of them is smaller than the space used by A_{R_i}. But the maintenance cost (time) to $A^1_{R_i}$ and $A^2_{R_i}$ may be much higher than that to A_{R_i} when there is an update to R_i. Simply, we probably need to update both $A^1_{R_i}$ and $A^2_{R_i}$, which are usually the disk residents. This means that it takes more I/O time to find their locations in disk and load them to the memory for the updates. On the other hand, if the space occupied by A_{R_i} is not too large, compared with that for both $A^1_{R_i}$ and $A^2_{R_i}$, then the maintenance cost of A_{R_i} is reduced dramatically, since we only need to load it to the memory once. However, the expense of this is that the CPU time for updating A_{R_i} and V_i is increased because the number of tuples in A_{R_i} is larger than the number of tuples in either $A^1_{R_i}$ or $A^2_{R_i}$. Therefore, there is a trade-off between the maintenance cost and space cost when we choose to store either$A^j_{R_i}$, $j=1,2$, or A_{R_i}, depending on the ratio between the I/O time and CPU processing time. Here we just choose that the simple one (i.e., $\alpha=1$) that when the space occupied by A_{R_i} is smaller than that by the sum of $A^1_{R_i}$ and $A^2_{R_i}$, A_{R_i} is stored in the auxiliary view set. Otherwise $A^j_{R_i}$ is stored in the auxiliary set, $j=1,2$.

If $N_i(s_{i1}+s_{i2}-S_i) > (n_{i1}s_{i1}+n_{i2}s_{i2})$ at Step 4, then the net increasing space would be $\Delta S_i = N_i(s_{i1}+s_{i2}-S_i) - \sum_{j=1}^{2} n_{ij}s_{ij}$ if A_{R_i} is kept in the data warehouse.

We should mention that the cost used for generating an auxiliary view set is only computed once. In some cases, when the base data involved is very large, some sophisticated sampling techniques may need to be used to examine a small part of the relations in order to give an approximate estimation of the cost.

3.3. Finding an auxiliary view set for multiple views

Through the previous discussion, we are ready to introduce our algorithm for finding an auxiliary view set $\mathscr{A}$ for a set of views $\mathscr{V}=\{V_1,V_2,\ldots,V_m\}$ with $m \geqslant 2$. The proposed algorithm is a greedy algorithm which proceeds as follows. Initially, the solution is empty and the cost for this solution is 0. Let $\mathscr{A}$ be the solution of the problem so far such that the views in $\mathscr{A} \cup \mathscr{E}\mathscr{V}$ are self-maintainable and minimal, where $\mathscr{E}\mathscr{V}$ is the set of explored views by the algorithm, which is a subset of $\mathscr{V}$. Then, each time we pick a view $V \in \mathscr{V} - \mathscr{E}\mathscr{V}$ such that the net increase of the cost of the new auxiliary view set is minimized, due to the addition V to $\mathscr{E}\mathscr{V}$. In doing so, we exploit the data sharing among the auxiliary views for V and the auxiliary views in the solution. This process continues until $\mathscr{V} - \mathscr{E}\mathscr{V} = \emptyset$, i.e., the final solution is obtained. The algorithm is described as follows.

Procedure `Find_Auxiliary_View_Setm`($\mathscr{V}$,$\mathscr{A}$)
/* Input: $\mathscr{V}$ is the set of m materialized views, $m \geqslant 2$. */
/* Output: $\mathscr{A}$ is the auxiliary view set such that $\mathscr{A} \cup \mathscr{V}$ is self-maintainable. */
1.
  for $i := 1$ **to** m **do**
    find an auxiliary view set $\mathscr{X}_i = \{A^i_{R_1}, A^i_{R_2}, \ldots, A^i_{R_n}\}$ for every V_i
      by local reduction rule.

endfor;
/* Without loss of generality, we here assume that a view is derived from all R_i in $\mathscr{R}$. */
/* In practice a view may be derived from a subset of the relations in $\mathscr{R}$.*/

2.

