Reasoning with Aggregation Constraints

Alon Y. Levy[1] and **Inderpal Singh Mumick**[2]

[1] AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA.
levy@research.att.com

[2] AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA.
mumick@research.att.com

Abstract. Aggregation queries are becoming increasingly common as databases continue to grow and provide parallel execution engines to enable complex queries over larger and larger amounts of data. Consequently, optimization of aggregation queries is becoming very important. In this paper we present a framework for reasoning with constraints arising from the use of aggregations. The framework introduces a constraint language, three types of inference rules to derive constraints that must hold given a set of aggregations and constraints in the query, and a sound and tractable inference procedure. The constraint language and inference procedure can be used by any system that deals with aggregations – be it constraint programming, databases, or global information systems. However, the prime application of aggregation reasoning is in database query optimizers to optimize SQL (or object-SQL) queries with grouping and aggregation. Our framework allows aggregation reasoning to be incorporated into an optimizer in a modular fashion, and we illustrate this through a detailed example.

1 Introduction

In advanced database applications (such as decision-support systems) we are witnessing a growing number of very complex queries. The complexity of these queries arises from the fact that they depend on many subqueries and views, each forming a query block in the query graph. The difficulty in optimizing such queries arises from the fact that the query blocks cannot always be merged (due to semantics of duplicates and aggregation), and therefore we cannot apply traditional cost-based plan optimizers which can only handle one query block at a time. In particular, query optimizers are especially ineffective in dealing with queries involving aggregation. At the same time, there is a realization by several parallel database vendors (e.g., Teradata) that optimization of aggregation queries is critical for their systems to scale to larger applications.

An important method of optimization is to rewrite the query so that predicates can be applied as early as possible. Predicate pushdown [Ull89] is a common and important optimization technique for pushing predicates down a query graph, into query blocks that are computed earlier during evaluation. Recently we described the *predicate move-around* algorithm [LMS94] that generalizes predicate pushdown. The key idea in predicate move-around is that the step of pushing predicates *down* the query graph should be preceded by a step in which predicates are pulled *up* the query graph. As a result, predicates that appear in one subtree of the graph can be applied in another subtree, if they are relevant.

A key requirement for such query rewrite algorithms is the ability to infer predicates on the attributes of a view from predicates on the attributes of the relations defining the view (for the pullup phase), or the other way around (for the pushdown phase). While making such inferences for views not involving aggregation is a well understood problem (e.g., see [Ull89]), the problem of making such inferences in the presence of aggregation is a largely open problem that has been considered only in simple cases [LMS94, RSSS94]. As a result, these techniques are unable to push predicates effectively in queries involving aggregation.

In this paper we present a general and principled approach to inferring predicates when views contain aggregation. Specifically, we make the following contributions:

- We describe a *constraint language* in which we can reason with constraints involving aggregation.
- We identify three different types of inferences that need to be made with aggregation constraints in order to use them for query optimization, and show how these inferences can be used to naturally extend query optimization algorithms.
- Finally, we describe algorithms for performing the three kinds of inferences.

Reasoning with aggregation constraints is important not only in query optimization, but also in logic programming, constraint programming, constraint databases, and global information systems [LSK95]. For example, in global information systems, the techniques for pushing constraints down a query graph are used in order to determine which of the many available external databases is *relevant* to a given query. Specifically, if the predicates on a relation R in the query are mutually exclusive with the integrity constraints describing an external source for R, then that source can be deemed irrelevant. For example, if we have a flight database, with the integrity constraint that the minimum flight cost is \$50, and the query asks how to get from NYC to Washington D.C. for less than \$40, then we can deem the flight database irrelevant.

Therefore, our first contribution, the constraint language, provides a basis for investigating the use of aggregation constraints in those domains as well. The problem of reasoning with aggregation constraints is a very broad one. Our second contribution is important because it identifies exactly the subset of the reasoning tasks that need to be addressed in order to use aggregation constraints in query optimization. These tasks are also important for the other areas mentioned above.

We begin with an example that illustrates the issues that arise in reasoning with aggregation constraints. Section 3 describes the constraint language we use and the different types of inferences which together form the framework within which reasoning about aggregation is done. Section 4 gives a detailed account of each type of inference rules. Section 5 shows how the reasoning framework and rules developed in Sections 3 and 4 can be used for query optimization. Related work is discussed in Section 6, and we conclude with Section 7.

2 Illustrative Example

Consider an example involving the relations described below. Phone numbers are broken into area code (AC), and the number (which includes the last 7 digits). The relation `customers` includes a tuple for each customer, specifying the area code, phone, name and membership level (regular, silver, or gold). The relation `calls` stores the calls placed on a telephone network over the last one year, including the From number (the source), the To number (destination), their length, and the date of the call.

