Practical Approach to Selecting Data Warehouse Views Using Data Dependencies

Gillian Dobbie and Tok Wang Ling

Department of Computer Science, National University of Singapore, Singapore
{dobbie,lingtw}@comp.nus.edu.sg

Abstract. Materialized views in data warehouses are typically complicated, making the maintenance of such views difficult. However, they are also very important for improving the speed of access to the information in the data warehouse. So, the selection of materialized views is crucial to the operation of the data warehouse both with respect to maintenance and speed of access. Most research to date has treated the selection of materialized views as an optimization problem with respect to the cost of view maintenance and/or with respect to the cost of queries. In this paper, we consider practical aspects of data warehousing. We identify problems with the star and snowflake schema and suggest solutions. We also identify practical problems that may arise during view selection and suggest heuristics based on data dependencies and access patterns that can be used to measure if one set of views is better than another set of views, or used to improve a set of views.

1 Introduction

A data warehouse stores huge volumes of data that has been gathered from one or more sources for the purpose of efficiently processing decision support or on-line analytic processing (OLAP) queries. Like in traditional database systems, frequently asked queries or subparts of frequently asked queries may be precomputed and stored as materialized views, providing faster access. Obviously there are many possible views that could be materialized, and the selection of which views to materialize is a trade-off between the cost of the view maintenance and the speed of access. Views in data warehouses are usually more complicated than in traditional database systems, typically based on many tables and including aggregation or summarization of the underlying data in the data warehouse.

Most research to date has treated the selection of materialized views as an optimization problem with respect to the cost of view maintenance and/or with respect to the cost of queries [1,4,8,9,10,11]. Each paper proposes an algorithm designed within the framework of general query and maintenance cost models without considering the physical properties of the actual data. In this paper, we identify practical problems that may arise during view selection and suggest heuristics based on data dependencies and access patterns that can be used to measure if one view is better than another or used to improve a set of views. Our work is related to physical database design and materialized view design in the relational databases.

The paper is organized as follows. Section 2 provides background information relevant to the rest of the paper, highlights problems with the widely accepted star and snowflake schema, and presents the enhanced star and snowflake schema. There is a sample data warehouse in Section 3 that is used in the following sections. Section 4 compares different views suggesting why one is better than another. Heuristics for good design are outlined in Section 5, and demonstrated in Section 6. We conclude in Section 7.

2 Background

In this section, we introduce the star and snowflake schema, describe inadequacies of the star and snowflake schema, present the enhanced star and snowflake schema, and introduce strong and weak functional dependencies.

The schema of a data warehouse that is built on top of a relational database is typically organized as a *star* or *snowflake* schema [6]. A *star schema* consists of a fact table, and a table for each dimension in the fact table. A star schema can be represented using an entity relationship diagram as shown in Figure 1. The fact table represents a relationship set between two or more dimension entities and has some measurement attributes (*salesQty* in Figure 1). Each dimension table represents an entity with an identifier and other single valued attributes. A *snowflake schema* is like a star schema except it represents the dimensional hierarchies directly, normalizing the dimension tables.

Example 1. Consider a data warehouse that stores information about employees, products and the quantity of each product sold by each employee. There would be a dimension table for employee and another for product storing the employee and product details respectively. The fact table would contain the identifier of employee, the identifier of product and the quantity of a product sold by an employee. A dimension table can contain a dimension hierarchy, e.g. if each product belongs to a category and each category has many products, then we say there is a product dimension hierarchy. □

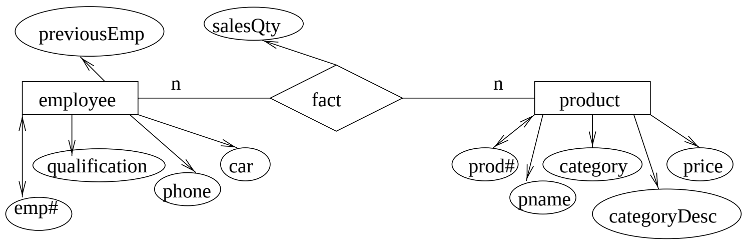

Fig. 1. ER diagram of typical star schema

While the star and snowflake schemas described above are convenient, they are overly simplistic. In practice, not all attributes in a dimension table are single valued, not all keys in dimension tables consist of just one attribute, and the relationships between the levels in a dimension hierarchy may be m-to-n (rather than 1-to-n). Consider the schema in Figure 2. Each employee may have more than one qualification, more than one previous employer, more than one phone and more than one car. That is the attributes in the employee dimension table are more likely to be multi-valued attributes and then the key is a composite key. Although we don't represent it in Figure 2, a product could also belong to more than one category. For example, a *health food drink* may belong to category *health food* and category *beverage*. The following functional dependencies hold on the schema in Figure 2:

$\{emp\#, prod\#\} \rightarrow salesQty$
$emp\# \twoheadrightarrow qualification$
$emp\# \twoheadrightarrow previousEmp$
$emp\# \twoheadrightarrow phone$
$emp\# \twoheadrightarrow car$
$prod\# \rightarrow \{pname, category, price\}$
$category \rightarrow categoryDesc.$

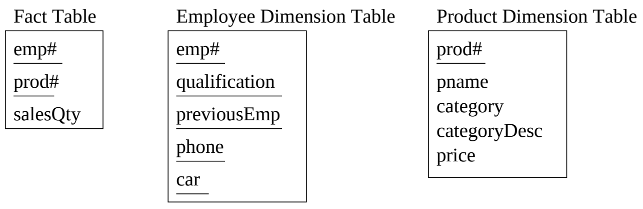

Fig. 2. Sample enhanced star schema

While the snowflake schema eliminates the redundancy in the dimension hierarchy, the problems we have described are not alleviated. An alternative organization would be to store the composite key of the employee dimension table in the fact table: *Fact Table*($emp\#$, $qualification$, $previousEmp$, $phone$, car, $prod\#$, $salesQty$). This organization is worse than the schema in Figure 2. Not only are we duplicating all the employee information, we are also confusing the relationship between $emp\#$, $prod\#$, and $salesQty$.

In summary, the star and snowflake schemas that are proposed in many papers and books are overly simplistic and not practical for real world applications. Similarly, any view design heuristics that are based on the same assumptions will be overly simplistic.

Strong and *weak* functional dependencies [7] extend classical functional dependencies and when used in relational database design can provide better schemas than those produced using classical functional dependencies.

Strong functional dependency: Let $X \rightarrow Y$ be a functional dependency such that for each $z \in Y$, $X \rightarrow z$ is a full functional dependency. Then $X \rightarrow Y$ is a *strong functional dependency* if the values of all the attributes in Y will not be updated, or if the update need not be performed at real-time or on-line and such updates seldom occur.

Example 2. A person's name seldom changes, so we can say there is a strong functional dependency between *employee_number* and *employee_name*. We write this as $employee\_number \overset{S}{\rightarrow} employee\_name$. □

Weak functional dependency: Let X and Y be subsets of a table R, such that there is a non-trivial multi-valued dependency $X \twoheadrightarrow Y$ in R. If most of the X values are associated with a unique Y-value in R, except for the occasional X-value that may be associated with more than one Y-value, we say Y is *weakly functionally dependent* on X, and write $X \overset{W}{\rightarrow} Y$.

Example 3. Assume that typically an employee lists only one phone number, and very occasionally an employee lists more than one phone number, then we can say that $employee\_number \overset{W}{\rightarrow} employee\_phone$. □

3 Motivating Example

In this section we introduce a populated sample data warehouse that we use in later sections of this paper. While the volume of data in this sample data warehouse (see Figure 3) is unrealistic, the relationships between the fields are realistic and are used to motivate our work. The *Fact table* stores the quantity of each product sold by each employee. The *Employee* table contains, for each employee, their employee number, qualifications, previous employers, phone numbers and car registration. It is uncommon for an employee to have more than one telephone number or more than one car, but quite common for an employee to have more than one qualification and previous employer. The *Product* table contains the product number, name, the category the product belongs to, a text description of the category, and the price of the product. Each product belongs to one category. We expect the fact table to be modified often, the price to be modified sometimes and other attributes to change less frequently. The following dependencies hold on the data:

$$\begin{array}{lll}
emp\# \overset{W}{\rightarrow} phone & emp\# \twoheadrightarrow qualification & prod\# \rightarrow price \\
emp\# \overset{W}{\rightarrow} car & \{emp\#, prod\#\} \rightarrow salesQty & prod\# \overset{S}{\rightarrow} pname \\
emp\# \twoheadrightarrow previousEmp & category \overset{S}{\rightarrow} categoryDesc & prod\# \overset{S}{\rightarrow} category.
\end{array}$$

