The Generalized Pre-Grouping Transformation: Aggregate-Query Optimization in the Presence of Dependencies

Aris Tsois Timos Sellis

Institute of Communication and Computer Systems and School of Electrical and Computer Engineering
National Technical University of Athens
Zographou 15773, Athens, Hellas
{atsois,timos}@dblab.ece.ntua.gr

Abstract

One of the recently proposed techniques for the efficient evaluation of OLAP aggregate queries is the usage of clustering access methods. These methods store the fact table of a data warehouse clustered according to the dimension hierarchies using special attributes called hierarchical surrogate keys. In the presence of these access methods new processing and optimization techniques have been recently proposed. One important such optimization technique, called Hierarchical Pre-Grouping, uses the hierarchical surrogate keys in order to aggregate the fact table tuples as early as possible and to avoid redundant joins.

In this paper, we study the Pre-Grouping transformation, attempting to generalize its applicability and identify its relationship to other similar transformations. Our results include a general algebraic definition of the Pre-Grouping transformation along with the formal definition of sufficient conditions for applying the transformation. Using a provided theorem we show that Pre-Grouping can be applied in the presence of functional and inclusion dependencies without the explicit usage of hierarchical surrogate keys. An additional result of our study is the definition of the Surrogate-Join transformation that can modify a join condition using a number of dependencies. To our knowledge, Surrogate-Join does not belong to any of the Semantic Query Transformation types discussed in the past.

**Proceedings of the 29th VLDB Conference,
Berlin, Germany, 2003**

1. Introduction

In the Data Warehousing (DW) and OnLine Analytical Processing (OLAP) areas, the need for fast response times to large aggregation queries has motivated research and implementation efforts for quite some time. Various methods and solutions have been proposed from both the industry and the academy. The well-known star-schema, the specialized access methods and the usage of materialized views containing pre-aggregated data are some of the proposed technologies that have been implemented and used in real-life systems. All these solutions are implemented on top of the very successful relational database technology.

One of the recently proposed techniques is the Multidimensional Hierarchical Clustering and Hierarchical Indexing technique ([MRB99, KS01]). This technique is based on the star-schema organization with the focus on the usage of clustering access methods. The main goal is the reduction of the number of I/O operations required to answer the large aggregate queries. According to this method the fact table of a data warehouse is stored clustered with respect to the dimension hierarchies by using special attributes called *hierarchical surrogate keys*. Since most aggregation queries apply restrictions on the dimension hierarchies, the fact-table data needed to answer such queries are found clustered in a relatively small number of disk pages, improving the performance.

In the presence of this new data organization schemata new processing and optimization techniques have been recently proposed ([KTS02, PER03, TT01]). One of these techniques, called Hierarchical Pre-Grouping, exploits the existence of the hierarchical surrogate keys in order to improve the query execution time even further. The technique modifies the query execution plan in an attempt to aggregate the fact-table tuples as early as possible and avoid redundant joins.

In order to illustrate the importance of this technique we next present an example scenario where Hierarchical Pre-Grouping is applied according to the heuristic algorithm presented in [KTS02].

Example: Applying Hierarchical Pre-Grouping

Consider the simplified data warehouse schema shown in Figure 1. The data warehouse stores sales transactions recorded per item, store, customer and date. It contains one fact table *SALES_FACT*, which is defined over the dimensions: *PRODUCT*, *CUSTOMER*, *DATE* and *LOCATION*. The single measure of *SALES_FACT* is *sales* representing the sales value for an item bought by a customer at a store on a specific day.

Each dimension is stored in a dimension table and it is organized according to a hierarchy. For example, the *LOCATION* dimension is organized into a hierarchy with three levels: Store-Area-Region. Stores are grouped into geographical areas and the areas are grouped into regions. The attributes *store_id*, *area* and *region* are called hierarchical attributes because they are used to define the hierarchy in the dimension table. It is important to note that there are functional dependencies among the hierarchical attributes. In our example schema *store_id* functionally defines *area*, which functionally defines *region*. Hence, *store_id* is the key of this dimension table. The hierarchies of our example schema are shown in Figure 2.

Each dimension table contains a hierarchical surrogate attribute (h-surrogate) named *hsk*. This attribute represents an encoding of the entire chain of hierarchical attributes. For example, the *hsk* attribute of the *LOCATION* dimension is assigned the values $oc_1(region)/oc_2(area)/oc_3(store\_id)$, where the functions oc_i (i = 1,2,3) define a numbering scheme for each hierarchy level and assign some *order-code* to each hierarchical attribute value. An important property of the h-surrogate attributes is that we can extract from their value any part of the path they encode. For example, from the *hsk* attribute of the *LOCATION* dimension we can extract the $oc_2(area)$ component (denoted with *hsk.area*) and obtain an encoding of the value of the *area* attribute. Due to the encoding function used there is a one-to-one mapping between the values of the component (like *hsk.area*) and the values of the corresponding attribute (*area*). Consequently, functional dependencies hold for each pair of h-surrogate component and hierarchical attribute. Obviously the *hsk* attributes are candidate key-attributes of their dimension table.

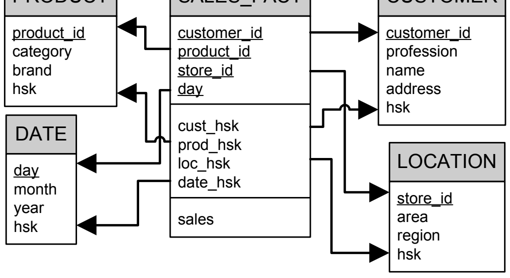

Figure 1: The schema of the data warehouse

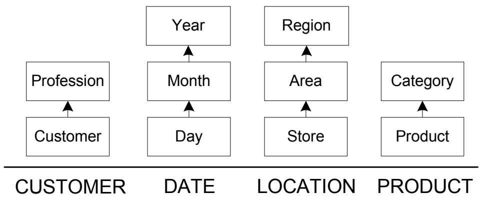

Figure 2: The dimension hierarchies

The main reasons for having these special h-surrogate attributes is the hierarchical clustering and indexing of the fact tables. The fact table contains foreign keys referencing the h-surrogate attributes of each dimension and uses these foreign h-surrogate keys to organize and cluster the fact table. For example, the *SALES_FACT* table contains the attribute *loc_hsk* as a foreign key to the *hsk* attribute of the *LOCATION* dimension. The existence of these additional foreign keys in the fact table allows the Hierarchical Pre-Grouping transformation to optimize the execution of star-join queries.

Consider the following SQL query on the previously described schema:

```
SELECT L.area, P.brand, SUM(F.sales)
FROM SALES_FACT F, LOCATION L, DATE D,
     PRODUCT P
 WHERE F.day = D.day AND F.store_id =
 L.store_id AND F.product_id = P.item_id
 GROUP BY L.area, D.month, P.brand
```

The straightforward execution plan for this query is sketched in Figure 3. The fact table is joined with the dimension tables using the equality join conditions mentioned in the query and the result of the join is grouped and aggregated according to the attributes of the GROUP BY clause.

The Hierarchical Pre-Grouping transformation can modify this execution plan in three different ways:

1. Since the *month* attribute is not part of the result we can use the component *date_hsk.month* for which we know that there is a one-to-one mapping among *date_hsk.month* and *month* in order to group the fact table tuples. This way we no longer need the join with the *DATE* dimension table.
2. In a similar way we can use the component *loc_hsk.area* instead of the *area* attribute in order to group the tuples. In this case we still need the join with the *LOCATION* dimension table in order to get the actual values of the *area* attribute but this join is not performed on the *store_id* as defined by the query. One way to do this join is to group the *LOCATION* dimension table on *hsk.area* and join using the equality condition *loc_hsk.area*=*hsk.area*.
3. Finally, although there is no attribute in the fact table that would allow us to group on *brand*, we can still do a partial grouping on *prod_hsk*, which is a foreign key of *PRODUCT*, and after the join with this dimension table we will have to aggregate some of

the previously created group-tuples based on the value of the *brand* attribute. This modification splits the aggregation operation into two stages: one before the join with *PRODUCT* and the second after this join.

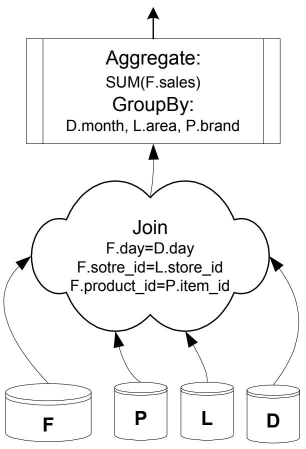

Figure 3: The original execution plan

The modified execution plan appears in Figure 4 and is expected to perform better than the original plan in most cases. However, a cost-based optimization approach ([TKS02]) can detect cases where the application of Hierarchical Pre-Grouping is not beneficial.