> `calls`(FromAC, FromTel, ToAC, ToTel, Date, Length)
> `customers`(AC, Tel, OwnerName, MemLevel)

A marketing query Q is constructed from two views `ptCustomers` (potential customers for marketing plans) and `wellCalled` as follows. The view `ptCustomers` considers only the customers with membership level "silver", and for each, it computes the maximum length call placed to every area code and the earliest date on which a call is placed to the area code (using a `MIN` aggregation function defined over dates.) The view `wellCalled` computes for every area code the maximum length call placed to that area code amongst all the calls in the `calls` relation. The query Q tries to find the customers who have been making long calls to area codes, where the longest incoming calls (from anyone) have been relatively short. The query further wants these customers to have started calling into this area code a long time ago, so as to do a targeted mailing to long standing callers. The query thus chooses the tuples from the view `ptCustomers` for which:

- The maximum length call placed by the user to the area code is greater than 10 minutes (i.e., MaxLen > 10), and
- The maximum length call placed to the area code amongst all the calls in the `calls` relation is *less* than 100 minutes (i.e., MaxLen < 100), and
- The earliest call placed by the user to the area code was made before April 1, 1994 (i.e., MinDate $<$ 1_Apr_1994),

(F):
```
CREATE VIEW ptCustomers (AC, Tel, ToAC, MaxLen,MinDate) AS
    SELECT c.AC, c.Tel, t.ToAC, MAX(t.Length), MIN(t.Date)
    FROM customers c, calls t
    WHERE c.AC = t.FromAC AND
          c.Tel = t.FromTel AND
          c.MemLevel = "Silver"
    GROUPBY c.AC, c.Tel, t.ToAC .
```

```
(E):  CREATE VIEW wellCalled (ToAC, MaxLen) AS
            SELECT t.ToAC, MAX(t.Length)
            FROM calls t
            GROUPBY t.ToAC .

(Q):  SELECT p.AC, p.Tel, p.ToAC, p.MaxLen, p.MinDate,
      FROM wellCalled w, ptCustomers p
      WHERE w.ToAC = p.ToAC AND
            w.MaxLen < 100 AND
            p.MaxLen > 10 AND
            p.MinDate < 1_Apr_1994 .
```

A careful examination of the query reveals that it can be optimized by

- Applying the predicate "t.Length > 10" early on in the computation of the view **wellCalled**, prior to the aggregation step, so that we may write an optimized view:

```
(Eo):  CREATE VIEW wellCalled ( ToAC, MaxLen) AS
             SELECT t.ToAC, MAX(t.Length)
             FROM calls t
             WHERE t.Length > 10                    ⇐ Inferred Predicate
             GROUPBY t.ToAC .
```

- Applying the predicate "(t.Length > 10 OR t.Date < 1_Apr_1994)" early on in the computation of the view ptCustomers, prior to the join and aggregation step, so that we may write an optimized view:

```
(Fo):  CREATE VIEW ptCustomers (AC, Tel, ToAC, MaxLen,MinDate) AS
         SELECT c.AC, c.Tel, t.ToAC, MAX(t.Length), MIN(t.Date)
         FROM customers c, calls t
         WHERE c.AC = t.FromAC AND
               c.Tel = t.FromTel AND
               c.MemLevel = "Silver" AND
               (t.Length > 10 OR t.Date < 1_Apr_1994)  ⇐ Inferred Predicate
         GROUPBY c.AC, c.Tel, t.ToAC .
```

Empirical observation on several gigabytes of telephone call data has shown that 95% of the calls are under 10 minutes, and since the **calls** table only stores data for one year, it is likely that less than 10% of the stored calls are made prior to April 1, 1994. Thus, both the inferred predicates are very selective. The optimized query is thus expected to run an order of magnitude faster than the initial query.

Informally, the inference of the two predicates follows from the following observations. In the query, it will always be the case that w.MaxLen $\geq$ p.MaxLen, because w.MaxLen is the maximum length call for *all* calls, while p.MaxLen is the maximum length call placed by a single customer. Since the query requires that p.MaxLen > 10, it follows that only tuples of **wellCalled** for which w.MaxLen > 10 holds will be relevant to the query. Such tuples of **wellCalled** will be computed correctly if the predicate "`t`.Length $>$ 10" is applied to `calls` before the aggregation operation. Similarly, since the query requires that (`p`.MaxLen $>$ 10 **AND** `p`.MinDate $<$ 1_Apr_1994), it follows that all the relevant tuples of `ptCustomers` can be computed even by applying the predicate "(`t`.Length $>$ 10 OR `t`.Date $<$ 1_Apr_1994)" before the aggregation step. This is because a tuple is relevant if the length is greater than 10 (therefore establishing that the customer makes long enough calls) *or* if the date is before April 1st, 1994 (establishing that the customer started calling early enough).

The optimized query uses the optimized views defined above; however the query statement itself is the same as before. Note that even though we have pushed predicates into the definitions of the views, in this example they still need to be applied in the query block (there are cases (see [LMS94]) in which applying the predicates earlier in the evaluation guarantees that they do not need to be applied later on).

Let us take a closer look at the kinds of inferences that we need to make to automate the above informal reasoning process in a principled fashion:

- First, we need to infer that the maximum length computed from the join of `calls` and `customers` is less than the maximum length computed from `calls`. This is because the maximum in the join is taken over a *subset* of the `calls` relation resulting from the join with the relation `customers`.
- Next, we need to infer that in the query, the maximum length from the view `wellCalled` is greater than the maximum from the view `ptCustomers`, i.e., `w`.MaxLen $\geq$ `p`.MaxLen. This is because `w`.MaxLen was obtained by grouping on a *subset* of the columns on which `p`.MaxLen was computed. (In particular, `wellCalled` grouped each area code, while the `ptCustomers` relation grouped each area code *and* customer).
- Finally, we need to infer that
  - If only tuples that satisfy `w`.MaxLen $>$ 10 are relevant to the query, then the predicate `t`.Length $>$ 10 can be applied on the relation `calls` in the computation of the view `wellCalled`.
  - If only tuples that satisfy (`p`.MaxLen $>$ 10 **AND** `p`.MinDate $<$ 1_Apr_1994) are relevant to the query, then the predicate ("`t`.Length $>$ 10" OR `t`.Date $<$ 1_Apr_1994) can be applied on the relation `calls` in the computation of the view `ptCustomers`. The conversion of a conjunction into a disjunction is counter-intuitive at first; however we must allow for the case where the two maximum length and the minimum date values that satisfy each predicate come from different tuples in the call relation.