Initialization. Let $S(\mathscr{X})$ be the total space by the views in $\mathscr{X}$.
$S(\mathscr{X}_{i_0}) = \min\{S(\mathscr{X}_i) |\ i = 1, 2, \ldots, m\}$.
/* choose the auxiliary set of a view with the minimum space as the initial solution */
$\mathscr{A} := \mathscr{X}_{i_0}$; /* $\mathscr{A}$ is the auxiliary view set. */
$\mathscr{UV} := \mathscr{V} - \{V_{i_0}\}$; /* the unexploited view set */
$\mathscr{EV} := \{V_{i_0}\}$; /* the exploited view set. */
/* Let $\mathscr{A} = \bigcup_{i=1}^{n} \mathscr{A}_{R_i}$, where $\mathscr{A}_{R_i} = \{Z_1, Z_2, \ldots, Z_s\}$ is */
/* the auxiliary view set derived from R_i. */

3.

while $\mathscr{UV} \neq \emptyset$ **do** /* choose the next view and add it to the solution */
Choose a $V_j \in \mathscr{UV}$ such that the net increase of space for auxiliary view set is minimal.
Such a V_j can be found by a subroutine `Cost_Comput(`$\mathscr{UV}, \mathscr{A}, V_j$`)`.
/* The procedure `Cost_Comput` is explained later */
$\mathscr{UV} := \mathscr{UV} - \{V_j\}$;
$\mathscr{EV} := \mathscr{EV} \cup \{V_j\}$;
Update $\mathscr{A}$ using the auxiliary view set of V_j
endwhile.

Now we turn to select a V_j at Step 3 in `Find_Auxiliary_View_Setm`. We already knew that $\mathscr{A}$ is the auxiliary view set for all views $\mathscr{EV}$ so far, and is self-maintainable, where $\mathscr{A} = \bigcup_{i=1}^{n} \mathscr{A}_{R_i}$ and $\mathscr{A}_{R_i} = \{Z_1, Z_2, \ldots, Z_s\}$ is the auxiliary view set derived from R_i.

When $\mathscr{UV} \neq \emptyset$, we choose a view $V_j \in \mathscr{UV}$ and add it to $\mathscr{EV}$ such that (i) the net increasing of the space occupied by the auxiliary view set is minimized, due to the addition of V_j, (i.e., we choose such a view V_j which incurs the minimum cost increase with respect to the current solution); and (ii) the new auxiliary view set is also self-maintainable. The basic strategy of our algorithm is that we treat $\mathscr{A}$ as an auxiliary view set, and then decide whether to merge every view in the auxiliary view set of V_j with the corresponding view in $\mathscr{A}$ into a single view according to the cost function. The selection of V_j is implemented by the following procedure.

Procedure `Cost_Comput`($\mathscr{UV}, \mathscr{A}, V$);
/* Input: $\mathscr{UV}$ and $\mathscr{A}$; */
/* Output: a view V from $\mathscr{UV}$ is chosen such that the net space increase of the auxiliary view set is minimal. */
$C(V) := \infty$; /* the cost by adding a view V to the solution */
while $\mathscr{UV} \neq \emptyset$ **do**
Let $V_j \in \mathscr{UV}$ be the considered view currently.
$C(V_j) := 0$; /* the cost incurred by adding V_j */
for $i = 1$ **to** n **do**
Call `Cost_Estimate`($A^j_{R_i}$,$\mathscr{A}_{R_i}$, ΔC_{ji});
/* `Cost_Estimate` is used to estimate the space cost if V_j is added to the solution, */

/* and it will be presented below. */
$C(V_j) := C(V_j) + \Delta C_{ji}$;
endfor;
/* choose the view which incurs the minimum increase in the cost */
if $C(V_j) < C(V)$ **then**
$V := V_j$;
$C(V) := C(V_j)$
endif
endwhile;
return V.

The subroutine `Cost_Estimate` is invoked by `Cost_Comput`, which is explained in more detail as following. Let V_j be the view being considered and $\mathscr{X}_j$ be the auxiliary view set for V_j only by the local reduction rule. Recall that $A^j_{R_i} \in \mathscr{X}_j$ is the initial auxiliary view derived from R_i for V_j solely and $\mathscr{A}_{R_i}$ be a set of auxiliary views derived from R_i for all the views in $\mathscr{EV}$. Assume that $\mathscr{A}_{R_i} = \{Z_1, Z_2, \ldots, Z_s\}$. The aim of `Cost_Estimate` is to decide whether gluing $A^j_{R_i}$ with a view $Z_l \in \mathscr{A}_{R_i}$ or leave it alone by the cost function.