The key of the *Fact table* is {*emp#, prod#*}. The key of the *Employee* table is {*emp#, qualification, previousEmp, phone, car*}. The key of the *Product* table is {*prod#*}. Access patterns are derived from typical queries. The monthly reports include:

Q1. total quantity of items sold by each employee,
Q2. total quantity of items sold by product, listing both product number and product name,
Q3. total quantity of items sold by category, listing both category and category description,

Fact table

emp#	prod#	salesQty
1	10	10
2	10	15
3	10	35
1	20	8
2	20	14
3	20	30
1	30	6
2	30	10
3	30	25

Employee

emp#	qualification	previousEmp	phone	car
1	HSC	Bob's fruit shop	8725911	DX6195
1	BSc	Bob's fruit shop	8725911	DX6195
1	MSc	Bob's fruit shop	8725911	DX6195
2	HSC	Quality Groceries	8741834	LX5255
2	BSc	Quality Groceries	8741834	LX5255
2	HSC	Shop and Save	8741834	LX5255
2	BSc	Shop and Save	8741834	LX5255
3	HSC	Shop and Save	7794580	MM5735
3	HSC	Shop and Save	6741856	MM5735

Product

prod#	pname	category	categoryDesc	price
10	weetbix	cereal	A cereal is typically eaten for breakfast. Cereals can be stored on shelves and have a shelf life of 3 months. ...	2
20	vegemite	spread	A spread is eaten with bread. Speads can be stored on shelves and have a shelf life of 6 months. ...	4
30	apple	fruit	Fruit (excluding bananas) must be stored in a partially refrigerated part of the shop. Fruit has a shelf life of 1 week to 1 month. ...	1
40	orange	fruit	Fruit (excluding bananas) must be stored in a partially refrigerated part of the shop. Fruit has a shelf life of 1 week to 1 month. ...	2

Fig. 3. Example data warehouse

Q4. total sales of items sold by employee,
Q5. total sales of items sold by product, listing both product number and product name,
Q6. total sales of items sold by category, listing both category and category description,
Q7. average number of items for all products sold by each employee,
Q8. average number of items for all employees by each product, and
Q9. details of each product, and employee number and phone number of the top performing employee for each product.

Less frequently the following information is required:

Q10. A manager is interested in the correlation between the average number of products sold by employees and the average number of qualifications of the employees.
Q11. To get a feel for what to look for when hiring people, a manager may want to know who the previous employers are of his three top sales staff.

4 Problems

In this section, we discuss possible problems in view selection, briefly compare different views and give reasons why one set of views is better than another. A fuller comparison can be found in [2].

If there is an attribute that changes frequently, how should the materialized views be selected? In traditional databases, if there is a relation $R(A,\ B,\ C)$ where $A\ \rightarrow\ B$ and $B\ \rightarrow\ C$ then it is suggested that the relation is decomposed into $R_1(A,\ B)$ and $R_2(B,\ C)$. The reason being that when C is updated, it will need to be updated in many places if the schema with R is used, but only once if the schema with R_1 and R_2 is used. Consider the scenario where the values of C are very unlikely to change, (i.e. $B \overset{S}{\rightarrow} C$) then there is no need to decompose R. The same reasoning can be followed when selecting data warehouse views.

If a multi-valued attribute is usually single-valued, how should the views be selected? In traditional databases, if there is a relation $R(A,B,C)$ and $A \twoheadrightarrow B$, then it is suggested that the relation be decomposed into $R_1(A,\ B)$ and $R_2(A,\ C)$. However, if the multi-valued attribute usually has only one value, like an employee usually having only one phone number, then this information can be stored as though it is single valued with an extra (overflow) table for the occasional extra phone number. The same reasoning can be followed when selecting data warehouse views. Methods for maintaining the overflow tables are described in [7].

If a multi-valued attribute is usually multi-valued, how should the views be selected? When designing traditional database systems, one of the overriding aims is to reduce the number of joins when answering queries, however there are occasions when selecting views for data warehouses where this is not the best approach. A view should be decomposed into two views if two attributes in the view are independent and frequently have multiple values.