In the first kind of inference, which we call *relation-to-view inference*, we infer relationships between the aggregates computed from a view and aggregates computed from the relations defining the view. In the second kind of inference, which we call *intra-relation inference*, we infer relationships between *different* groupby lists and aggregate functions applied to the *same* relation (or view). Finally, in the third kind of inference, which we call *view-to-relation inference,* we infer predicates that can be applied to the relations defining a view from predicates that will *ultimately* be applied to the view itself. These three kinds of inference fit naturally into query optimizers, such as the Starburst optimizer [PHH92] or the predicate move-around algorithm [LMS94]. Relation-to-view inferences are made in the predicate pullup phase of predicate move-around, and view-to-relation inferences are made in the pushdown phase. The intra-relation inferences are made in both phases, while taking the deductive closure of the predicates in a node of the query graph. We will show the actual inference steps in Section 5, after we define each of these inferences formally and then describe how to automate them.

3 Framework for Reasoning with Aggregation

As illustrated in the example above, using aggregation constraints for query optimization requires several different types of inferences to be made about them. In this section we explain formally these types of inferences. In the next section we explain how to perform each one.

As a basis for reasoning with aggregation constraints we need to define a *constraint language* in which we represent aggregation constraints and make inferences about them. Our language extends constraint languages used to reason about constraints that do not involve aggregation. In that case (e.g., as described in [Ull89, LS92, LMS94]), our constraints were of the form:

$$(\forall\ t \in R)\ t.A_1\ \theta\ t.A_2,$$

where R is some relation with attributes A_1 and A_2, t is a tuple variable quantified over all the tuples in R, and θ is one of the operators $\{\leq, <, =, \neq\}$. The meaning of such a constraint is that for every tuple in R the value of the attribute A_1 and the attribute A_2 satisfy the relation θ. We also have simpler constraints of the form $(\forall\ t \in R) t.A_1\ \theta\ c$, where c is a constant.

To extend the language, we note that aggregation constraints essentially introduce new functions that can be applied to attributes in a relation. For example, grouping the relation R by the attribute A_1 and computing the maximum of the values of the attribute A_2 essentially defines a unary function from a value x of attribute A_1 to the maximum value of attribute A_2 amongst all tuples with $A_1 = x$. In our language, we will denote this function by the symbol $f_{\{R,A_1,Max,A_2\}}$. Clearly, there are close relationships between the different functions we can define. For example, for any value x, $f_{\{R,A_1,Max,A_2\}}(x) \geq f_{\{R,A_1,Min,A_2\}}(x)$, that is, for any column, the maximum of the column will always be greater or equal to the minimum. The key behind our algorithm will consist of methods for inferring such relationships.

Formally, a function symbol in our language has the form: $f_{\{R,\bar{X},A,Y\}}$, where:

- R is a relation name (base or view relation).
- $\bar{X}$ is a subset of the attributes of the relation R, denoting the grouping attributes. $\bar{X}$ is called the grouping list.

- A is an aggregation function that is applied to the attribute Y of relation R. For the purpose of this paper, A will be drawn from the following set of five aggregation functions – $\{\texttt{MIN}, \texttt{MAX}, \texttt{SUM}, \texttt{AVG}, \texttt{COUNT}\}$.

The arity of such a function is the number of attributes in $\bar{X}$. The term $f_{\{R,\bar{X},A,Y\}}(x_1,\ldots,x_n)$ denotes the result of grouping the relation R on the columns of $\bar{X}$ that have values $x_1,\ldots,x_n$, and applying the aggregate A to the column Y.

Atomic constraints in our constraint language are of the form

$$(\forall\ t \in R)(\forall\ \bar{x})\ \ \tau_1\ \theta\ \tau_2,$$

where τ_1 and τ_2 are either

- constants,
- attributes of the tuple t (i.e., of the form $t.A$, where A is an attribute of relation R), or
- functional terms of the form $f_{\{R,\bar{X},A,Y\}}(\alpha_1,\ldots,\alpha_n)$, where the α_i's are either constants or attributes of the tuple t.

The set of variables appearing in τ_1 and τ_2 are $\bar{x}$. The meaning of such a formula is that for every tuple of R and values $\bar{a}$ for the variables $\bar{x}$, the formula $\tau_1[\bar{x}/\bar{a}]\theta\tau_2[\bar{x}/\bar{a}]$ holds, where $\tau[\bar{x}/\bar{a}]$ denotes the constraint resulting from replacing the variables $\bar{x}$ by $\bar{a}$ respectively. Note that an atomic constraint does not necessarily have to have both a tuple quantifier (i.e., $\forall\ t \in R$) and variable quantifiers (i.e., $(\forall\ \bar{x})$).

Definition 1. (Atomic and Functional Terms) A term that contains a function symbol, such as $f_{\{R,\bar{X},A,Y\}}$ is called a *functional term*; otherwise the term is a constant or an attribute of a tuple, and is called an *atomic term*.