Let $Z_l = \pi_{\overline{W}}\sigma_P R_i$ and $A^j_{R_i} = \pi_{\overline{X}_j}\sigma_{P_j} R_i$. Let $Z'_l = \pi_{\overline{W}\cup\overline{X}_j}\sigma_{P'} R_i$ be the resulting view by gluing Z_l with $A^j_{R_i}$ where $P' = P \vee P_j$.

Denote by $c(A^j_{R_i}, Z_l)$ the cost by gluing $A^j_{R_i}$ with Z_l, $1 \leqslant l \leqslant s$, which is defined as follows.

$$c(A^j_{R_i}, Z_l) = |Z'_l| \cdot S_{Z'_l} - (|A^j_{R_i}| \cdot S_{A^j_{R_i}} + |Z_l| \cdot S_{Z_l}), \tag{2}$$

where $|R|$ is the number of tuples in R, S_R is the number of bytes occupied per tuple in R. Then

Procedure `Cost_Estimate`($A^j_{R_i}$,$\mathscr{A}_{R_i}$, ΔC_{ji});
/* Input: $A^j_{R_i}$ and $\mathscr{A}_{R_i}$; */
/* Output: The net increase of space by incorporating $A^j_{R_i}$ to the auxiliary view set ΔC_{ji} */
1.
$\mathscr{U} := \mathscr{A}_{R_i}$;
$\Delta C_{ji} = |A^j_{R_i}| \cdot S_{A^j_{R_i}}$; /* initialization */
2.
while $\mathscr{U}_i \neq \emptyset$ **do**
Let $Z_l \in \mathscr{U}_i$ be the auxiliary view considered currently.
if $\Delta C_{ji} > c(A^j_{R_i}, Z_l)$ **then**
$\Delta C_{ji} := c(A^j_{R_i}, Z_l)$;
endif
endwhile;
return ΔC_{ji}.

Step 2 in `Cost_Estimate` is explained as follows. Compare the minimum cost $\min\{c(A^j_{R_i}, Z_l) | 1 \leqslant l \leqslant s\}$ with the cost of $A^j_{R_i}$ which is $|A^j_{R_i}| \cdot S_{A^j_{R_i}}$, if the former is smaller, then $\Delta C_{ji} = \min\{c(A^j_{R_i}, Z_l) | 1 \leqslant l \leqslant s\}$, which means $\mathscr{A}_{R_i}$ needs updating by gluing $A^j_{R_i}$ to a Z_l with the minimum cost $c(A^j_{R_i}, Z_l$ and replace it by the gluing result; otherwise, $\Delta C_{ji} = |A^j_{R_i}| \cdot S_{A^j_{R_i}}$, which means $A^j_{R_i}$ will be included to $\mathscr{A}_{R_i}$, i.e., $\mathscr{A}_{R_i} = \mathscr{A}_{R_i} \cup \{A^j_{R_i}\}$.

Theorem 1. *Given a set of views $\mathscr{V}$, let $\mathscr{A}$ be the auxiliary view set generated by algorithm* `Find_Auxiliary_View_Setm`. *Then, $\mathscr{A}$ can be generated in time $O(m^2nT_{CE})$, where m and n are the number of materialized views and the base relations respectively, and T_{CE} is the time for running* `Cost_Estimate` *which is a linear function of the number of auxiliary views derived from a base relation in the auxiliary view set.*

Proof. We now analyze the computational complexity of algorithm `Find_Auxiliary_View_Setm`. Step 1 takes $O(mn)$ time because it needs $O(n)$ time to generate an auxiliary view set for each materialized view. Step 2 takes $O(m)$ time. There are m iterations at Step 3 and at each iteration, it invokes a subroutine `Cost_Comput` which requires $O(mnT_{CE})$ time, where T_{CE} is the time for running `Cost_Estimate`, which is a linear function of the number of auxiliary views derived from a base relation in the auxiliary view set so far. Therefore, Step 3 requires $O(m^2nT_{CE})$ time. □

3.4. Maintaining the materialized views using auxiliary views

A view maintenance expression calculates the effects on the view of a certain type of change: insertions, deletions and updates to a base relation. View maintenance expressions are usually written in terms of the changes and the base relations [3,5].

In the following we show how to transform a view maintenance expression written in terms of the changes and the base relations to an equivalent view maintenance expression written in terms of the changes, the view and the auxiliary views generated by our algorithm.