□

The above example demonstrates the significant impact that Hierarchical Pre-Grouping can have on the execution plan of a star-join query. The experimental measurements ([KTS02, PER03]) have shown that this technique can reduce the time needed to answer large aggregate queries to less than 50%.

These results have motivated us to study the details of the Hierarchical Pre-Grouping technique and identify the conditions under which it can be applied in order to make this technique applicable to database schemata that do not contain hierarchical surrogate keys. In this paper we present the main results of this work which include:

1. The formal, declarative definition of the Generalized Hierarchical Pre-Grouping optimization technique in the form of an algebraic transformation.
2. The definition of the conditions under which the transformation can be applied in terms of integrity constraints. This result makes the transformation available to other database schemata where hierarchical surrogate keys are not available. Proof for the sufficiency of the defined conditions is also provided.
3. The identification of a number of simple transformations that can be considered as the building blocks from which Hierarchical Pre-Grouping is constructed. These building blocks can be used to identify other types of transformations where integrity constraints play an important role. In fact, one these blocks, the Surrogate-Join transformation, can be directly used for query optimization.

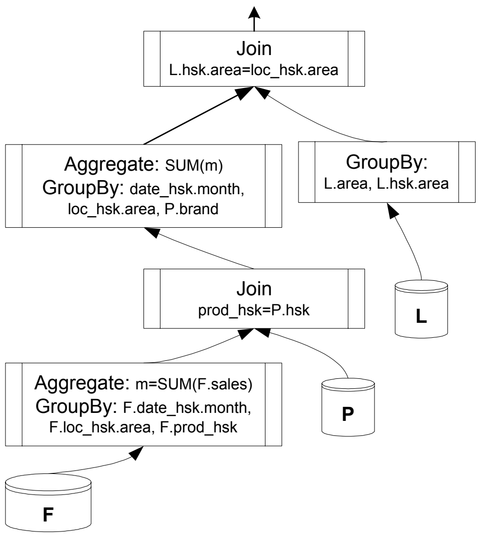
Join
L.hsk.area=loc_hsk.area
Aggregate: SUM(m)
GroupBy: date_hsk.month, loc_hsk.area, P.brand
GroupBy:
L.area, L.hsk.area
Join
prod_hsk=P.hsk
L
Aggregate: m=SUM(F.sales)
GroupBy: F.date_hsk.month, F.loc_hsk.area, F.prod_hsk
P
F

Figure 4: The transformed execution plan

The rest of the paper is organized as follows. Section 2 presents related work and briefly comments on how our results use or extend previous work. In Section 3 we introduce a number of terms and explain the notation used throughout this paper. Section 4 presents the Hierarchical Pre-Grouping transformation and its algebraic general form while Section 5 contains the main results of our work. In this section we define a number of simple and complex transformations along with the conditions required to apply them. These transformations are used in the proof of the final theorem. The final theorem defines sufficient conditions for the application of the Generalized Hierarchical Pre-Grouping transformation. Finally, Section 6 summarizes our contribution and presents directions for further research.

2. Related work

Throughout this paper we adopt the relational model with bag semantics, assuming (and allowing) each relation to be a bag. Hence, all our algebraic expressions use the relational algebra operators extended for bags.

The practical importance of bags (initially called multisets) has been recognized from the early days of the relational model. Dayal et al. ([DGK82]) where the first to publish the extension of the relational model and of the relational algebra for bags. Mumick et al. ([MPR90]) presented a formal treatment of bags and aggregate operators studying the semantics and showing that the Magic-Set technique can be extended to support these ‘non-relational features’. Albert ([Alb91]) provided various important results regarding the algebraic properties of bag data types and the work of Chaudhuri and Vardi ([CV93]) and of Ioannidis and Ramakrishnan ([IR95]) addressed various additional issues related to conjunctive query containment and query equivalence for the relational model with bag semantics.

Many of the transformations presented in this paper rely on the existence of integrity constraints. The idea of using integrity constraints to optimize queries is not new. In the area of Semantic Query Optimization, starting with King ([King81]), researchers have proposed various ways to use integrity constraints for optimization. The relation elimination proposed by Shenoy and Ozsoyoglu ([SO87]) and the elimination of an unnecessary join described by Sun and Yu ([SY94]) are very similar to the one that we use in our transformations. The difference is that our transformations are applicable on bags while the previous semantic query optimization techniques were discussed only for sets.

In the presence of bag semantics one needs to control duplicate elimination in order to efficiently take advantage of integrity constraints. Group and aggregate operators can be used for this purpose. The manipulation of the grouping and aggregation operations for SQL query optimization has been presented by Kim ([Kim82]) and completed by Dayal ([Day87]). Their proposal was to move a grouping and aggregation operation before a join operation in the query tree. Later, with the intensive usage of relational databases by Decision Support Systems the issue of optimization of aggregate SQL queries was re-examined. The emerged results addressed the problem of having relations with bag semantics.

Chaudhuri and Shim ([Chau94]) defined a number of transformations for the execution trees of SQL queries with aggregate functions. These transformations can move an aggregation operator bellow or above join nodes using key integrity constraints. The most powerful transformation can split the aggregation operator into (at least) two aggregation operators: one of which is applied before the join and the other after the join. The main idea here is that grouping and aggregation can be performed in stages. At each stage the aggregation node groups a number of tuples that were generated during the previous stage, creating group tuples that represent larger groups. This process ends when the desired grouping level is achieved.

Yan and Larson presented a similar result ([YL94]). In a following paper ([YL95]) they extended these results covering the case where aggregate functions are applied on attributes of both sources of a join. Using a COUNT function they proved that even in this case the aggregation operator could be split into two stages: one before the join on one of the sources, and one after the join. An important contribution of this work is the definition of the sufficient (and some times necessary) conditions for applying the transformations, even in the presence of NULL values, and the provision of formal proofs.

Gupta, Harinarayan and Quass ([GHQ95]) clearly state the similarity of the classical (duplicate eliminating) project operator and of the grouping and aggregation operator by defining a combined operator called Generalized Projection (GP). Using this operator they provide a number of rules that can be used to move (or split/combine) a GP in the query execution tree.

The idea of having only one operator that combines project with grouping and aggregation seems quite elegant and it is adopted in this paper. In fact we redefine the Generalized Project operator (denoted with the symbol Π in this paper) in a way that can also include the duplicate retaining project operation. We argue that this general definition of the Π operator simplifies the definition of various transformations.

We consider the main theorem provided by Yan and Larson ([YL95]) to be the most general result regarding the transformation of queries with aggregations. It seems that this theorem can be used to prove all the other transformations defined in the previously mentioned publications (except for some very special results involving MIN and MAX). In this paper we adopt this transformation and use it under the name Split-Π. In fact, we show that the new transformations we discuss are particularly useful when Split-Π can be applied. With the help of additional integrity constraints, Split-Π can be applied more efficiently, even in cases where previously it could not be applied.

3. Notation

In this section we provide a number of definitions required to present our results. We clarify the meaning assigned to relations, we introduce the notation and operators used in our algebraic expressions and define a number of integrity constraints used in following sections. Furthermore, we discuss some interesting properties of aggregate functions. Due to space limitations the reader is referred to the extended version of this paper [TS03] for further details.

Assume the existence of a countable set of attributes $\boldsymbol{A}$. Each attribute $a \in \boldsymbol{A}$ is related to a domain of values *dom*(a). Each domain of values contains the special value NULL. A relation schema $a(\mathbf{R})$ is a set of attributes: $a(\mathbf{R}) \subseteq \boldsymbol{A}$. We define a relation to be a collection of tuples not necessarily distinct (bag). Each tuple is a mapping from the attributes of the relation schema to values in the corresponding domains.

In the following we use the capital letters (X, H, K, G, …) to denote sets of attributes and the bold symbols $\mathbf{R}$, $\mathbf{R_u}$, $\mathbf{R_d}$ to represent relations. If t is a tuple of the relation $\mathbf{R}$ and H a set of attributes of $\mathbf{R}$, we use the notation $t[H]$ to represent the vector of values in the tuple t that correspond to the attributes in H. We also use the relational algebra operators Select (σ) and Cross-Product ($\times$) with their extended definitions for bags.

Note that the evaluation of the selection conditions implies the usage of a three-state logic in the presence of NULLs as defined by SQL. For example the expression $\sigma_{K=K}(\mathbf{R})$ contains all tuples of **R** without the ones that assign a NULL value to any of the attributes in *K*.

3.1 Integrity Constraints

In this paper integrity constraints are used to identify conditions under which various transformations can be applied. The integrity constraints used are:

Functional dependencies: $K \rightarrow H$ denotes that the attributes *K* functionally define each of the attributes in *H*. In the case of functional dependencies the NULL values are treated as ordinary values. So, $K \rightarrow H$ implies that all tuples with the same $t[K]$ part, even containing NULLs, must have the same $t[H]$ part, which can also contain NULLs.