Definition 2. (Ordinary and Aggregation Predicates) An atomic constraint in which both the terms τ_1 and τ_2 are atomic is called an *ordinary predicate*; otherwise the constraint involves at least one functional term and is called an *aggregation predicate*.

In this paper we consider constraints that are conjunctions of atomic constraints. An atomic constraint will be denoted by c_i, and the conjunction of atomic constraints would be denoted by ϕ. The language can also be generalized to more complex constraints (such as linear equations involving the terms). We say that a constraint is a *single-relation* constraint of relation R if its functional terms use only functions over R, and its tuple quantifier ranges over the tuples of R (if such a quantifier exists).

In order to use aggregation constraints for query optimization, we need to perform three kinds of inferences with expressions in our constraint language:

1. **Intra-relation inference:** This type of inference deduces predicates between functional terms of the *same* relation. For example, we can infer

$$(\forall\ x,y)\ f_{\{R,[A],Max,G\}}(x) \geq f_{\{R,[A,B],Max,G\}}(x,y)$$

since the maximum value of attribute G amongst all tuples with the same value for attribute A is guaranteed to be greater than the maximum value of attribute G amongst all tuples with the same value for attributes A and B.
As another example, from $(\forall t \in R)\,(R.B > R.C)$ we can infer that

$$(\forall x)\, f_{\{R,[A],Max,B\}}(x) > f_{\{R,[A],Max,C\}}(x).$$

Formally, we are given single relation constraints ϕ_1 and ϕ_2 of relation R, and the problem is to decide whether ϕ_1 is valid (i.e., whether $\models \phi_1$) or to decide whether $\phi_1 \models \phi_2$.

2. **Relation-to-view inference:** This type of inference deduces predicates between attributes and functional terms of a view and those of relations defining the view. For example, if V is defined by

   CREATE VIEW V AS
   SELECT $E_1.A, E_1.B$
   FROM E_1, E_2
   WHERE $E_1.A = E_2.C$

   we can infer that $\forall x\, f_{\{E_1,A,Max,B\}}(x) \geq f_{\{V,A,Max,B\}}(x)$. This holds because the tuples of the view are a subset of the tuples in E_1, and therefore, the maximum values from V cannot be greater than the maximum values from E_1.
   Formally, suppose the view V is defined using the relations $E_1, \ldots, E_m$. Let c be an atomic constraint involving the relation V and at least one of the relations $E_1, \ldots, E_m$. Let $\phi_1, \ldots, \phi_m$ be conjunctive constraints on the relations $E_1, \ldots, E_m$, respectively (i.e., every tuple in E_i must satisfy ϕ_i). The relation-to-view inference problem is to decide whether

$$\phi_1 \wedge \ldots \wedge \phi_m \models c.$$

3. **View-to-relation inference:** In this type of inference, we begin with a constraint c involving only the view V. This constraint represents the tuples of V that are relevant to the query (i.e., tuples of V not satisfying constraint c can be ignored without changing the result of the query). We use the constraint c to infer which tuples of the relations $E_1, \ldots, E_m$ (from which V is defined) are needed in order to compute the necessary part of V. For example, suppose V is defined from the view V_1 as follows:

   CREATE VIEW $V(A, C)$ AS
   SELECT $V_1.A, Max(V_1.B)$
   FROM V_1
   GROUPBY $V_1.A$

   Suppose that we have determined that only tuples of V that satisfy $5 \leq C \leq 10$ are relevant to the query. We can infer that we only need to compute tuples of V_1 that satisfy $B \geq 5$.

Formally, the view V is defined using the relations $E_1, \ldots, E_m$. We are given a single-relation constraint ϕ_V on the view V, and a single relation constraint ϕ_{E_i} on one of the relations E_i defining V. Let $V(G_1, \ldots, G_m)$ represent the result of computing V by substituting the expressions $G_1, \ldots, G_m$ for the relations $E_1, \ldots, E_m$. The view-to-relation inference problem is to decide whether the following equivalence holds:

$$\sigma_{\phi_V} V(E_1, \ldots, E_m) \equiv \sigma_{\phi_V} V(E_1, \ldots, \sigma_{\phi_{E_i}}(E_i), \ldots, E_m).$$

An optimization engine may restrict this inference problem to the case where ϕ_{E_i} is an ordinary predicate (does not involve any functional terms). The reason for this restriction is that if the constraints involve functional terms, then evaluating them may require performing additional aggregation operations. Such a predicate would then be classified as an expensive predicate, and it may be better not to push it down from a view into the referenced relations [HS93, Hel94].

4 Making Inferences with Aggregation Constraints

Recall that the inference problem in our constraint language is to decide whether a constraint c is entailed by a conjunction (possibly empty) of constraints $c_1 \wedge \ldots \wedge c_n$. To do so, we apply a general strategy, in which we begin with a set of constraints ϕ, initially containing $c_1, \ldots, c_n$, and apply a set of *inference rules* to add new constraints to ϕ. We apply the inference rules until no new constraints can be added to ϕ, and then, if $c \in \phi$, we will say that $c_1 \wedge \ldots \wedge c_n \models c$. To illustrate, consider the following example inference rules. We use the letters p and q to denote relations in the inference rules.