A SPJ view is expressed as follows.

$$V_i = \pi_{\overline{X}_i}\sigma_{P_i}(R_{i_1} \bowtie R_{i_2} \bowtie \ldots \bowtie R_{i_s}). \tag{3}$$

Now we use the auxiliary view set to re-write the definition of V_i, then

$$V_i = \pi_{\overline{X}_i}\sigma_{P_i}(Z_{j_1} \bowtie Z_{j_2} \bowtie \ldots \bowtie Z_{j_s}), \tag{4}$$

where $Z_{j_l} \in \mathscr{A}_{R_{i_l}} \subset \mathscr{A}$, i.e., if $A^i_{R_{i_l}} \in \mathscr{A}_{R_{i_l}}$, then $Z_{j_l} = A^i_{R_{i_l}}$, otherwise, Z_{j_l} is the auxiliary view to which $A^i_{R_{i_l}}$ is glued, $1 \leqslant l \leqslant s$.

Now we consider the insertion updates to a base relation R_{i_l}. Let ΔR_{i_l} be the net effect to R_{i_l} which leads to update V_i [12]. Then, the maintenance expression of V_i that uses R_{i_l} in its definition can be written as

$$\Delta V_i = \pi_{\overline{X}_i}\sigma_{P_i}(Z_{j_1} \bowtie Z_{j_2} \bowtie \ldots \bowtie \Delta R_{i_l} \bowtie \ldots \bowtie Z_{j_s}). \tag{5}$$

Therefore, the content of V_i is $V_i \cup \Delta V_i$ after the insertion update. The deletion case can be discussed in a similar way, however we omit it here. The modification value update case can be treated by a deletion followed by an insertion. When an insertion, a deletion, or a modification update to more than one data resource happens at the same time, this case can be dealt by using the technique developed in [7]. For the same reason, we will not discuss it further in this paper.

3.5. Maintaining auxiliary views

In the previous section we have shown that the materialized view in $\mathscr{V}$ is self-maintainable using the views in the auxiliary view set found by our algorithm. We now show that the auxiliary view set itself is also self-maintainable. Thus, all the views in $\mathscr{V} \cup \mathscr{A}$ are self-maintainable.

Consider an arbitrary auxiliary view $Z_l \in \mathscr{A}_R \subset \mathscr{A}$ and $R \in \mathscr{R}$, which can be written as follows.

$$Z_l = \pi_{\text{Schema}(Z_l)}\sigma_P R. \tag{6}$$

It is clear that Z_l is a SPJ view. When there is a sequence of changes to R, the changes are propagated to the data warehouse. Denote by ΔR the net effect of insertions to R, then the maintenance expression for Z_l is

$$\Delta Z_l = \pi_{\text{Schema}(Z_l)}\sigma_P \Delta R. \tag{7}$$

Thus, the content of Z_l after insertions is $Z_l \cup \Delta Z_l$. The deletion as well the modification updates can be dealt with using a similar approach, but we omit them for further discussion.

In the following we show that the information needed to maintain the original views is also sufficient to maintain each of their auxiliary views.

Theorem 2. *The auxiliary view set $\mathscr{A}$ for $\mathscr{V}$ generated by algorithm* `Find_Auxiliary_View_Setm` *is a minimal, self-maintainable set.*

Proof. In order to verify that $\mathscr{A}$ is a minimal, self-maintainable set, following [11], we must show $\mathscr{A}$ satisfies the following three properties.

1. $\mathscr{A}$ is sufficient to maintain every $V \in \mathscr{V}$;
2. $\mathscr{A}$ is self-maintainable;
3. $\mathscr{A}$ is a minimal view set which make the above two conditions hold.

We now show that $\mathscr{A}$ satisfies the above three properties one by one. We first show that the maintenance expression propagating insertions/deletions to the base relations onto every V that is rewritten by substituting for each base relation R with a corresponding auxiliary view $Z_l = \pi_{\overline{W}_l}\sigma_{P_l} R \in \mathscr{A}_R \subseteq \mathscr{A}$ is equivalent to the maintenance expression using the base relations. Let $V_i = \pi_{\overline{X}_i}\sigma_{P_i}(R_{i_1} \bowtie R_{i_2} \bowtie \ldots \bowtie R_{i_s})$ be written in terms of the base relations. Then V_i can be rewritten in terms of the views in the auxiliary view set, i.e., $V_i = \pi_{\overline{X}_i}\sigma_{P_i}(Z_{j_1} \bowtie Z_{j_2} \bowtie \ldots \bowtie Z_{j_s})$ where $Z_{j_l} \in \mathscr{A}_{R_{i_l}} \subseteq \mathscr{A}$ for all l, $1 \leqslant l \leqslant s$.