Super-Keys: $A \in \text{SupKey}(\mathbf{R})$ denotes the attributes *A* as a (super) key of **R**. Besides the fact that the attributes *A* functionally define all attributes of **R**, we require that $t[A]$ can act as a unique identifier for all the tuples in **R** that contain no NULL value in $t[A]$. However, there can be many duplicate tuples in **R** that contain a NULL value for some attribute of *A*. This flexibility is removed when a Super-Key is called a *Strict Super-Key*. So, when $A \in \text{Strict-SupKey}(\mathbf{R})$ we have as an additional condition that the $t[A]$ vector acts as a unique identifier for all tuples in **R,** even when $t[A]$ contains NULL values. If a relation has a Strict Super-Key then this relation can contain no duplicate tuples. In this case the relation can be considered as a set of tuples.

Inclusion dependencies: Let $\mathbf{R_u}$, $\mathbf{R_d}$ be two relations and H_u, K_d be two sets of attributes so that H_u contains only attributes of $\mathbf{R_u}$ and K_d contains only attributes of $\mathbf{R_d}$. We say that that there is an inclusion dependency restricting H_u to K_d when the values of the H_u attributes of $\mathbf{R_u}$ are restricted to the values of the attributes K_d in $\mathbf{R_d}$:

$$\forall t \in \mathbf{R_u} : \exists t' \in \mathbf{R_d} \wedge t[H_u] \overset{n}{=} t'[K_d] .$$

In the above expression the equality predicate treats NULL values as any other ordinary value and does not use the three-state logic of the SQL equality predicate.

An inclusion dependency is called *strict* when in addition to the above conditions there is no tuple in $\mathbf{R_u}$ for which $t[H_u]$ contains a NULL value.

3.2 Generalized Aggregate functions

In order to simplify notation we need to define *generalized aggregate functions*. Just like ordinary aggregate functions, the generalized aggregate functions are applied on a bag of tuples and produce as a result a single value. Note that a bag does not define an order for its members so we do not discuss aggregate functions that require any kind of ordering of the input tuples. The new property of the generalized aggregate functions is an additional parameter that controls the multiplicity of each input tuple. We use the notation $Ag(M;x)$ for generalized aggregate functions. The parameter *M* defines the set of attributes used to compute the result value while the *x* parameter defines the multiplicity with which each input tuple is considered. For example *SUM*(*salary*;2) computes the sum of salaries adding twice the salary value of each tuple. Note that *x* may also be an attribute of the input tuple or even an arithmetic expression. In any case the *x* parameter must at all times be a non-negative integer number. In the following we use the term aggregate function to refer to a generalized aggregate function.

In order to present a set of aggregate functions in a compact way we use the notation: $A=F(\{Ag^i(M^i;n^i)\})$ where $A=\{a_1,a_2,\ldots,a_k\}$ is a set of result attributes the values of which are computed by the function *F*. The function *F* uses in its computations the values of aggregate functions: $Ag^1(M^1;n^1)$, $Ag^2(M^2;n^2)$, …, $Ag^k(M^m;n^m)$, where *m* and *k* are arbitrary non-negative integers implied by the context.

Furthermore, we write $\{\mathbf{Ag}\}$ as a shortcut to $\{Ag^i(M^i;n^i)\}$ and we use the notation $Ag(N)$ to represent $Ag(M;x)$. In this case we are not interested in the precise definition of the parameter *x* but only on the set of attributes on which the aggregate function depends. Hence, *N* contains all attributes of *M* plus the attributes referenced in *x* (if any).

3.3 The Generalized Projection operator Л

The generalized project operator is a combination of the classical (duplicate eliminating) project operator, the duplicate preserving project operator and of the grouping and aggregation operator. As shown in the next sections, these operators have very similar properties with regard to algebraic transformations. The unification of these operators in a single operator does not alter the expressive power of the defined algebra but simplifies the definition of various transformation rules.

The generalized project operator is defined as:

$$Л^{G}_{S_1,S_2,\ldots,S_k,A=F(\{\mathbf{Ag}\})}(\mathbf{R})$$

The relation **R** is the input on which the operator is applied. The set *G* is the set of grouping attributes of **R**, while S_1, S_2, … S_k are sets of selected (projected) attributes. The set $A=\{a_1, a_2, \ldots, a_m\}$ is the set of computed and projected attributes.

Assuming that $S=\{S_1 \cup S_2 \cup \ldots \cup S_k\}$ is the set of all selected (projected) attributes of **R**, in the above definition we do not require that $S \subseteq G$ as one would expect. In order for the operator definition to be correct *S* must be functionally dependent on *G* ($G \rightarrow S$) in **R**. Obviously when $S \subseteq G$ this property holds for any **R**. Note that an immediate consequence of this flexible definition is that the Л operator can produce duplicate tuples.

The Л operator has two special forms that need to be explained: when the *G* set is missing the operator does no grouping and the aggregation functions in $\{\mathbf{Ag}\}$ are applied for each tuple of **R** separately. In this case the operator functions as a typical project operator that maintains duplicates. The second special form of the operator is when $G=\varnothing$. In this case the operator outputs exactly one tuple after applying the aggregation functions on all tuples in **R**. If **R** is empty then the operator still outputs a tuple. Compatibility with SQL requires that in this case all the *COUNT* aggregation functions return 0.

The semantics of the operator Π can be defined as follows: $\Pi^{G}_{S_1,S_2,\ldots,S_k,A=F(\{\mathbf{Ag}\})}(\mathbf{R})$ groups the tuples in **R** according to the values of the attributes defined in *G*. The tuples in each group have the same values for the attributes in *G*. The grouping process treats NULL as a regular value. If $G=\varnothing$ a group is formed with all tuples of **R** (even zero). If *G* is missing, each tuple of **R** forms a group (if **R** is empty the operator produces zero groups). For each group the operator returns an output tuple. In the output tuple the value of each attribute $s \in S_1 \cup S_2 \cup \ldots \cup S_k$ is equal to the value of the *s* attribute of any tuple in the group. The attributes in *A* are computed by the *F* function after the aggregate functions {**Ag**} are computed from the tuples of the group.

Using the Π operator we can use the formula $\Pi^{K_d}_{K_d}(\mathbf{R_d}) \supseteq \Pi^{H_u}_{H_u}(\mathbf{R_u})$ to define an inclusion dependency restricting the values of the attributes H_u to the values of the attributes K_d. In this formula, the results of the Π operators are known to be sets, allowing us to use the symbol $\supseteq$ with its usual set semantics. In a similar way, the formula $\Pi^{K_d}_{K_d}\sigma_{K_d=K_d}(\mathbf{R_d}) \supseteq \Pi^{H_u}_{H_u}(\mathbf{R_u})$ defines a strict inclusion dependency.

3.4 Interesting classes of aggregate functions

In general we need not restrict the aggregate functions to the set of SQL2 aggregate functions (*SUM*, *MIN*, *MAX*, *COUNT*, *AVERAGE*). Any aggregate function can be used as long as it has the required properties. The properties that may be required for an aggregate function are:

Definition 1: (*Distribution property*)
We say that an aggregate function $Ag(M;x)$ *is decomposable into* $Ag_O(Ag_I())$ if and only if for every compatible relation **R** containing the attributes in *M* and *x* and for every attribute sets *G*, *G*1 of **R,** so that $G \subseteq G1$, there are two aggregate functions $Ag_I(M;x)$ and $Ag_O(y;z)$ so that: $\Pi^{G}_{a=Ag(M;x)}(\mathbf{R}) = \Pi^{G}_{a=Ag_O(y;1)}(\Pi^{G1}_{y=Ag_I(M;x)}(\mathbf{R}))$

Definition 2: (*Identity property*)
We say that an aggregate function $Ag(M;x)$ has the identity property if and only if for any compatible tuple *t* the result of applying $Ag(M;1)$ on the tuple *t* is $t[M]$.

It is trivial to show that the typical SQL functions *SUM*, *MIN*, *MAX* have both of the above properties and that *COUNT* is decomposable into *SUM*(*COUNT*()).

4. The Pre-Grouping transformation

The *Hierarchical Pre-Grouping* transformation (called Pre-Grouping for short) has been presented as a heuristic optimization algorithm in the context of processing star queries on hierarchically clustered fact tables. In the area of Data Warehouses and ROLAP, the schema of the database is almost certain to be based on a number of star schemata containing fact tables and dimension tables ([CD97]). The usage of hierarchically clustered and indexed fact tables is a recent approach ([MRB99, KS01]) that aims to minimize the number of I/O operations required to answer heavy aggregate queries. The major bottleneck in evaluating such queries has been the join of the central (and usually very large) fact table with the surrounding dimension tables (also known as a *star-join*).