r_1 : **from** $(x \leq y)$ and $(y \leq z)$ **infer** $x \leq z$.
r_2 : **from** $(\forall\, t \in p)(t.B \leq t.C)$ **infer** $(\forall\, x) f_{\{p,[\bar{X}],Min,B\}}(x) \leq f_{\{p,[\bar{X}],Min,C\}}(x)$
r_3 : $(\forall x) f_{\{p,[\bar{X}],Min,B\}}(x) \leq f_{\{p,[\bar{X}],Max,B\}}(x)$

Suppose $R(A, B, C)$ is a relation, and we are given a constraint $c_1 : (\forall t \in R)(t.B \leq t.C)$. Using rule r_2, we can infer

$c_2 : (\forall\, x) f_{\{R,A,Min,B\}}(x) \leq f_{\{R,A,Min,C\}}(x)$

Using rule r_3 we can infer

$c_3 : (\forall x) f_{\{R,A,Min,C\}}(x) \leq f_{\{R,A,Max,C\}}(x)$

Finally, using r_1, and the inferred constraints c_2 and c_3, we can infer

$c_4 : (\forall x) f_{\{R,A,Min,B\}}(x) \leq f_{\{R,A,Max,C\}}(x)$

Therefore, we conclude that $c_1 \models c_4$.

In general, an inference rule has a set of *premises* and a set of *conclusions*. If we can find an instantiation of the premises of an inference rule in ϕ, then we can infer the corresponding instantiation of its conclusion and add it to ϕ.

Some inference rules, such as r_3, do not have any premises, and therefore their instantiations can always be added to ϕ. Such inference rules can be viewed as axiom schemas.

The worst case complexity of the inference procedure is $O(|r| \times |c|)$, where $|r|$ is the number of inference rules and $|c|$ is the number of different possible atomic constraints. The number $|c|$ is polynomial in the number of relations and is exponential in the arity of the relations (which tend to be small). In Section 5 we describe how to further speedup the inference process by focusing the application of the inference rules only to *needed* functional terms. The needed functional terms are determined by examining the structure of the query. In the following subsections we focus on the inference rules used and the principles underlying them. The following theorem holds for all the rules presented.

Theorem 3. *The inference procedure using the rules and axioms in Section 4 is sound, i.e., every predicate that is inferred can be applied without changing the result of the query. The inference can be done in time polynomial in the number of inference rules and functional terms.*

4.1 Relation-to-View Inferences

Recall that relation-to-view inferences are meant to infer relationships between terms involving a view V and terms involving the relations defining V, which we denote by $E_1, \ldots, E_n$. We assume that a view V can be defined from $E_1, \ldots, E_n$ using a combination of selection, projection, cross product, union, grouping and aggregation operations. In order to better understand the inference rules, we present rules that involve a single such operation at a time. Rules for more complex view definitions are obtained by composing these rules. We only present representative rules here. A more comprehensive list of rules appears in the full paper.

Projection: Let the view $V(\bar{X}_1)$ be defined by projecting arguments from the relation $E(\bar{X}_2)$, i.e., $\bar{X}_1 \subset \bar{X}_2$. For each set $\bar{X} \subset \bar{X}_1$, for each attribute $Y \in \bar{X}_1$, and for each aggregation function $A \in \{\texttt{MIN}, \texttt{MAX}\}$ we have the following rule:

$(L1)$: $(\forall \bar{x}) f_{\{V,\bar{X},A,Y\}}(\bar{x}) = f_{\{E,\bar{X},A,Y\}}(\bar{x})$.

That means that the minimum and maximum of a column do not change as a result of projection. If duplicates are preserved during projection, then the same holds for Count and Sum (hence for Avg). If duplicates are not preserved, then, for `COUNT` we have the following rule:

$(L2)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{COUNT},Y\}}(\bar{x}) \leq f_{\{E,\bar{X},\texttt{COUNT},Y\}}(\bar{x})$.

Selection: Suppose the view $V(\bar{X}_1)$ is defined by applying the selection c on the relation $E(\bar{X}_1)$. Then the following axioms will hold (for each $\bar{X} \subset \bar{X}_1, Y \in \bar{X}_1$):

$(L3)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{MIN},Y\}}(\bar{x}) \geq f_{\{E,\bar{X},\texttt{MIN},Y\}}(\bar{x})$.
$(L4)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{MAX},Y\}}(\bar{x}) \leq f_{\{E,\bar{X},\texttt{MAX},Y\}}(\bar{x})$.

$(L5)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{COUNT},Y\}}(\bar{x}) \leq f_{\{E,\bar{X},\texttt{COUNT},Y\}}(\bar{x})$.

For example, the first rule says that the minimum can only increase as a result of selection. Note that we cannot say anything about the average of V. More inference rules can be stated for the cases where the selection predicate c is of a specific form (such as $Y \leq k$, where k is a constant). For example, when $Y \leq k$, the `MIN` values in the view V is equal to the `MIN` value from E.

Cross Product: Suppose $V(\bar{X}_1, \bar{X}_2)$ is defined by the cross product of $E_1(\bar{X}_1)$ and $E_2(\bar{X}_2)$, where $\bar{X}_1$ and $\bar{X}_2$ are disjoint sets of variables. Then, for each $\bar{X} \subset \bar{X}_1$ and $Y \in \bar{X}_1$, we have the following rules (similar rules hold for E_2):

$(L6)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{MIN},Y\}}(\bar{x}) = f_{\{E_1,\bar{X},\texttt{MIN},Y\}}(\bar{x})$.
$(L7)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{MAX},Y\}}(\bar{x}) = f_{\{E_1,\bar{X},\texttt{MAX},Y\}}(\bar{x})$.
$(L8)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{AVG},Y\}}(\bar{x}) = f_{\{E_1,\bar{X},\texttt{AVG},Y\}}(\bar{x})$.
$(L9)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{COUNT},Y\}}(\bar{x}) = f_{\{E_1,\bar{X},\texttt{COUNT},Y\}}(\bar{x}) \times Size(E_2)$.

where $Size(E_2)$ is the number of tuples in relation E_2. For example, the first rule says that the value of the minimum does not change as a result of cross product.