Now, we consider an insertion update ΔR_{i_l}. By the definition, the changes to V_i is $\Delta V_i = \pi_{\overline{X}_i}\sigma_{P_i}(R_{i_1} \bowtie R_{i_2} \bowtie \ldots \bowtie \Delta R_{i_l} \bowtie \ldots \bowtie R_{i_s})$. Let $U = \pi_{\overline{X}_i}\sigma_{P_i}(Z_{j_1} \bowtie Z_{j_2} \bowtie \ldots \bowtie \Delta Z_{i_l} \ldots \bowtie Z_{j_s})$ where $\Delta Z_{i_l} = \pi_{\overline{W}_{i_l}}\sigma_{P_{i_l}}\Delta R_{i_l}$. Our objective is to show that $\Delta V_i = U$. We prove this by contradiction approach. Assume that there is a tuple t such that $t \in \Delta V_i$ but $t \notin U$. Since $t \in \Delta V_i$, then there is at least a tuple $t_{i_{l'}} \in R_{i_{l'}}$ for every l' such that $t_{i_{l'}}[\overline{W}_{i_{l'}}] \in Z_{j_{l'}}$, $t_{i_l}[\overline{W}_{i_l}] \in \pi_{\overline{W}_{i_l}}\sigma_{P_{i_l}}\Delta R_{i_l}$, i.e., $t_{i_l} \in \Delta Z_{i_l}$, and t is the result of joining $t_{i_1}, t_{i_2}, \ldots, t_{i_s}$ together, $1 \leqslant l' \leqslant s$ with $l \neq l'$. From this discussion, it is understood that t can also be derived by joining $t_{i_1}[\overline{W}_{i_1}], t_{i_2}[\overline{W}_{i_2}], \ldots, t_{i_s}[\overline{W}_{i_s}]$ together, i.e., $t \in U$ because $t_{i_l}[\overline{W}_{i_l}] \in \Delta Z_{j_l}$ and $t_{i_{l'}}[\overline{W}_{i_{l'}}] \in Z_{j_{l'}}$ for every $l' \neq l$. This contradicts with our initial assumption. The deletion and modification cases can be dealt similarly, and they are omitted.

We then show that $\mathscr{A}$ is self-maintainable. Consider every $Z_l = \pi_{\overline{X}}\sigma_P R_i \in \mathscr{A}_{R_i}$. Let ΔR_i be the set of insertions to R_i, then $\Delta Z_l = \pi_{\overline{X}}\sigma_P \Delta R_i$. Thus, the content of Z_l after the insertion update is $Z_l := Z_l \cup \Delta Z_l$. The deletion and modification case can be dealt similarly, omitted. By the definition of self-maintainability, clearly $\mathscr{A}$ is self-maintainable.

We finally show that $\mathscr{A}$ is a minimal set. Let Z_l be an auxiliary view derived from R_i. Clearly $Z_l \in \mathscr{A}_{R_i}$. Recall that we say that a set $\mathscr{A}$ is *minimal*, which means neither a Z_l can be removed

from $\mathscr{A}$ nor an additional local selection condition can be applied to Z_l to further reduce the number of tuples in Z_l. Firstly, we show that each $Z_l \in \mathscr{A}$ cannot be removed and is useful. If $Z_l = A^j_{R_i}$, then there is at least a view V_j which uses Z_l, this means the removal of Z_l will lead to the maintenance of V_j becoming impossible. Otherwise, Z_l is the result derived from views $V_{i_1}, V_{i_2}, \ldots, V_{i_s}$ by gluing the corresponding auxiliary views together. From the definition of V_{i_p}, it uses Z_l as a component in its definition, $1 \leqslant p \leqslant s$. The removal of Z_l will make the maintenance of these views impossible. So, all the auxiliary views in $\mathscr{A}$ are useful and cannot be removed. Note that all views in $\mathscr{V}$ cannot be removed either because they are the materialized views. Secondly, we show that for each $Z_l \in \mathscr{A}_{R_i}$, the number of tuples in Z_l cannot be further reduced by adding the local selection conditions. If $Z_l = A^j_{R_i}$, then Z_l is at least used by V_j and the entire local conditions for V_j have been already added to Z_l, so, any further additional local selection condition added to it may remove some tuples for V_j. Otherwise, Z_l is a result derived from the views $V_{i_1}, V_{i_2}, \ldots, V_{i_s}$ by gluing the corresponding auxiliary views together. In this case we cannot add any additional local selection condition either, because each such additions will make at least one view V_{i_p} lose its tuples. □

4. Evolution of a self-maintainable data warehouse

In the previous section we considered how to generate a self-maintainable data warehouse for a given set of views $\mathscr{V}$. Due to the dynamic changes of user query environment, some materialized views in the warehouse may become useless, while some new views are needed to be materialized. Under such an environment, how to make the data warehouse remain self-maintainable is very important.