In the above context, the application of the Pre-Grouping transformation depends on the existence of the hierarchical surrogate keys (h-surrogates). Using the properties of these special attributes heuristic algorithms have been proposed to modify the initial query execution plan in order to remove redundant joins and group fact table tuples as early as possible, even if this requires grouping them in stages ([PER03, KTS02, TT01]).

As already mentioned, the Pre-Grouping transformation has been proven to be a very efficient optimization technique. The previously published measurement results ([KTS02, PER03]) show that Pre-Grouping can speed up typical OLAP star-join queries by a factor of more than two.

In this paper we try to generalize the applicability of Pre-Grouping by separating it from the context in which it was defined and identify the elementary concepts on which it is based. In order to achieve this goal we first provide a general definition of the Pre-Grouping transformation in the form of an algebraic transformation.

4. 1 The Generalized Pre-Grouping transformation

The Generalized Pre-Grouping transformation is applied to *star-join queries*. These queries join a central (fact) table with a number of (dimension) tables using foreign keys. The star-join query may define a number of *local* constraints on the values of various attributes. The characterization “local” means that the constraints do not contain expressions involving attributes of more than one table. Finally, the star-join query may define a number of grouping attributes and require the computation of a number of aggregate functions on the attributes of the central (fact) table. Obviously, the example query in section 1 is a star-join query.

Using the Π operator defined in the previous section we can represent the core part of a star-join query with the algebraic expression E1:

$$\text{E1:}\quad \Pi^{S_u,S_{d1},S_{d2},S_{d3}}_{S_u,S_{d1},S_{d2},A=F(\{Ag^i(M^i_u)\})}\sigma_{H_{u1}=K_{d1},H_{u2}=K_{d2},H_{u3}=K_{d3}}$$
$$(\mathbf{R_u} \times \mathbf{R_{d1}} \times \mathbf{R_{d2}} \times \mathbf{R_{d3}})$$

The symbols $\mathbf{R_u}$, $\mathbf{R_{d1}}$, $\mathbf{R_{d2}}$, $\mathbf{R_{d3}}$ represent relations or views (expressions). The attributes S_u, M_u, H_{u1}, H_{u2}, H_{u3} are attributes of $\mathbf{R_u}$: S_u are grouping and selected attributes, M_u are the attributes on which the aggregate functions $Ag^i()$ are computed and H_{u1}, H_{u2}, H_{u3} are the foreign key attributes for $\mathbf{R_{d1}}$, $\mathbf{R_{d2}}$, $\mathbf{R_{d3}}$. The attributes S_{d1} are attributes of $\mathbf{R_{d1}}$ and K_{d1} is a (strict super) key of $\mathbf{R_{d1}}$. Likewise, S_{d2}, K_{d2} are attributes of $\mathbf{R_{d2}}$ and S_{d3}, K_{d3} are attributes of $\mathbf{R_{d3}}$.

It is rather trivial to see that by using appropriate substitutions the expression E1 can represent a star-join query with an arbitrary number of tables. Any of the $\mathbf{R_{d1}}$, $\mathbf{R_{d2}}$, $\mathbf{R_{d3}}$ can represent an expression containing a join or a cross-product of a number of (dimension) tables. A local condition on a relation (or even a view) can be expressed by substituting the corresponding symbol with a selection expression: for example we can substitute $\mathbf{R_{d1}}$ with $\sigma_c(\mathbf{R_{d1}})$. Also, any of the $\mathbf{R_{d1}}$, $\mathbf{R_{d2}}$, $\mathbf{R_{d3}}$ can be removed from the expression along with the corresponding join condition and the corresponding attributes in the Π operator.

A thorough study of the heuristic algorithms used to define Hierarchical Pre-Grouping in previous publications reveal that the following expression E2 can represent the result of applying Pre-Grouping to the expression E1.

$$\text{E2:}\quad \Pi_{S_u,S_{d1},S_{d2},A}\sigma_{SH_{u2}=SK_{d2}}(\Pi^{S_u,S_{d1},SH_{u3},SH_{u2}}_{S_u,S_{d1},SH_{u3},SH_{u2},A=F(\{Ag^i_O(m^i_I;1)\})}$$
$$(\sigma_{SH_{u1}=SK_{d1}}(\Pi^{S_u,SH_{u3},SH_{u2},SH_{u1}}_{S_u,SH_{u3},SH_{u2},SH_{u1},\bigcup_i m^i_I=Ag^i_I(M^i_u)}(\mathbf{R_u})\times$$
$$\Pi^{SK_{d1}}_{S_{d1},SK_{d1}}(\mathbf{R_{d1}})))\times\Pi^{SK_{d2}}_{S_{d2},SK_{d2}}(\mathbf{R_{d2}}))$$

Note that the join with the relation $\mathbf{R_{d3}}$ is missing and the grouping attribute S_{d3} is substituted with the attribute SH_{u3}. Also, the remaining join conditions have changed and four Π operators are used to perform the grouping and aggregation and one (the first one) to perform projection.

The E1 expression was carefully created in order to contain all situations on which Pre-Grouping can be applied. In fact the $\mathbf{R_{d1}}$, $\mathbf{R_{d2}}$, $\mathbf{R_{d3}}$ relations (or expressions) are used in order to classify the (dimension) tables of the star-join query according to the type of modification that Pre-Grouping can apply to them:

1. Some (dimension) tables may be skipped from the join sequence completely ($\mathbf{R_{d3}}$). This is the case for the *DATE* dimension in our example.
2. Some (dimension) tables are joined after all grouping operations ($\mathbf{R_{d2}}$). This is the case for the *LOCATION* dimension in our example.
3. Aggregation may be split into two stages: before and after the join with some (dimension) tables ($\mathbf{R_{d1}}$). This is the case for the *PRODUCT* dimension in our example.

One can check that the optimized execution plan presented in Figure 4 represents an instance of E2.

Definition 3: *Generalized Hierarchical Pre-Grouping*: We define Generalized Hierarchical Pre-Grouping as the transformation of the expression E1 into the expression E2: E2=E1.

The previous definition completes the generalization of the Pre-Grouping transformation and its definition in an algebraic form. Note that none of the expressions use h-surrogates or dimension hierarchies. Using these general expressions, the Pre-Grouping transformation can be applied in databases that do not use h-surrogates or multidimensional hierarchical clustering and indexing. The conditions required to apply the Generalized Hierarchical Pre-Grouping are the subject of the next section.

5. Decomposing Pre-Grouping

In this section we start with the algebraic expressions E1 and E2 that define the Generalized Pre-Grouping transformation. Our goal is to transform the expression E1 into E2 using a sequence of *simple* transformations for which we can prove the sufficient conditions required in order to apply them. This task mainly achieves two goals:

- Identify and prove that a particular set of conditions is sufficient in order to apply the Generalized Pre-Grouping transformation.
- Identify the relationship of the Pre-Grouping transformation to other similar and more elementary transformations like the "eager groupby-count" defined by Yan and Larson ([YL95]) and the "Literal Elimination" technique described in the Semantic Query Optimization area ([CGM90]).

An additional result of this decomposition task is the definition of the Surrogate-Join transformation. This transformation, as described in detail in the following, can use a number of integrity constraints in order to modify a join expression by altering the join condition.

In the following we define a number of simple and complex transformations and provide propositions and theorems proving the conditions under which each transformation can be applied. All omitted proofs can be found in the extended version of this paper [TS03].

5.1 Simple transformations

Simplify-GroupBy: The transformation removes one or more grouping attributes of the Π operator when it is known that the remaining grouping attributes functionally define the removed ones. The algebraic expression of the Simplify-GroupBy transformation is:

$$\Pi^{G1,G2}_{S,A=F(\{Ag^i(M^i)\})}(\mathbf{R})\equiv\Pi^{G1}_{S,A=F(\{Ag^i(M^i)\})}(\mathbf{R})$$

Proposition: Let both sides of the Simplify-GroupBy transformation be valid expressions. The transformation is valid if and only if $G2$ is functionally dependent on $G1$ ($G1 \rightarrow G2$) in $\mathbf{R}$.

Simplify-Aggregations: The transformation removes a number of redundant aggregation functions simplifying the expression of a Π operator. This transformation is applicable when the Π operator performs no grouping and acts only as a duplicate-retaining project operator. The algebraic expression of the transformation is:

$$\Pi_{S,A=F(\{Ag_1^i(M^i;1)\},\{\mathbf{Ag_2}\})}(\mathbf{R}) \equiv \Pi_{S,A=F(\{M^i\},\{\mathbf{Ag_2}\})}(\mathbf{R})$$

Proposition: Let both sides of the Simplify-Aggregations transformation be valid expressions. The transformation is valid for any **R** if and only if all the aggregate functions $Ag_1{}^i()$ have the identity property.