Union: Suppose $V(\bar{X}_1)$ is defined to be the union of $E_1(\bar{X}_1)$ and $E_2(\bar{X}_1)$. For each $\bar{X} \subset \bar{X}_1$ and $Y \in \bar{X}_1$, the following holds:

$(L10)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{MIN},Y\}}(\bar{x}) \leq f_{\{E_1,\bar{X},\texttt{MIN},Y\}}(\bar{x})$.
$(L11)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{MAX},Y\}}(\bar{x}) \geq f_{\{E_1,\bar{X},\texttt{MAX},Y\}}(\bar{x})$.
$(L12)$: $(\forall \bar{x})(f_{\{V,\bar{X},\texttt{MAX},Y\}}(\bar{x}) = f_{\{E_1,\bar{X},\texttt{MAX},Y\}}(\bar{x})) \vee (f_{\{V,\bar{X},\texttt{MAX},Y\}}(\bar{x}) = f_{\{E_2,\bar{X},\texttt{MAX},Y\}}(\bar{x}))$.
$(L13)$: $(\forall \bar{x})(f_{\{V,\bar{X},\texttt{MIN},Y\}}(\bar{x}) = f_{\{E_1,\bar{X},\texttt{MIN},Y\}}(\bar{x})) \vee (f_{\{V,\bar{X},\texttt{MIN},Y\}}(\bar{x}) = f_{\{E_2,\bar{X},\texttt{MIN},Y\}}(\bar{x}))$.
$(L14)$: $(\forall \bar{x}) f_{\{V,\bar{X},\texttt{COUNT},Y\}}(\bar{x}) \geq f_{\{E_1,\bar{X},\texttt{COUNT},Y\}}(\bar{x})$.

For example, the first rule states that the minimum can only decrease as a result of a union.

Grouping and Aggregation: Suppose the view $V(\bar{X}_1, \bar{Z})$ is defined by grouping on a relation $E(\bar{X}_1, \bar{Y})$, where $\bar{X}_1$ is the grouping list, and $Z_i = A_i(Y_i)$ is the list of attributes obtained by applying aggregation function A_i on the attribute Y_i of relation E. Then, for each $Z_i \in \bar{Z}$, the following holds by definition:

$(L15)$: $(\forall v \in V) v.Z_i = f_{\{E,\bar{X}_1,A_i,Y_i\}}(E.\bar{X})$.

Further, for each $\bar{X} \subseteq \bar{X}_1$, and $W \in \bar{X}_1$, and $A \in \{\texttt{MAX}, \texttt{MIN}\}$:

$(L16)$: $(\forall v \in V) f_{\{V,\bar{X},A,W\}}(v.\bar{x}) = f_{\{E,\bar{X},A,W\}}(v.\bar{x})$.

while for `COUNT` we have:

$(L17)$: $(\forall v \in V) f_{\{V,\bar{X},\texttt{COUNT},W\}}(v.\bar{x}) \leq f_{\{E,\bar{X},\texttt{COUNT},W\}}(v.\bar{x})$.

4.2 Intra-relation Inferences

Intra-relation inferences deduce relationships between functional terms concerning the *same* relation. For a given relation R, we need to consider functional terms that differ on (1) the aggregate function, (2) the groupby list, and (3) the aggregated attribute. In addition, we can deduce special relationships when the aggregated column is a member of the grouping list.

Changing the Aggregation function: Ross et al. [RSSS94] present a complete set of rules for the case that only the aggregation function changes. For example, the following two rules hold:

$(L18)$: $(\forall \bar{x}) f_{\{R,\bar{X},\mathtt{MAX},Y\}}(\bar{x}) \geq f_{\{R,\bar{X},\mathtt{MIN},Y\}}(\bar{x})$.
$(L19)$: $(\forall \bar{x}) f_{\{R,\bar{X},\mathtt{MAX},Y\}}(\bar{x}) \geq f_{\{R,\bar{X},\mathtt{AVG},Y\}}(\bar{x})$.

i.e., the maximum is greater than either the minimum or the average.

Changing the groupby list: We consider the case in which one groupby list is a subset of the other (note that a *groupby list* is actually a set, not a list). Given a relation $R(\bar{X})$, the following rules apply for changing the groupby list. For each $\bar{X}_1 \subset \bar{X}_2 \subset \bar{X}$, $Y \in \bar{X}$.

$(L20)$: $(\forall \bar{x}_2) f_{\{R,\bar{X}_2,\mathtt{MAX},Y\}}(\bar{x}_2) \leq f_{\{R,\bar{X}_1,\mathtt{MAX},Y\}}(\bar{x}_1)$.
$(L21)$: $(\forall \bar{x}_2) f_{\{R,\bar{X}_2,\mathtt{MIN},Y\}}(\bar{x}_2) \geq f_{\{R,\bar{X}_1,\mathtt{MIN},Y\}}(\bar{x}_1)$.
$(L22)$: $(\forall \bar{x}_2) f_{\{R,\bar{X}_2,\mathtt{COUNT},Y\}}(\bar{x}_2) \leq f_{\{R,\bar{X}_1,\mathtt{COUNT},Y\}}(\bar{x}_1)$.