Example 4. Consider view $V(emp\#,\ previousEmp,\ prod\#,\ pname,\ salesQty)$ where an employee has on average 10 previous employers and sells 100 products. To find the previous employer of an employee, it is necessary to access 1000 tuples on average. If instead, the view is decomposed into $V_1(emp\#,\ previousEmp)$ and $V_2(emp\#,\ prod\#,\ pname,\ sales)$, it'd be necessary to access only 10 tuples on average to find the previous employers of an employee. On the other hand, view V would be preferable to views V_1 and V_2 if very few employees have more than one previous employer, or *previousEmp* and *pname* are frequently accessed together. □

Is there a possibility of anomalous data being introduced when views are selected? If there is an attribute in a table that you will aggregate on, then this table can only be joined with other tables with the same key, otherwise anomalous data will be introduced and the following aggregation will be incorrect. This usually arises when you are joining a fact table (or a summarization of a fact table) and a dimension table, and the fact table contains a measurement attribute that is likely to be aggregated in the future.

Example 5. Consider creating a view $V(emp\#, prod\#, qualification, salesQty)$ with employee number 1, and product number 10. Using the sample data warehouse in Section 3, the view would contain the following information:

emp#	prod#	qualification	salesQty
1	10	HSC	10
1	10	BSc	10
1	10	MSc	10

The total sales calculated from view V is 30 whereas the total sales for *emp#* 1, *prod#* 10 calculated from the Fact table is 10. The problem is that the *salesQty* has been replicated for each *qualification* of the employee. □

If the size of attributes varies greatly or views are large, how should the views be selected? There are two situations in which vertical partitioning is advantageous. If there are some attributes that are small and some that are very large, then when the smaller attributes are accessed, the whole tuple including the very large attributes must be read into temporary storage. A better design is to vertically partition the large attributes from the other attributes. Another related situation is where a view has many attributes and those attributes are never accessed together. The access speed is negatively affected by the extra attributes in the view.

If a view is very large and queries are often based on particular values for an attribute, the table can be horizontally partitioned on that attribute to improve performance. For example, consider a large view with an attribute *city#*, where access frequently involves individual cities, then performance will be improved if the view is horizontally partitioned based on the *city#*.

If a key attribute is large, how should the views be selected? If there is a set of views where one table has a large key and that key is used as a foreign key in another view then a surrogate key can be introduced and the surrogate key can replace the foreign key e.g. where the attribute *categoryDesc* is a key in one view and a foreign key in another view.

When should two views be joined? Consider joining (a subset of the attributes of) a fact table and (a subset of the attributes of) a dimension table to form a view. If on average the key of the dimension table occurs infrequently in the fact table, then (the subset of the attributes of) the dimension table can be joined to (the subset of the attributes of) the fact table to form a view, otherwise it isn't worthwhile forming a view.

Example 6. Consider the view $V(emp\#, prod\#, pname, category, price)$ formed by joining a subset of the attributes of the fact table with a subset of the attributes of the product table. View V can be formed if on average each *prod#* value occurs infrequently in the fact table. If on the other hand, each *prod#* value occurs frequently on average then forming view V wastes a lot of space and the product information is harder to maintain, so no such view should be formed.

How should views with aggregates be selected? The way aggregates are stored for a dimension hierarchy is dependent on the frequency of updates, and the access pattern as demonstrated in Example 7. Recall that when you are decomposing a table while normalizing a traditional database, it is important that no information is lost. When you are selecting views for data warehouses, this constraint no longer applies because the underlying tables remain.

Example 7. Assume we require both the sum of *salesQty* grouped by *prod#*, $\Sigma salesQty_{prod\#}$, and the sum of *salesQty* by category, $\Sigma salesQty_{category}$.[1] The following are possible sets of views:

– $V(prod\#,\ pname,\ category,\ \Sigma salesQty_{prod\#})$,
– $V_1(prod\#,\ pname,\ \Sigma salesQty_{prod\#})$, $V_2(category,\ \Sigma salesQty_{category})$,
– $V_3(prod\#,\ pname,\ category,\ \Sigma salesQty_{prod\#})$,
  $V_4(category,\ \Sigma salesQty_{category})$.

The view V is preferable over the other sets of views under the following conditions:

– View V is preferable if the number of sales is updated often. If either of the alternative sets of views are used, when the sales quantity is updated, both aggregations of the sales quantity must also be recalculated.
– View V is preferable if there are very few queries asking for sales by category, because only then is it worthwhile computing the aggregation at the time of the query.
– Otherwise, one of the other sets of views is preferable.