Remove-GroupBy: The transformation is the combination of the previous two transformations. It removes the grouping attributes and the aggregation functions of a of Π operator when we know that they are redundant. The algebraic expression of the Remove-GroupBy transformation is:

$$\Pi^{G}_{S,A=F(\{Ag^i(M^i;1)\})}(\mathbf{R}) \equiv \Pi_{S,A=F(\{M^i\})}(\mathbf{R})$$

Proposition: Let both sides of the Remove-GroupBy transformation be valid expressions. The transformation is valid if all the aggregation functions $Ag^i()$ have the identify property and $G\in$SupKey(**R**).

Remove-Join: When the attributes of a joining relation are not used and all tuples of the other join relation appear exactly once in the result of the join then the join is redundant. The Remove-Join transformation removes such a redundant join. The algebraic expression of the Remove-Join transformation is:

$$\Pi^{G_u}_{S_u,A=F(\{Ag^i(M_u^i)\})}\sigma_{H_u=K_d}(\mathbf{R_u}\times\mathbf{R_d}) \equiv \Pi^{G_u}_{S_u,A=F(\{Ag^i(M_u^i)\})}(\mathbf{R_u})$$

Proposition: Let both sides of the Remove-Join transformations be valid expressions and let H_u contain only attributes of $\mathbf{R_u}$ while K_d contains only attributes of $\mathbf{R_d}$. The transformation is valid if and only if the following conditions hold:

C1: $\Pi^{K_d}_{K_d}\sigma_{K_d=K_d}(\mathbf{R_d}) \supseteq \Pi^{H_u}_{H_u}(\mathbf{R_u})$

C2: $\Pi^{\varnothing}_{MAX(a)}\Pi^{H_u}_{a=COUNT(H_u)}\sigma_{H_u=K_d}(\Pi^{H_u}_{H_u}(\mathbf{R_u})\times\mathbf{R_d}) \in \{1, \mathrm{NULL}\}$

Condition C1 defines a strict inclusion dependency (see section 3) while condition C2 is quite difficult to define in terms of integrity constraints. Instead of C2 one can use a more restrictive condition and require K_d to be a Super-Key of $\mathbf{R_d}$ or at least $K_d\in$SupKey($\Pi_{K_d}(\mathbf{R_d})$). However, this condition is not required (but it is sufficient along with the condition C1) in order for the transformation to be valid.

Normalization: The transformation replaces a relation with the natural join of two generalized projections of this relation. The goal of the transformation is to separate the functionally dependent attributes from other attributes of the relation. The algebraic expression of Normalization is:

$$\mathbf{R} \equiv \Pi_R\sigma_{S1=G1}(\Pi_{R-G1\cup G2,S1=G1}(\mathbf{R})\times\Pi^{G1}_{G1,G2}(\mathbf{R}))$$

Proposition: The Normalization transformation is valid whenever the right-hand side is a valid expression.

The right-hand side expression requires that $G1$ and $G2$ are attributes of **R** and that $G2$ is functionally dependent on $G1$ ($G1\rightarrow G2$) in **R**. Note that in this transformation the Π operator acts only as a project operator, some times eliminating duplicates and some times retaining them. The correctness of the transformation can easily be proven using the Remove-Join and Simplify-GroupBy transformations.

An interesting remark regarding this transformation is its applicability in the process of normalizing a database schema into 3rd Normal Form. Any such normalization process implicitly uses this transformation.

Add-Join-Predicate: The transformation simply adds (or removes) a redundant equi-join predicate to an existing equi-join. With this transformation one can modify and even replace join conditions. The algebraic expression of the Add-Join-Predicate transformation is:

$$\sigma_{H_u=K_d}(\mathbf{R_u}\times\mathbf{R_d}) \equiv \sigma_{H_u=K_d\wedge SH_u=SK_d}(\mathbf{R_u}\times\mathbf{R_d})$$

Proposition: Let both sides of the Add-Join-Predicate transformation be valid expressions. The transformation is valid if the following conditions hold:

C1: $\Pi^{K_d,SK_d}_{K_d,SK_d}\sigma_{K_d=K_d\wedge SK_d=SK_d}(\mathbf{R_d}) \supseteq \Pi^{H_u,SH_u}_{H_u,SH_u}\sigma_{H_u=H_u}(\mathbf{R_u})$

C2: $\Pi^{\varnothing}_{MAX(a)}\Pi^{H_u}_{a=COUNT(H_u)}\sigma_{H_u=K_d}(\Pi^{H_u}_{H_u}(\mathbf{R_u})\times\mathbf{R_d}) \in \{1, \mathrm{NULL}\}$

Condition C1 can be satisfied using SQL integrity constraints. If $\{H_u, SH_u\}$ are defined as not NULL and as foreign keys referencing $\{K_d, SK_d\}$ then we can prove C1. Also, just like in the case of the Remove-Join transformation, we can use the constraint $K_d\in$SupKey($\mathbf{R_d}$) in order to prove condition C2. Note that the above conditions are required in order to make sure that the removed condition SH_u=SK_u is redundant with respect to the remaining condition H_u=K_d.

5.2 Complex transformations

Split-Π: The transformation pushes an aggregation operation that is applied on the result of a join into the members of the join. This is achieved by splitting the aggregation operation into two aggregation stages. The first one is performed before the join on one of the joined sources and the other is performed on the result of the join in place of the original aggregation operation (see [YL95] for a detailed discussion of this complex transformation). The algebraic expression of the Split-Π transformation is:

$$\Pi^{G_d,G_u}_{S_d,S_u,A=F(\{Ag_u^i(M_u^i)\},\{Ag^i(M_d^i\cup C_u^i;g^i(N_d^i\cup Z_u^i))\})}\sigma_{c(E_d,E_u)}(\mathbf{R_u}\times\mathbf{R_d}) \equiv$$
$$\Pi^{G_d,G_u}_{S_d,S_u,A=F(\{Ag_O^i(m_I^i;1)\},\{Ag^i(M_d^i\cup C_u^i;g^i(N_d^i\cup Z_u^i)*x_u)\})}\sigma_{c(E_d,E_u)}($$
$$\Pi^{NG_u}_{E_u,G_u,S_u,\bigcup_i C_u^i,\bigcup_i Z_u^i,\bigcup_i m_I^i=Ag_I^i(M_u^i),x_u=COUNT(TID)}(\mathbf{R_u})\times\mathbf{R_d})$$

In the above expressions the *COUNT*(*TID*) aggregate function counts the number of tuples in each group. Note that this is the only transformation that uses the analytic notation *Ag*(*X*;*g*(*Y*)) for the aggregate functions. In fact, it is due to this transformation that we have chosen to adopt this notation. Without this notation the formal definition of the impact of the term $g^i()*x_u$ would be quite complicated. A final remark is that the selection operator is not essential to the transformation and we could have also defined it without it.

Theorem 5.1: Let both sides of the Slit-Л transformation be valid expressions. The transformation is valid if the following conditions hold:

C1: $NG_u \rightarrow G_u \cup S_u \cup C^i_u \cup Z^i_u$.

C2: The aggregate functions $Ag^i_u()$ are decomposable into $Ag^i_O(Ag^i_I())$

This theorem is equivalent to the eager-groupby-count theorem presented by Yan and Larson ([YL95]). The only minor difference is that we are using a different notation and we allow the $Ag^i()$ functions to use not only attributes of $\mathbf{R_d}$ but also attributes of $\mathbf{R_u}$. Also, due to the way we have defined aggregate functions, we need not impose any conditions on the properties of the $Ag^i()$ aggregate functions as done by the eager-groupby-count theorem. In fact, the extra condition imposed by that theorem is not really restrictive.

Surrogate-Join: This transformation modifies an equi-join expression and provides an alternative way to join the two sources. The goal is to remove any redundant parts that exist in the joined sources and join only the remaining necessary sections. In order to achieve this goal the transformation replaces the join condition with a different one while performing appropriate projections and duplicate elimination on the joined sources. The algebraic definition of Surrogate-Join is:

$$\Pi_{S_u,S_d}\sigma_{H_u=K_d}(\mathbf{R_u}\times\mathbf{R_d}) \equiv$$
$$\Pi_{S_u,S_d}\sigma_{SH_u=SK_d}(\Pi_{S_u,SH_u}(\mathbf{R_u})\times\Pi^{SK_d}_{S_d,SK_d}(\mathbf{R_d}))$$

Theorem 5.2: Let both sides of the Surrogate-Join transformation be valid expressions. The transformation is valid if the following conditions hold:

C1: $\Pi^{K_d,SK_d}_{K_d,SK_d}\sigma_{K_d=K_d \wedge SK_d=SK_d}(\mathbf{R_d}) \supseteq \Pi^{H_u,SH_u}_{H_u,SH_u}\sigma_{H_u=H_u}(\mathbf{R_u})$

C2: $\Pi^{\varnothing}_{MAX(a)}\Pi^{H_u}_{a=COUNT(H_u)}\sigma_{H_u=K_d}(\Pi^{H_u}_{H_u}(\mathbf{R_u})\times\mathbf{R_d}) \in \{1, \mathrm{NULL}\}$

Note that in order to apply the Surrogate-Join transformation an additional constraint must hold: $SK_d \rightarrow S_d$. Without this constraint the right-hand side expression is not valid.