In this section we deal with this issue by considering to add a new view V to, or delete an existing view V from, the data warehouse. Suppose we only allow to make "minor modifications" to the data warehouse. Otherwise, it is quite straightforward to run the algorithm in Section 3 from scratch to generate a new auxiliary view set, using either $\mathscr{V} \cup \{V\}$ (inserting V) or $\mathscr{V} - \{V\}$ (deleting V) as an input. However, the cost of such a process may be very expensive. The reasons are twofold: one is that we need to generate a new auxiliary view set to replace the current one. In doing so, the data warehouse needs to communicate with the remote sources heavily which is very expensive or impossible sometimes, despite just only one view removal from or addition to the data warehouse. The other is that usually it takes much longer time to generate a new auxiliary view set, compared to incremetial updates the view in the existing auxiliary view set.

In the following we first discuss the insertion case, we then deal with the deletion case. The modification case can be dealt by a deletion followed by an insertion and omitted.

4.1. Adding new materialized views to the data warehouse

We now consider the insertion case. Assume that $\mathscr{V} \cup \mathscr{A}$ is self-maintainable initially, now the question is to how to ensure $\mathscr{V} \cup \{V\} \cup \mathscr{A}'$ is still self-maintainable when a new view V is added to the data warehouse, where $\mathscr{A}'$ is the resulting auxiliary view set which is obtained through minor modifications to $\mathscr{A}$. We discuss this case by the following two subcases.

If either $V \subseteq V_i$ and $V_i \in \mathscr{V}$ or V can be derived from the existing views in $\mathscr{V}$, then $\mathscr{A}' := \mathscr{A}$, i.e., $\mathscr{V} \cup \{V\} \cup \mathscr{A}'$ is self-maintainable and we do nothing about it. We re-write the definition of V

by using the auxiliary views involved. The checking takes polynomial time since the containment relationships between SPJ queries can be checked in polynomial time.

Otherwise, we proceed by first running the algorithm for the single view V to obtain an auxiliary view set $\mathscr{X}_{\text{new}} := \{A_{R_1}^{\text{new}}, A_{R_2}^{\text{new}}, \ldots, A_{R_n}^{\text{new}}\}$ of V. We then run the following procedure to update the existing materialized auxiliary views in the data warehouse. Assume that $\mathscr{A}_{R_i} = \{Z_1, Z_2, \ldots, Z_{s_i}\}$ is the auxiliary view set derived from R_i for all the views in $\mathscr{V}$. The new auxiliary view set $\mathscr{A}'$ for $\mathscr{V} \cup \{V\}$ is generated as follows.

```
Update_Auxi_Set (𝒜, 𝒜', V)
/* Input: 𝒜 is the original auxiliary view set for 𝒱 and */
/*        V is a new view to be added to 𝒱. */
/* Output: 𝒜' is the resulting auxiliary view set after adding V to 𝒱 */
   𝒜' := 𝒜; /* initial assignment */
   for   i = 1 to n do
      𝒜' := 𝒜' − 𝒜_{R_i}; /* delete the auxiliary views derived from R_i */
      Call Cost_Estimate(A_{R_i}^{new}, 𝒜_{R_i}, ΔC_{ji});
      /* update the auxiliary views derived from R_i by incorporating the auxiliary views of V.
      /* The definition of Cost_Estimate can be seen in Section 3. */
      Update A_{ℛ_i} according to the result of Cost_Estimate
      𝒜' := 𝒜' ∪ 𝒜_{R_i};
   endfor.
```

Theorem 3. *Assume that a set of SPJ views $\mathscr{V}$ and their auxiliary view set $\mathscr{A}$ is self-maintainable. Let $\mathscr{A}'$ be the auxiliary view set generated by adding a new view V to the data warehouse using the above algorithm, then the set $\mathscr{A}' \cup \mathscr{V} \cup \{V\}$ is a minimal, self-maintainable set.*

Proof. The proof is similar to the proof of Theorem 1, and it is omitted here. □

Following this, we rewrite the maintenance expression of V in terms of the auxiliary views which has been described in Section 3.4. The maintenance expressions of those auxiliary views which glued with the views in $\mathscr{X}_{\text{new}}$ are also needed to be rewritten accordingly.