The set of views with V_1 and V_2 is preferable if there are few queries involving the relationship between *pname* and *category* otherwise the set of views with V_3 and V_4 is preferable. □

5 Design Heuristics

The following heuristics take into account the problems described in Section 4. The heuristics can be used for two related but different purposes, to judge if one set of views is better than another, and to improve a set of views. The aims are to make maintenance easier, and to reduce the access cost, without introducing anomalous data. The process of selecting materialized views involves the following three steps:

S_1. Create a temporary view for each query by selecting the attributes that are required to answer the query.
S_2. Join temporary views from Step S_1 grouping attributes that belong to the same entities together. This minimizes the space that will be needed to store the materialized views and makes updating the views less error prone.
S_3. Finally select the best set of views by decomposing the temporary views found in Step S_2, using the following heuristics.

[1] $\Sigma salesQty_{category}$ can also be calculated by grouping $\Sigma salesQty_{prod\#}$ by *category*.

5.1 Reduce Access Cost Using Data Dependencies

The heuristics in this section are based on the weak and strong functional dependencies that were introduced in Section 2.

OK Rule

a1. The view $V(A,\ B,\ C,\ D)$ is OK if
   - $A \rightarrow \{B,\ C,\ D\}$, or
   - $A \rightarrow \{B,\ D\}$ and $B \overset{S}{\rightarrow} C$. [2]

a2. The view $V(A,\ B,\ C)$ is OK, if there is a non-trivial multi-valued dependency $A \twoheadrightarrow B$, on average there are not many values for B and C, and the attributes B and C are frequently accessed together.

Not OK Rule

b1. If $A \rightarrow \{B,\ D\}$ and $B \rightarrow C$ but $B \overset{S}{\not\rightarrow} C$,[3] then the view $V(A,\ B,\ C,\ D)$ should be decomposed into $V_1(A,\ B,\ D)$ and $V_2(B,\ C)$.

b2. If there is a non-trivial multi-valued dependency $A \twoheadrightarrow B$ and $A \overset{W}{\rightarrow} B$ and $A \overset{W}{\rightarrow} C$, the view $V(A,\ B,\ C)$ is replaced by $V_1(A,\ B,\ C)$, $V_{bOverflow}(A,\ B)$ and $V_{cOverflow}(A,\ C)$.

b3. Let there be a fact table, R(A,E,F) where $\{A, E\} \rightarrow F$, and $\{A, E\} \overset{S}{\not\rightarrow} F$. Let B be the aggregation of F grouped on A (i.e. ΣF_A) and C be an aggregate that is computed using B (e.g. sum of B, average of B), then view $V(A,\ B,\ C)$ should be replaced by $V_1(A,\ B,\ D)$, where D is another attribute can be used to compute C from B. [4]

b4. If there is a non trivial multi-valued dependency $A \twoheadrightarrow B$ and $A \overset{W}{\not\rightarrow} B$, on average there are many values for B and C, and the attributes B and C are not frequently accessed together the view $V(A,\ B,\ C)$ should be decomposed into $V_1(A,\ B)$ and $V_2(A,\ C)$.

5.2 Reduce Access Cost Using Physical Database Principles

Traditionally in database systems the query cost is reduced by choices made during physical database design. The heuristics in this section are based on physical database design principles, like those in [3], and adapted for views in data warehouses.

OK Rule

c1. A view $V(A, B, C)$ is OK if $A \rightarrow \{B, C\}$ and each of the attributes is a "reasonable" size. It is up to the designer to judge what a reasonable size is.

c2. A view is OK, if it has a large key but the key is not used as a foreign key in another view.

---

[2] The values of the attributes in C are not updated frequently.

[3] The values of the attributes in C are updated frequently.

[4] If B is ΣF_A, and C is **av** F_A, then D may be **count** F_A.

c3. A view that is created by joining (a subset of the attributes of) a fact table with (a subset of the attributes of) a dimension table is OK if
  – the key of the dimension table is a subset of the key of the fact table, and
  – each value of the key of the dimension table occurs infrequently, on average, in the fact table.

c4. Consider a view $V(A,\ B,\ C)$ (with key A) where A and B are two levels in a dimension hierarchy and C is a measurement for A. For example, A could be a product number, B a category and C the quantity of the product sold. The view V is OK if
  – the attribute C is updated often, or
  – there are few queries that aggregate C grouping by B.