Proof: Our approach is to split the relation $\mathbf{R_d}$ into two parts separating the attributes SK_d and S_d from the rest. We can do that using the Normalization transformation. The redundant part of $\mathbf{R_d}$ can then be removed with the Remove-Join transformation while the new join condition is introduced with the Add-Join-Predicate transformation.

Surrogate-Join application example

Consider the following example. A web hosting company is maintaining a database with statistics about the web sites it hosts. The company uses a number of servers and each hosted page is assigned to a particular server. Every month the company is making available to its customers a number of materialized views with aggregated results. One of these views, called **Page_Server_Hits**, reports the total hits for each page along with the total hits for the server to which the page is allocated. The schema of this view contains (at least) the attributes: *PageID, PageHits, ServerID, ServerHits*. A second view is made available per client containing additional details about the pages owned by the particular client. This second view, called **Page_Hour_Hits**, contains the hits of each page aggregated separately for each hour of the day. So, in this view there are 24 tuples describing the hits of each page. The attributes of **Page_Hour_Hits** contain at least the attributes: *PageID, ServerID, Hour, HourPageHits*.

If a client wants to compute the percentage *HourPageHits*/*ServerHits* for each of his/her pages and each of the 24 hours, he/she could use the following SQL query:

```
SELECT PageID, HourPageHits/ServerHits
FROM Page_Hour_Hits P, Page_Server_Hits S
WHERE P.PageID=S.PageID
```

However, using the semantics implied by the schema there is a different query that produces the same result:

```
SELECT PageID, HourPageHits/ServerHits
FROM Page_Hour_Hits P,
 (SELECT ServerID, ServerHits
  FROM Page_Server_Hits
  GROUPBY ServerID, ServerHits) S
WHERE P.ServerID=S.ServerID
```

The second query is the result of applying the Surrogate-Join transformation on the first one. We know that the view **Page_Hour_Hits** contains page IDs that also appear in the **Page_Server_Hits** and that in both views a page ID is related to the same server ID. These facts satisfy the condition C1 of the Surrogate-Join theorem. With the additional knowledge that each page ID appears only once in the view **Page_Server_Hits**, the condition C2 is also satisfied allowing us to apply the Surrogate-Join transformation. □

Surrogate-Join-Early-GroupBy: The transformation is similar to the Split-Л transformation, only that it does not divide the aggregation operation into two parts. The complete aggregation operation is performed on only one of the joined sources. In order to achieve this *push-down* of the aggregation operation the transformation may need to modify the join condition and eliminate possible duplicate tuples. In fact, the transformation can be considered as a combination of Surrogate-Join and Split-Л. The algebraic expression of the transformation is:

$$\Pi^{S_u,S_d}_{S_u,S_d,A=F(\{Ag^i(M^i_u)\})}\sigma_{H_u=K_d}(\mathbf{R_u}\times\mathbf{R_d})\equiv$$
$$\Pi_{S_u,S_d,A}\sigma_{SH_u=SK_d}(\Pi^{S_u,SH_u}_{S_u,SH_u,A=F(\{Ag^i(M^i_u)\})}(\mathbf{R_u})\times\Pi^{SK_d}_{SK_d,S_d}(\mathbf{R_d}))$$

Theorem 5.3: Let both sides of the Surrogate-Join-Early-GroupBy be valid expressions. The transformation is valid if the following conditions hold:

C1: $\Pi^{K_d,SK_d}_{K_d,SK_d}\sigma_{K_d=K_d\wedge SK_d=SK_d}(\mathbf{R_d})\supseteq\Pi^{H_u,SH_u}_{H_u,SH_u}\sigma_{H_u=H_u}(\mathbf{R_u})$

C2: $\Pi^{\varnothing}_{MAX(a)}\Pi^{H_u}_{a=COUNT(H_u)}\sigma_{H_u=K_d}(\Pi^{H_u}_{H_u}(\mathbf{R_u})\times\mathbf{R_d})\in\{1,\text{NULL}\}$

C3: $SK_d\leftrightarrow S_d$

C4: The aggregate functions $Ag^i()$ are decomposable into $Ag^i_O(Ag^i_I())$ and the functions $Ag^i_O()$ have the identity property.

Proof: We start with the original expression and apply the Surrogate-Join transformation in order to modify the join condition. This allows us to apply Split-Π without adding H_u as grouping attribute for the inner aggregation performed on $\mathbf{R_u}$ before the join. Finally, using the given conditions, we realize that the outer aggregation operator is redundant. We apply Remove-GroupBy and get the final result. Using a more complicated proof we could show that the condition C4 is not really required.

We now present the final and main theorem of the paper. Recall that the Generalized Hierarchical Pre-Grouping transformation is defined as E1=E2 where:

E1: $\Pi^{S_u,S_{d1},S_{d2},S_{d3}}_{S_u,S_{d1},S_{d2},A=F(\{Ag^i(M^i_u)\})}\sigma_{H_{u3}=K_{d3}}($
$\sigma_{H_{u2}=K_{d2}}(\sigma_{H_{u1}=K_{d1}}(\mathbf{R_u}\times\mathbf{R_{d1}})\times\mathbf{R_{d2}})\times\mathbf{R_{d3}})$
$\Pi_{S_u,S_{d1},S_{d2},A}\sigma_{SH_{u2}=SK_{d2}}(\Pi^{S_u,S_{d1},SH_{u3},SH_{u2}}_{S_u,S_{d1},SH_{u3},SH_{u2},A=F(\{Ag^i_O(m^i_I;1)\})}($

E2: $\sigma_{SH_{u1}=SK_{d1}}(\Pi^{S_u,SH_{u3},SH_{u2},SH_{u1}}_{S_u,SH_{u3},SH_{u2},SH_{u1},\bigcup_i m^i_I=Ag^i_I(M^i_u)}(\mathbf{R_u})\times$
$\Pi^{SK_{d1}}_{S_{d1},SK_{d1}}(\mathbf{R_{d1}})))\times\Pi^{SK_{d2}}_{S_{d2},SK_{d2}}(\mathbf{R_{d2}}))$

Main Theorem: Let both E1 and E2 be valid expressions. The Generalized Hierarchical Pre-Grouping transformation is valid if the following conditions hold:

I1: $SK_{d3}\leftrightarrow S_{d3}$ in $\mathbf{R_{d3}}$

I2: $\Pi^{K_{d3},SK_{d3}}_{K_{d3},SK_{d3}}\sigma_{K_{d3}=K_{d3}\wedge SK_{d3}=SK_{d3}}(\mathbf{R_{d3}})\supseteq\Pi^{H_{u3},SH_{u3}}_{H_{u3},SH_{u3}}\sigma_{H_{u3}=H_{u3}}(\mathbf{R_u})$

I3: $\Pi^{\varnothing}_{MAX(a)}\Pi^{H_{u3}}_{a=COUNT(H_{u3})}\sigma_{H_{u3}=K_{d3}}(\Pi^{H_{u3}}_{H_{u3}}(\mathbf{R_u})\times\mathbf{R_{d3}})\in\{1,\text{NULL}\}$

I4: $\Pi^{K_{d2},SK_{d2}}_{K_{d2},SK_{d2}}\sigma_{K_{d2}=K_{d2}\wedge SK_{d2}=SK_{d2}}(\mathbf{R_{d2}})\supseteq\Pi^{H_{u2},SH_{u2}}_{H_{u2},SH_{u2}}\sigma_{H_{u2}=H_{u2}}(\mathbf{R_u})$

I5: $\Pi^{\varnothing}_{MAX(a)}\Pi^{H_{u2}}_{a=COUNT(H_{u2})}\sigma_{H_{u2}=K_{d2}}(\Pi^{H_{u2}}_{H_{u2}}(\mathbf{R_u})\times\mathbf{R_{d2}})\in\{1,\text{NULL}\}$

I6: $SK_{d2}\leftrightarrow S_{d2}$ in $\mathbf{R_{d2}}$

I7: The aggregate functions $Ag^i()$ are decomposable into $Ag^i_O(Ag^i_I())$ and the functions $Ag^i_O()$ have the identity property.