For example, the first rule states that aggregating on a subset of the columns can only increase the maximum.

Changing the Aggregated Attribute: When we change the aggregated attribute, then the count remains the same, as formalized by the following rule:

$(L23)$: $(\forall \bar{x}) f_{\{R,\bar{X},\mathtt{COUNT},Y_1\}}(\bar{x}) = f_{\{R,\bar{X},\mathtt{COUNT},Y_2\}}(\bar{x})$.

All the inference rules we have stated up to now are axiom schemas, i.e., the conclusions can be inferred without any premises. The next few inference rules deduce a relationship between functional terms of different attributes given with premises that are ordinary predicates:

$(L24)$: $(\forall t \in R)(t.Y_1 \leq t.Y_2) \longrightarrow (\forall \bar{x}) f_{\{R,\bar{X},\mathtt{MIN},Y_1\}}(\bar{x}) \leq f_{\{R,\bar{X},\mathtt{MIN},Y_2\}}(\bar{x})$.
$(L25)$: $(\forall t \in R)(t.Y_1 \leq t.Y_2) \longrightarrow (\forall \bar{x}) f_{\{R,\bar{X},\mathtt{MAX},Y_1\}}(\bar{x}) \leq f_{\{R,\bar{X},\mathtt{MAX},Y_2\}}(\bar{x})$.
$(L26)$: $(\forall t \in R)(t.Y_1 \leq t.Y_2) \longrightarrow (\forall \bar{x}) f_{\{R,\bar{X},\mathtt{AVG},Y_1\}}(\bar{x}) \leq f_{\{R,\bar{X},\mathtt{AVG},Y_2\}}(\bar{x})$.

Similar rules apply if $Y_1 < Y_2$, $Y_1 > Y_2$, or $Y_1 \geq Y_2$. Furthermore, if $Y_1 \leq k$, where k is a constant, the above rules can be simplified to:

$(L27)$: $(\forall t \in R)(t.Y_1 \leq k) \longrightarrow (\forall \bar{x}) f_{\{R,\bar{X},\mathtt{MIN},Y_1\}}(\bar{x}) \leq k$.
$(L28)$: $(\forall t \in R)(t.Y_1 \leq k) \longrightarrow (\forall \bar{x}) f_{\{R,\bar{X},\mathtt{MAX},Y_1\}}(\bar{x}) \leq k$.

$(L29)$: $(\forall t \in R)(t.Y_1 \leq k) \longrightarrow (\forall \bar{x}) f_{\{R,\bar{X},\mathtt{AVG},Y_1\}}(\bar{x}) \leq k$.

and similarly for $Y_1 < a$, $Y_1 > k$, and $Y_1 \geq k$.

Aggregating on a grouping column: When the aggregated column is one of the columns in the grouping list, the result of the aggregation function can often be statically determined. Given a relation $R(\bar{X})$, for each $\{X_1, \ldots, X_m\} \subset \bar{X}$, and for each X_i:

$(L30)$: $(\forall \bar{x}) f_{\{R,\{X_1,\ldots,X_m\},\mathtt{MAX},X_i\}}(\bar{x}) = x_i$.
$(L31)$: $(\forall \bar{x}) f_{\{R,\{X_1,\ldots,X_m\},\mathtt{MIN},X_i\}}(\bar{x}) = x_i$.
$(L32)$: $(\forall \bar{x}) f_{\{R,\{X_1,\ldots,X_m\},\mathtt{AVG},X_i\}}(\bar{x}) = x_i$.

For example, the first rule states that the maximum value of an attribute X_i in the grouping list $\{X_1, \ldots, X_m\}$ for the group defined by $\{X_1, \ldots, X_m\} = \{x_1, \ldots, x_m\}$ is x_i.

4.3 View-to-relation Inference

Suppose a view V is defined from the relations $E_1, \ldots, E_m$. As we saw in the example, it is often the case that if we know that some predicates are applicable to the tuples of V without changing the result of the query, then we can infer predicates on $E_1, \ldots, E_n$, that can be *safely* applied in the computation of V (i.e., they can be applied without changing the result of the query). This section describes conditions under which such predicates can be pushed down the query graph.

We assume the view V has the arguments $\bar{X} = X_1, \ldots, X_m$, and $\bar{Z} = Z_1, \ldots, Z_l$, where $\bar{X}$ is the set of grouped variables in the view definition, and $\bar{Z}$ is the set of attributes resulting from aggregation. The attribute Z_i is of the form $A(Y_i)$ where A is some aggregate function (one of `MIN, MAX, AVG, SUM, COUNT`) and Y_i is an attribute of one of the E_i's.

We assume we have a conjunctive predicate $\phi = c_1 \wedge \ldots \wedge c_n$, such that if ϕ is applied to the tuples of V, the result of the query will not change. Each of the c_i's is an ordinary atomic predicate of the form $\tau_1 \theta \tau_2$, where τ_1 and τ_2 are either constants or attributes of V (i.e., one of $\bar{X} \cup \bar{Z}$).