Not OK Rule

d1. If there is a view $V(A,\ B,\ C)$ where $A \to \{B, C\}$ and attribute C is very large then decompose the view into $V_1(A,\ B)$ and $V_2(A,\ C)$.

d2. If there is a view $V(A,\ B,\ C)$ where $A \to \{B, C\}$ and queries are frequently asked for particular values of attribute B then V can be horizontally partitioned on B.

d3. If the key in a view V_1 is large and the key is being used as a foreign key in another view V_2 then introduce a surrogate key into V_1 and replace the foreign key in V_2 by the surrogate key.

d4. A view $V(A,\ B,\ C,\ E,\ F,\ G)$ that is created by joining (a subset of the attributes of) a fact table $R(A,\ B,\ C)$ (with key $\{A,\ B\}$) with (a subset of the attributes of) a dimension table $R_1(B,\ E,\ F,\ G)$ (with key B) should not be formed if each value of the key of the dimension table occurs frequently, on average, in the fact table.

d5. Consider a view $V(A,\ B,\ C)$ (with key A) where A and B are two levels in a dimension hierarchy and C is a measurement of A. The view V should be decomposed into $V_1(A,\ C)$ and $V_2(B, \Sigma C_B)$ if
  – the attribute C is not updated often,
  – there are few queries that include the relationship between A and B,
  – there are frequent queries that aggregate C grouping by B.

  If the above properties hold but there are frequent queries that include the relationship between A and B then replace V with $V_3(A, B, C)$ and $V_4(B, \Sigma C_B)$.

d6. If there is a view $V(A,\ B,\ C,\ D,\ E,\ F)$ with key A and there are frequent queries that access $A,\ B,\ C$ and frequent queries that access $D,\ E,\ F$ then vertically partition V into $V_1(A,\ B,\ C)$ and $V_2(A,\ D,\ E,\ F)$.

5.3 Do Not Introduce Anomalous Data

The heuristics in this section guard against anomalous data being introduced when views are selected. The anomalous data problem arises where there is aggregation or summarization of the underlying data. Views are built from underlying tables using select, project, join, and aggregation operations. As we demonstrated in Example 5 anomalous data may be introduced if there is a measurement that is likely to be aggregated or a measurement that is aggregated, and the keys of the underlying tables are not the same.

Let there be a table $R_1(A, B, C)$ and another table $R_2(A, D, E)$. We write $\mathcal{F}C_A$ to denote that function $\mathcal{F}$ is likely to be or has been applied to attribute C after grouping on A.

OK Rule

e1. It is OK to create a view as the result of joining a temporary table $R_T(A, \mathcal{F} C_A)$ to any table that has key A.

Not OK Rule

f1. If there is a table (or temporary table) $R_T(A, \mathcal{F}C_A)$ and a table $R_2(A, D, E)$ with a composite key e.g. $\{A, D\}$ then the tables should not be joined.

Example 8. Consider a fact table $F(A,\ B,\ C)$ with key $\{A,\ B\}$ and a dimension table $D(A,\ E,\ F)$ with key $\{A,\ E\}$. Anomalous data will be introduced if the view $V(A,\ E,\ \Sigma C_A)$ is created. The view must be replaced by $V_1(A,\ E)$ and $V_2(A,\ \Sigma C_A)$. □

6 Demonstrating the Heuristics

In this section we demonstrate how views are selected for the example data warehouse in Section 3 based on the procedure outlined in Section 5.

Step S_1, we create a view for each of the queries presented in Section 3.[5] Both $Q9$ and $Q11$ involve selecting top performing employees.[6]
$Q1(\underline{emp\#},\ \Sigma salesQty_{emp\#})$
$Q2(\underline{prod\#},\ pname,\ \Sigma salesQty_{prod\#})$
$Q3(\underline{category},\ categoryDesc,\ \Sigma salesQty_{category})$
$Q4(\underline{emp\#},\ \Sigma(price\ \times\ salesQty)_{emp\#})$
$Q5(\underline{prod\#},\ pname,\ \Sigma(price\ \times\ salesQty)_{prod\#})$
$Q6(\underline{category},\ categoryDesc,\ \Sigma(price\ \times\ salesQty)_{category})$
$Q7(\underline{emp\#},\ \mathbf{av} salesQty_{emp\#})$
$Q8(\underline{prod\#},\ pname,\ \mathbf{av} salesQty_{prod\#})$
$Q9(\underline{prod\#,\ emp\#,\ phone},\ pname,\ category,\ categoryDesc,\ price)$
$Q10(\underline{emp\#},\ \Sigma salesQty_{emp\#},\ \mathbf{count}\ qualification_{emp\#})$
$Q11(\underline{emp\#,\ previousEmp})$

[5] For simplicity we use the query number (from Section 3) as the name of the matching view.

[6] Horizontal partitioning could be appropriate but because the tuples in the partition change often, we do not perform the partitioning.