I8: $\Pi^{K_{d1},SK_{d1}}_{K_{d1},SK_{d1}}\sigma_{K_{d1}=K_{d1}\wedge SK_{d1}=SK_{d1}}(\mathbf{R_{d1}})\supseteq\Pi^{H_{u1},SH_{u1}}_{H_{u1},SH_{u1}}\sigma_{H_{u1}=H_{u1}}(\mathbf{R_u})$

I9: $\Pi^{\varnothing}_{MAX(a)}\Pi^{H_{u1}}_{a=COUNT(H_{u1})}\sigma_{H_{u1}=K_{d1}}(\Pi^{H_{u1}}_{H_{u1}}(\mathbf{R_u})\times\mathbf{R_{d1}})\in\{1,\text{NULL}\}$

Before providing the proof of the theorem we briefly explain the semantics of the previously defined conditions I1-I9.

The first three conditions I1-I3 identify the relationship of the central table $\mathbf{R_u}$ with the table $\mathbf{R_{d3}}$. Condition I1 is used to guaranty that the attributes of $\mathbf{R_{d3}}$ on which grouping is required (S_{d3}) are in the set of attributes SK_{d3} or have a one-to-one mapping to the attributes in this set. Note that the sets K_{d3} and SK_{d3} can contain the exact same attributes. Then, condition I2 requires $\mathbf{R_u}$ to contain foreign keys (H_{u3}, SH_{u3}) connecting the central table $\mathbf{R_u}$ to $\mathbf{R_{d3}}$. The H_{u3} foreign key is required in order to guaranty that the corresponding join operation in E1 joins each tuple of $\mathbf{R_u}$ to at least one tuple of $\mathbf{R_{d3}}$. Condition I3, which mainly requires K_{d3} to be a key of the table $\mathbf{R_{d3}}$, complements I2 and the two conditions together guaranty that, in E1, each tuple of $\mathbf{R_u}$ joins to exactly one tuple of $\mathbf{R_{d3}}$.

The I2 condition is also used in order to require the SH_{u3} attributes of $\mathbf{R_u}$ to reflect the values of the SK_{d3} attributes of $\mathbf{R_{d3}}$. In this way the grouping attributes of $\mathbf{R_{d3}}$ are mapped to attributes in $\mathbf{R_u}$.

In a very similar way the conditions I4, I5 and I6 define the relationship of the central table $\mathbf{R_u}$ with the table $\mathbf{R_{d2}}$. In fact the only difference of $\mathbf{R_{d2}}$ to $\mathbf{R_{d3}}$ in the transformation is that the grouping attributes of $\mathbf{R_{d2}}$ are also projected attributes.

The condition I7 is required in order to be able to split the aggregation operations into two stages. This is required only due to the $\mathbf{R_{d1}}$ table. If the $\mathbf{R_{d1}}$ table does not participate in the transformation then this condition is not needed.

Finally, the conditions I8 and I9 are used in a similar way to I2 and I3. They guaranty that each tuple of $\mathbf{R_u}$ joins, in E1, with exactly one tuple of $\mathbf{R_{d1}}$. Also, I8 requires that the values of the SK_{d1} attributes are reflected in the SH_{u1} attributes of $\mathbf{R_u}$. These facts in combination with the condition $SK_{d1}\rightarrow S_{d1}$, which is implied by the correctness of E2, allow the aggregation of $\mathbf{R_u}$ tuples before the join.

Proof: We start with E1 and by applying a sequence of transformations we get E2. The proof process can be separated into three stages. Each stage concentrates on one of the features of Pre-Grouping. In the first stage we deal with the elimination of the redundant join with $\mathbf{R_{d3}}$ using the Surrogate-Join, Simplify-GroupBy and Remove-Join transformations. In the second stage we push down the aggregation operation, before the join with $\mathbf{R_{d2}}$ using mainly the Surrogate-Join-Early-GroupBy transformation. In the third and final stage we split the aggregation operation into two phases, one before the join with $\mathbf{R_{d1}}$ and the second after this join. For this stage we need the Surrogate-Join, Split-Π and Simplify-GroupBy transformations. The description of the steps in each stage follows:

Stage 1: Remove the redundant join with the $\mathbf{R_{d3}}$.

In order to remove $\mathbf{R_{d3}}$ we need to replace the attributes S_{d3}. This requires the introduction of SK_{d3} for which we know that $S_{d3} \leftrightarrow SK_{d3}$. Conditions I1, I2 and I3 allow us to apply Surrogate-Join and get:

$$\Pi^{S_u, S_{d1}, S_{d2}, S_{d3}}_{S_u, S_{d1}, S_{d2}, A=F(\{Ag^i(M_u^i)\})} \sigma_{SH_{u3}=SK_{d3}} ($$
$$\Pi_{S_u, S_{d1}, S_{d2}, SH_{u3}, \bigcup_i M_u^i} \sigma_{H_{u2}=K_{d2}} (\sigma_{H_{u1}=K_{d1}} (\mathbf{R_u} \times \mathbf{R_{d1}}) \times \mathbf{R_{d2}})$$
$$\times \Pi^{SK_{d3}}_{S_{d3}, SK_{d3}} \mathbf{R_{d3}})$$

For the outer Π operator we have that $SH_{u3}=SK_{d3} \leftrightarrow S_{d3}$. By exploiting this fact we apply Simplify-GroupBy and replace S_{d3} with SH_{u3}. This transformation makes the join with $\mathbf{R_{d3}}$ redundant. Using conditions I2 and I3 we apply Remove-Join and after combining the two consecutive Π operators we finally get:

$$\Pi^{S_u, S_{d1}, S_{d2}, SH_{u3}}_{S_u, S_{d1}, S_{d2}, A=F(\{Ag^i(M_u^i)\})} \sigma_{H_{u2}=K_{d2}} (\sigma_{H_{u1}=K_{d1}} (\mathbf{R_u} \times \mathbf{R_{d1}}) \times \mathbf{R_{d2}})$$

Stage 2: Push-down aggregation below the join with $\mathbf{R_{d2}}$.

The Surrogate-Join-Early-GroupBy is used in order to push the aggregation operation bellow the join with $\mathbf{R_{d2}}$. Conditions I4-I7 satisfy the conditions of the theorem 5.3. By applying the transformation we get:

$$\Pi_{S_u, S_{d1}, S_{d2}, A} \sigma_{SH_{u2}=SK_{d2}} (\Pi^{S_u, S_{d1}, SH_{u2}, SH_{u3}}_{S_u, S_{d1}, SH_{u2}, SH_{u3}, A=F(\{Ag^i(M_u^i)\})}$$
$$\sigma_{H_{u1}=K_{d1}} (\mathbf{R_u} \times \mathbf{R_{d1}}) \times \Pi^{SK_{d2}}_{S_{d2}, SK_{d2}} (\mathbf{R_{d2}}))$$

Stage 3: Split the aggregation into two phases.

In order to maximize the effect of an early aggregation operation on $\mathbf{R_u}$ before the join with $\mathbf{R_{d1}}$, we need to eliminate the H_{u1} attribute used in the join condition. Surrogate-Join is used to perform the modification using conditions I8 and I9. The resulting expression is:

$$\Pi_{S_u, S_{d1}, S_{d2}, A} \sigma_{SH_{u2}=SK_{d2}} ($$
$$\Pi^{S_u, S_{d1}, SH_{u2}, SH_{u3}}_{S_u, S_{d1}, SH_{u2}, SH_{u3}, A=F(\{Ag^i(M_u^i)\})} \sigma_{SH_{u1}=SK_{d1}} ($$
$$\Pi_{S_u, SH_{u1}, SH_{u2}, SH_{u3}, \bigcup_i M_u^i} (\mathbf{R_u}) \times \Pi^{SK_{d1}}_{S_{d1}, SK_{d1}} (\mathbf{R_{d1}})) \times$$
$$\Pi^{SK_{d2}}_{S_{d2}, SK_{d2}} (\mathbf{R_{d2}}))$$

Then we use the Split-Π transformation and split the aggregation process into two phases: before and after the join with $\mathbf{R_{d1}}$. Using I7 we apply Split-Π and get:

$$\Pi_{S_u, S_{d1}, S_{d2}, A} \sigma_{SH_{u2}=SK_{d2}} ($$
$$\Pi^{S_u, S_{d1}, SH_{u2}, SH_{u3}}_{S_u, S_{d1}, SH_{u2}, SH_{u3}, A=F(\{Ag_O^i(m_{II}^i;1)\})} \sigma_{SH_{u1}=SK_{d1}} ($$
$$\Pi^{S_u, SH_{u1}, SH_{u2}, SH_{u3}}_{S_u, SH_{u1}, SH_{u2}, SH_{u3}, \bigcup_i m_{II}^i = Ag_I^i(M_u^i)} \Pi_{S_u, SH_{u1}, SH_{u2}, SH_{u3}, \bigcup_i M_u^i} (\mathbf{R_u}) \times$$
$$\Pi^{SK_{d1}}_{S_{d1}, SK_{d1}} (\mathbf{R_{d1}})) \times \Pi^{SK_{d2}}_{S_{d2}, SK_{d2}} (\mathbf{R_{d2}}))$$

Finally, we combine the two consecutive Π operators and after using the Simplify-GroupBy for the Π operators of $\mathbf{R_{d1}}$ and $\mathbf{R_{d2}}$ we get the final expression E2. □

The conditions of the above theorem appear quite complicated at first sight. Still, most of them can be satisfied by integrity constraints defined in SQL. Conditions I3, I5, and I9 can be satisfied by UNIQUE or PRIMARY_KEY constraints. For example, if K_{d1} is UNIQUE in $\mathbf{R_{d1}}$ then I9 is satisfied. Conditions I2, I4 and I8 can be satisfied by foreign key constraints. For example, if $\{H_{u3}, SH_{u3}\}$ of $\mathbf{R_u}$ are not allowed to have NULL values and they are declared as foreign keys referencing $\{K_{d3}, SK_{d3}\}$ of $\mathbf{R_{d3}}$, then I2 is satisfied.