4.2. Deleting materialized views from the data warehouse

The deletion case can be dealt, using a similar technique for the insertion case. Assume that $\mathscr{V} \cup \mathscr{A}$ is self-maintainable initially, the problem is then how to keep $\mathscr{V} - \{V\} \cup \mathscr{A}'$ still self-maintainable when V is deleted from the warehouse, where $\mathscr{A}'$ is obtained through minor modifications to $\mathscr{A}$.

The naive approach is to do nothing about it. But in the end this leads to many useless auxiliary views in the data warehouse. A better approach in handling this case is presented below.

If there is a view $V_i \in \mathscr{V}$ such that V has been used in the definition of V_i, then we take no action, and $\mathscr{A}' := \mathscr{A}$. Otherwise, V now becomes useless and V should be removed from the warehouse. It is easy to remove V. However, to remove some auxiliary views that are no longer useful seems not to be trivial. We proceed as follows.

First, we run the algorithm for the single view V to generate an auxiliary view set $\mathscr{X}_{\text{old}} := \{A^{\text{old}}_{R_1}, A^{\text{old}}_{R_2}, \ldots, A^{\text{old}}_{R_n}\}$ for V. Then, we check the usefulness of the views in $\mathscr{A}$ one by one. Assume that $Z_l \in \mathscr{A}$ is the current considered view. If Z_l is the result of gluing $A^{\text{old}}_{R_i}$ with some other views, check all the views $V' \in \mathscr{V}$ which are added to the solution after V during generating the auxiliary view set $\mathscr{A}$ to see whether V' uses Z_l, if there is such a V' using Z_l, we do nothing about Z_l. Otherwise, if only V solely uses it, we remove Z_l from $\mathscr{A}$, i.e., $\mathscr{A}' = \mathscr{A} - \{Z_l\}$; otherwise the definition of Z_l is re-written by the following method.

Initially let $Z_l = \pi_{\overline{W}}\sigma_P R_i$, where $P = P_{i_1} \vee P_{i_2} \vee \ldots \vee P_{i_p} \vee P_i$ and $\sigma_{P_{i_j}}$ is the local selection condition to R_i by the definition of V_{i_j}, $1 \leqslant i_j \leqslant i$ and $1 \leqslant j \leqslant p$. Let $A^{\text{old}}_{R_i} = \pi_{\overline{X}_j}\sigma_{P_j}R_i$. Then, Z_l is redefined as $Z_l = \pi_{\overline{W}'}\sigma_{P'}R_i$, which is the definition of Z_l before adding V_i to the solution during the generation of $\mathscr{A}$, where W' is a subset of attributes of R_i which consists of the preserved projection and those appeared in the join conditions in V_{i_j} and $P' = P_{i_1} \vee P_{i_2} \vee \ldots \vee P_{i_p}$, $1 \leqslant j \leqslant p$.

5. Conclusions and future directions

In this paper we have considered the self-maintainability issue of views in data warehousing. That is, given a set of SPJ views, we have shown how to derive an auxiliary view set such that (i) the SPJ views and the auxiliary views together are self-maintainable; and (ii) the total space occupied by the auxiliary views is minimized, by devising a polynomial greedy algorithm for it. As a result, the auxiliary view set generated by the proposed algorithm occupies reasonable space. We have also considered the evolution issue of a slef-maintainable data warehouse by presenting incremetial algorithms for dealing with views' insertion and deletion. It is interesting to investigate how the key and referential integrity constraints in [11] can be applied to further reduce the size of the auxiliary views.

Acknowledgements

We appreciate the three anonymous referees for their invaluable suggestions and comments which help us improve the paper's quality and presentation.