Step S_2, new temporary views are formed by grouping all attributes that belong to an entity together. For example, we group all attributes that belong to the *employee* entity in one view.[7] The resulting views are:

- $VT_{emp\#}(\underline{emp\#,\ previousEmp,\ phone},\ \Sigma salesQty_{emp\#}, \mathbf{av} salesQty_{emp\#},$ $\Sigma(price\ \times\ salesQty)_{emp\#},\ \mathbf{count}\ qualification_{emp\#})$,[8]
- $VT_{prod\#}(\underline{prod\#},\ pname,\ category,\ \Sigma salesQty_{prod\#}, \mathbf{av} salesQty_{prod\#},$ $\Sigma(price\ \times\ salesQty)_{prod\#},\ price)$, [9]
- $VT_{category}(\underline{category},\ \Sigma salesQty_{category},\ \Sigma(price\ \times\ salesQty)_{category},$ $categoryDesc)$.[10]

Step S_3, we use the heuristics together with the dependencies described in Section 3 to improve the views produced in the previous step as illustrated in Figures 4 and 5. We consider $VT_{emp\#}$, $VT_{prod\#}$ and $VT_{category}$, and then the resulting views together. Based on the heuristics, the set of views, $V_{overflow}$, V_3, V_4, V_7 and V_9, are the best for this schema, with the given access pattern.

7 Conclusions

The selection of materialized views is crucial to data warehouse performance. To date, most of the research in the area uses general cost models to find the optimal or near optimal set of views, without considering the physical properties of the data.[11] In this paper, we have shown that the widely recognized star and snowflake schema are based on overly simplistic assumptions, so any view design heuristics based on these schema are also overly simplistic. We discussed problems that may arise when selecting which views to materialize for a data warehouse. Our main contributions are the enhanced star and snowflake schema and the set of heuristics that can be used to improve a set of materialized views, or judge if one set of views is better than another set of views. The heuristics are based on physical properties of the data such as how likely it is that the value of an attribute will change, how often a multi-valued attribute will have more than one value, access patterns, and size of attributes. We have provided a preliminary investigation on which further practical work, involving the selection of materialized views in data warehouses, can be based.

This is preliminary work and there are many directions we could take from here. One direction is to perform experiments to verify our claims of improved performance and another is to formalize the heuristics presented and further

[7] The attributes in $Q9$ are split between the three entity views.

[8] Formed by joining the attributes from $Q1$, $Q4$, $Q7$, $Q10$, $Q11$ and the attributes *emp#* and *phone* from $Q9$.

[9] Formed by joining the attributes from $Q2$, $Q5$, $Q8$, and the attributes *prod#*, *pname*, *category* and *price* from $Q9$.

[10] Formed by joining the attributes from $Q3$, $Q6$, and the attributes *category* and *categoryDesc* from $Q9$.

[11] See [2] for a more thorough discussion of the related work.

$VT_{emp\#}(\underline{emp\#, phone, previousEmp}, \Sigma salesQty_{emp\#}, \Sigma(price \times salesQty)_{emp\#},$
$\quad \mathbf{av} salesQty_{emp\#}, \mathbf{count} qualification_{emp\#})$

$$\Downarrow b_2 \leftharpoonup emp\# \overset{W}{\rightarrow} phone \leftharpoondown$$
$$VT_{emp\#} is replaced by V_{overflow} and VT_{emp\#}$$

$V_{overflow}(\underline{emp\#, phone})$
$VT_{emp\#}(\underline{emp\#, previousEmp}, phone, \Sigma salesQty_{emp\#}, \Sigma(price \times salesQty)_{emp\#},$
$\quad \mathbf{av} salesQty_{emp\#}, \mathbf{count} qualification_{emp\#})$

$$\Downarrow b_3 \leftharpoonup aggregates based on \Sigma salesQty_{emp\#} \leftharpoondown$$
$$VT_{emp\#} is replaced by VT_2$$

$V_{overflow}(\underline{emp\#, phone})$
$VT_2(\underline{emp\#, previousEmp}, phone, \Sigma salesQty_{emp\#}, \mathbf{count} qualification_{emp\#})$

$$\Downarrow b_4 \leftharpoonup emp\# \twoheadrightarrow previousEmp \leftharpoondown$$
$$VT_2 is replaced by V_3 and V_4$$

$V_{overflow}(\underline{emp\#, phone})$
$V_3(\underline{emp\#, previousEmp})$
$V_4(\underline{emp\#}, phone, \Sigma salesQty_{emp\#}, \mathbf{count} qualification_{emp\#})$