Conditions I1 and I6 are harder to satisfy. For example, assume that $\mathbf{R_{d2}}$ is not a relation but the view:

$$\Pi_{K_{d2}, SK_{d2}, S_{d2}} \sigma_{XSK_{d2}=SK_{d2}} (\Pi_{K_{d2}, XSK_{d2}} (\mathbf{R_{d21}}) \times \Pi_{SK_{d2}, S_{d2}} (\mathbf{R_{d22}}))$$

Assume also that SK_{d2} is a key of $\mathbf{R_{d22}}$, S_{d2} is defined UNIQUE and not NULL in R_{d22} and XSK_{d2} is defined as a foreign key referencing SK_{d2}. Using the above constraints we get that $SK_{d2} \leftrightarrow S_{d2}$ in $\mathbf{R_{d2}}$, thus satisfying condition I6. Finally, note that it is trivial to satisfy condition I7 when the aggregate functions are limited to the typical SQL aggregate functions.

6. Conclusions

In this paper we started with the analysis of the Hierarchical Pre-Grouping transformation and derived a generalized algebraic form using the expressions E1 and E2. This general form of Pre-Grouping was then decomposed into a sequence of elementary transformations proving that the set of conditions I1 – I9 are sufficient in order to apply Pre-Grouping. Furthermore, we have shown that by using only functional and inclusion dependencies that can be defined in SQL, we can satisfy the conditions I1-I9 and apply Pre-Grouping.

Based on the presented decomposition we have identified the main 'ingredients' of this complex transformation. One of them is obviously Split-Π, a transformation that represents the various optimization techniques defined in previous publications ([CS94, YL95, GHQ95]) for aggregate queries. This transformation is used to push down or split in stages an aggregation operation.

An additional important 'ingredient' of Pre-Grouping was identified to be the Surrogate-Join transformation. The Surrogate-Join transformation is a Semantic Query Transformation technique that uses a number of integrity constraints to alter equi-join expressions modifying the attributes used in the join condition. It is this join-condition modification that allows the Pre-Grouping transformation to eliminate redundant joins and group on an extended number of attributes before performing a join. The importance of this transformation is emphasized by the experimental results presented in previous publications ([KTS02, PER03]). Furthermore, Surrogate-Join can be applied on its own, as shown in the example of section 5, generating alternative plans that may be chosen for execution.

Our results emphasize the belief that Semantic Query Optimization techniques are particularly appropriate in the OLAP area where a large number of dependencies are present. Today's commercial systems make little usage of SQO techniques, missing the opportunity to optimize an important class of queries ([CGK99]).

As future work, we plan to investigate the application of Pre-Grouping and Surrogate-Join to a larger query class including nested queries and attempt to identify useful transformations that combine the Л operator with outerjoin and semijoin operators.

7. Acknowledgments

This work has been partially funded by the European Union's Information Society Technologies Programme (IST) under project EDITH (IST-1999-20722). The authors would like to thank Nikos Karayannidis, Roland Pieringer, Klaus Elhardt, Frank Ramsak and Dimitri Theodoratos for the many fruitful discussions on the Hierarchical Pre-Grouping transformation.

8. Bibliography

[Alb91] Joseph Albert: Algebraic Properties of Bag Data Types. VLDB 1991: 211-219

[CD97] S. Chaudhuri, U. Dayal: An Overview of Data Warehousing and OLAP Technology. SIGMOD Record 26(1): 65-74 (1997)

[CGK99] Qi Cheng, Jarek Gryz, Fred Koo, T. Y. Cliff Leung, Linqi Liu, Xiaoyan Qian, K. Bernhard Schiefer: Implementation of Two Semantic Query Optimization Techniques in DB2 Universal Database. VLDB 1999: 687-698

[CGM90] U. S. Chakravarthy, J. Grant, J. Minker: Logic-Based Approach to Semantic Query Optimization. TODS 15(2): 162-207 (1990)

[CS94] S. Chaudhuri, K. Shim: Including Group-By in Query Optimization. VLDB 1994: 354-366

[CV93] Surajit Chaudhuri, Moshe Y. Vardi: Optimization of Real Conjunctive Queries. PODS 1993: 59-70

[Day87] Umeshwar Dayal: Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Subqueries, Aggregates, and Quantifiers. VLDB 1987: 197-208

[DGK82] Umeshwar Dayal, Nathan Goodman, Randy H. Katz: An Extended Relational Algebra with Control over Duplicate Elimination. PODS 1982: 117-123

[GHQ95] Ashish Gupta, Venky Harinarayan, Dallan Quass: Aggregate-Query Processing in Data Warehousing Environments. VLDB 1995: 358-369

[IR95] Yannis E. Ioannidis, Raghu Ramakrishnan: Containment of Conjunctive Queries: Beyond Relations as Sets. TODS 20(3): 288-324 (1995)

[Kim82] Won Kim: On Optimizing an SQL-like Nested Query. TODS 7(3): 443-469 (1982)

[King81] Jonathan J. King: QUIST: A System for Semantic Query Optimization in Relational Databases. VLDB 1981: 510-517

[KS01] N. Karayannidis, T. Sellis: SISYPHUS: A Chunk-Based Storage Manager for OLAP Cubes. DMDW 2001

[KTS02] Nikos Karayannidis, Aris Tsois, Timos K. Sellis, Roland Pieringer, Volker Markl, Frank Ramsak, Robert Fenk, Klaus Elhardt, Rudolf Bayer: Processing Star Queries on Hierarchically-Clustered Fact Tables. VLDB 2002: 730-741

[MPR90] Inderpal Singh Mumick, Hamid Pirahesh, Raghu Ramakrishnan: The Magic of Duplicates and Aggregates. VLDB 1990: 264-277

[MRB99] Volker Markl, Frank Ramsak, Rudolf Bayer: Improving OLAP Performance by Multidimensional Hierarchical Clustering. IDEAS 1999: 165-177

[NPS91] Mauro Negri, Giuseppe Pelagatti, Licia Sbattella: Formal Semantics of SQL Queries. TODS 16(3): 513-534 (1991)

[PER03] Roland Pieringer, Klaus Elhardt, Frank Ramsak, Volker Markl, Robert Fenk, Rudolf Bayer, Nikos Karayannidis, Aris Tsois, Timos Sellis: Combining Hierarchy Encoding and Pre-Grouping: Intelligent Grouping in Star Join Processing. ICDE 2003

[SO87] Sreekumar T. Shenoy, Z. Meral Özsoyoglu: A System for Semantic Query Optimization. SIGMOD Conference 1987: 181-195

[SY94] Wei Sun, Clement T. Yu: Semantic Query Optimization for Tree and Chain Queries. TKDE 6(1): 136-151 (1994)

[TKS02] Aris Tsois, Nikos Karayannidis, Timos K. Sellis, Dimitri Theodoratos: Cost-based optimization of aggregation star queries on hierarchically clustered data warehouses. DMDW 2002: 62-71

[TS03] Aris Tsois, Timos Sellis: The Generalized Pre-Grouping Transformation: Aggregate-Query Optimization in the Presence of Dependencies. Technical Report, KDBS Lab, NTUA, TR-2003-4 [http://www.dblab.ece.ntua.gr/]

[TT01] Dimitris Theodoratos, Aris Tsois: Heuristic Optimization of OLAP Queries in Multidimensionally Hierarchically Clustered Databases. DOLAP 2001.

[YL94] W. P. Yan, P-Å. Larson: Performing Group-By before Join. ICDE 1994: 89-100

[YL95] Weipeng P. Yan, Per-Åke Larson: Eager Aggregation and Lazy Aggregation. VLDB 1995: 345-357