References

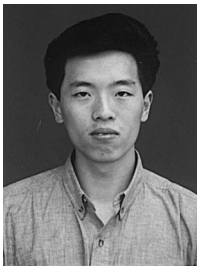
[1] M.O. Akinde, O.G. Jensen, M.H. Bohlen, Minimizing detail data in data warehouses, in: Proceedings of the Sixth International Conference on Extending Database Technology, LNCS, vol. 1377, Springer, Spain, March 1998, pp. 293–307.
[2] J. Blakeley, N. Coburn, P. Larson, Updating derived relations: detecting irrelevant and autonomously computable updates, ACM Transactions on Database Systems 14 (3) (1989) 369–400.
[3] L. Colby, T. Griffin, L. Libkin, I. Mumick, H. Trickey, Algorithms for deferred view maintenance, in: Proceedings of the ACM-SIGMOD Conference, 1996, pp. 469–480.
[4] IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2) (1995).
[5] T. Griffin, L. Libkin, Incremental maintenance of views with duplicates, in: Proceedings of the ACM-SIGMOD Conference, 1995, pp. 328–339.
[6] A. Gupta, H. Jagadish, I. Mumick, Data integration using self-maintainable views, in: Proceedings of the Fourth International Conference on Extending Database Technology, 1996, pp. 140–144.
[7] A. Gupta, I. Mumick, V.S. Subrahmanian, Maintaining views incrementally, in: Proceedings of the ACM-SIGMOD Conference, 1993, pp. 157–166.

[8] A. Gupta, I. Mumick, Maintenance of materialized views: problems, techniques, and applications, IEEE Data Engineering Bulletin (1995) 3–19.
[9] R. Hull, G. Zhou, A framework for supporting data integration using the materialized and virtual approaches, in: Proceedings of the ACM-SIGMOD Conference, 1996, pp. 481–492.
[10] N. Huyn, Efficient view self-maintenance, in: Proceedings of the 23rd VLDB Conference, Athens, Greece, 1997, pp. 26–35.
[11] D. Quass, A. Gupta, I.S. Mumick, J. Widom, Making views self-maintainable for data warehousing, in: Proceedings of the International Conference on Parallel and Distributed Information Systems, Miami Beach, FL, 1996, pp. 158–169.
[12] A. Segev, J. Park, Updating distributed materialized views, IEEE Transactions on Knowledge and Data Engineering 1 (2) (1989) 173–184.
[13] Y. Zhuge, H. Garcia-Molina, J. Hammer, J. Widom, View maintenance in a warehousing environment, in: Proceedings of the ACM-SIGMOD Conference, 1995, pp. 316–327.
[14] Y. Zhuge, H. Garcia-Molina, J.L. Wiener, Consistency algorithms for multi-source warehouse view maintenance, Journal of Distributed and Parallel Database 6 (1998) 7–40.

Weifa Liang received his Ph.D. degree in computer science from the Australian National University in 1998. He received his M.E. degree in computer science from University of Science and Technology of China in 1989 and his B.S. degree in computer science from Wuhan University, China in 1984. He is currently holding a teaching position in the Department of Computer Science at the Australian National University. His research interests include parallel processing, parallel and distributed algorithms, computer networking, data warehousing, query optimisation and graph theory.

Hui Wang is currently a master (by research) student in computer science at Department of Computer Science and Electrical Engineering in the University of Queensland. She received her B.S. degree in mathematics from Anhui University, China in 1984. Before coming to Australia, as a software engineer, she has been worked in an institution in China for a decade to conduct research and development of application software in the simulation of VLSI circuits. Her current research interests include design and analysis of data warehousing, the consistency control of views in data warehousing and relational database application.

Maria Orlowska is currently Professor in Information Systems at the University of Queensland in Australia. Since 1992 she has also acted as Distributed Databases Unit Leader in the Cooperative Research Centre for Distributed Systems Technology (DSTC). She graduated with a PhD (Computer Science) in June 1980 from the Institute of Applied Mathematics, Technical University of Warsaw. She is a trustee of the VLDB Endowment. Her research expertise lies in the areas of: theory of Relational Databases, Distributed Databases, various aspects of Information Systems Design Methodologies (including Distributed Systems), enhancement of semantic data modelling techniques by rigorous factors, transaction processing in distributed systems, concurrency control, Distributed and Federated Database Systems, and workflows technology.



Hui Li is a research scientist of the Cooperative Research Center for Distributed Systems Technology of Australia and Ph.D. candidate of University of Queensland. He received his M.Sc. degree (1993) from Southwest Jiaotong University, and B.Sc. degree from Nanjing University, both in Computer Science. His research interests include query processing, distributed and heterogeneous databases, object-oriented and object-relational data model, workflow systems, Internet Information Retrieval and expert systems.