$VT_{prod\#}(\underline{prod\#}, pname, category, \Sigma salesQty_{prod\#}, \Sigma(price \times salesQty)_{prod\#},$
$\quad \mathbf{av} salesQty_{prod\#}, price)$

$$\Downarrow b_3 \leftharpoonup aggregates based on \Sigma salesQty_{prod\#} \leftharpoondown$$
$$VT_{prod\#} is replaced by VT_5$$

$VT_5(\underline{prod\#}, pname, category, \Sigma salesQty_{prod\#}, price)$

$VT_{category}(\underline{category}, \Sigma salesQty_{category}, \Sigma(price \times salesQty)_{category},$
$\quad categoryDesc)$

$$\Downarrow b_3 \leftharpoonup aggregates based on \Sigma salesQty_{category} \leftharpoondown$$
$$VT_{category} is replaced by VT_6$$

$VT_6(\underline{category}, categoryDesc, \Sigma salesQty_{category})$

$$\Downarrow d_1 \leftharpoonup categoryDesc is large field \leftharpoondown$$
$$VT_6 is replaced by V_7 and VT_8$$

$V_7(\underline{category}, categoryDesc)$
$VT_8(\underline{category}, \Sigma salesQty_{category})\triangleright$

Fig. 4. Views after heuristics applied in Step S_3 to 3 entity views from Step S_2

$VT_5(\underline{prod\#}, pname, category, \Sigma salesQty_{prod\#}, price)$
$VT_8(\underline{category}, \Sigma salesQty_{category})$

$\Downarrow$ c4 (*salesQty* updated often)
VT_5 and VT_8 are replaced by V_9

$V_9(\underline{prod\#, category}, pname, \Sigma salesQty_{prod\#}, price)$.

Fig. 5. Resulting views in Step S_3 from VT_5 and VT_8

investigate the heuristics. We need to find answers to the following questions: Does one heuristic make something that was OK, not OK or vice versa? Do any of the rules conflict? Is the result different if the rules are applied in a different order? Could some of the heuristics be replaced by first ensuring that the views are in relax-replicated 3NF (see [7])? Following this line of investigation would clarify the distinction between traditional database design and data warehouse view design and could lead into investigating the kinds of indexes that are best, from a practical perspective, for materialized views in data warehouses.

References

1. Elena Baralis, Stefano Paraboschi and Ernest Teniente. Materialized Views Selection in a Multidimensional Database. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97)*, 1997.
2. Gillian Dobbie and Tok Wang Ling. Practical Approach to Selecting Data Warehouse Views Using Data Dependencies. *Technical Report from School of Computing, National University of Singapore, No. TRA7/00.*
3. Rob Gillette, Dean Muench and Jean Tabaka. *Physical Database Design for SYBASE SQL Server.* Prentice Hall PTR, 1995.
4. Himanshu Gupta and Inderpal Singh Mumick. Selection of Views to Materialize Under a Maintenance Cost Constraint. In *Database Theory - ICDT '99, 7th International Conference on Database Theory (ICDT)*, 1999, pages 453–470, Springer-Verlag LNCS 1540.
5. Ashish Gupta and Inderpal Singh Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. In *Data Engineering Bulletin*, 18(2), pages 3–18, 1995.
6. R. Kimball. *The data warehouse toolkit.* John Wiley and Sons, 1996.
7. Tok Wang Ling, Cheng Hian Goh and Mong Li Lee. Extending Classical Functional Dependencies for Physical Database Design. In *Information and Software Technology*, pages 601-608, vol 38 (1996), Elsevier Science.
8. Kenneth A. Ross, Divesh Srivastava and S. Sudarshan. Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 447-458.
9. Amit Shukla, Prasad Deshpande and Jeffrey F. Naughton. Materialized View Selection for Multidimensional Datasets. In *Proceedings of 24th International Conference on Very Large Data Bases, (VLDB'98)*, 1998, pages 488-499.

10. Dimitri Theodoratos, Spyros Ligoudistianos, and Timos Sellis. Designing the Global Data Warehouse with SPJ Views. In *Proceedings of 11th International Conference on Advanced Information Systems Engineering, (CAiSE'99)*, 1999, Springer-Verlag LNCS 1626.
11. Jian Yang, Kamalakar Karlapalem and Qing Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proceedings of 23rd International Conference on Very Large Data Bases, (VLDB'97)*, 1997, pages 136-145.