In our description below, we assume that ϕ is closed under logical deduction, except for the following condition:

> ϕ does not contain trivial redundancies, that is, if $\tau \leq k$ and $\tau \leq k_1$ are in the deductive closure of ϕ and $k < k_1$, then ϕ does not include $\tau \leq k_1$ (and the analogous rule for $\geq$).

We illustrate the intuition behind the pushing of predicates with the following example.

Example 1. Suppose the view V contains the attributes X_1, X_2, Z_1, Z_2, where $Z_1 = \mathtt{MAX}(Y_1)$ and $Z_2 = \mathtt{MIN}(Y_2)$, and suppose that we have the predicate $\phi =$

$(X_1 \leq X_2) \wedge (Z_1 \geq 5) \wedge (Z_2 \leq 2)$. The predicate $(X_1 \leq X_2)$ can be pushed down to the computation of V because it applies to *whole* groups resulting from the grouping operation, and therefore it can be applied before the grouping. Furthermore, only tuples for which $(Y_1 \geq 5)$ are relevant to the computation of Z_1, and similarly, only tuples for which $(Y_2 \leq 2)$ are relevant to the computation of Z_2. However, we cannot apply the conjunction of these two predicates, because a tuple with $(Y_2 > 2)$ may actually be the one containing the maximum of Y_1. However, the *disjunction* of these predicates can still be applied, i.e., we can apply the predicate $(X_1 \leq X_2) \wedge (Y_2 \leq 2 \vee Y_1 \geq 5)$. □

Formally, the predicates that can be inferred are computed as follows:

1. Let $\phi_{\bar{X}}$ be the subset of ϕ that does not include variables in $\bar{Z}$.
2. For every Z_i compute a predicate ψ_i as follows:
   - If Z_i is of the form $\mathtt{MAX}(Y_i)$, then let $\bar{d} = d_1, \ldots, d_{m_i}$ be the set of predicates in ϕ of the form $Z_i \geq x$ or $Z_i > x$, where x is a constant or a variable in $\bar{X}$. $\psi_i = d'_1 \wedge \ldots \wedge d'_{m_i}$, where d'_j substitutes Y_i for Z_i in d_j. If $\bar{d}$ is empty, then $\psi_i = True$.
   - If Z_i is of the form $\mathtt{MIN}(Y_i)$, ψ_i is computed analogously.
   - If Z_i involves `AVG`, `SUM`, or `COUNT`, then $\psi_i = True$.
3. The predicate that can be pushed into the view definition is: $\phi_{\bar{X}} \wedge (\psi_1 \vee \ldots \vee \psi_l)$.

In our example above, $\phi_{\bar{X}}$ would be $X_1 \leq X_2$. For Z_1 we would have $\psi_1 = Y_1 \geq 5$, and for Z_2 we would have $\psi_2 = Y_2 \leq 2$. Note that if we would have an additional attribute in the view, $Z_3 = \mathtt{MIN}(Y_3)$, we would get $\phi_3 = True$, and therefore, we would only be able to push the predicate $X_1 \leq X_2$ to the definition of the view.

5 Aggregation Reasoning in Query Optimization

The aggregation constraint language and the inference rules for aggregation can be used to formalize the informal reasoning we outlined in Section 2 for optimizing queries. Algorithmically, the functional terms and the inference rules can be made a part of the constraint language and inference mechanism that an optimizer uses for predicate analysis. To do so in an effective manner, we first need to identify the functional terms *relevant* to a query on which the inference rules need apply, and then need to show how the inference rules can be applied to optimize the query. The inference rules infer ordinary predicates as well as aggregation predicates. After the inference process ends, the derived ordinary predicates can be used for more efficient query evaluation. However, the axiomatic aggregation predicates are not useful for query evaluation. Other aggregation predicates (such as $f_{\{R,\{X\},\mathtt{MAX},Y\}}$) may be useful, but these are typically expensive to evaluate. Thus the optimizer's hope is that the inference process would have used aggregation predicates as intermediate predicates to derive ordinary predicates that can be used in the optimized query to restrict computation.

To describe the use of aggregation reasoning in optimization, we will represent an SQL query using a query graph [MFPR90, PHH92, MP94, LMS94]. A query graph is a pictorial representation of the query. For example, the marketing query Q from Section 2 is represented by the query graph of Figure 1. The graph has

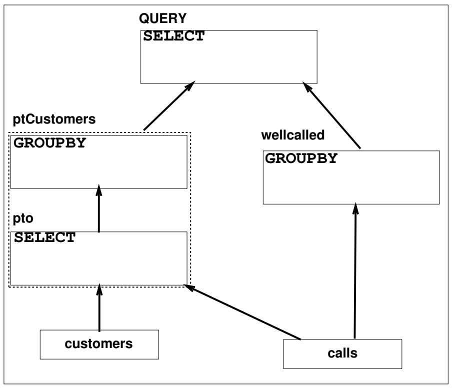

Fig. 1. The query graph for the marketing query of Section 2.

six nodes. The nodes labeled `customers` and `calls` represent base relations; all other nodes represent computed relations. The node labeled `pt0` represents the result of the join between `calls` and `customers` that is grouped upon in the node labeled `ptCustomers` to generate the `ptCustomers` view. The node labeled `wellCalled` represents the grouping operation on the `calls` relation to generate the view `wellCalled`. Finally, the query node, labeled Q, takes a join between the two views to derive the answer.

5.1 Identifying the Functional Terms

The functional symbols that can be used in the aggregation reasoning are of the form:

$$f_{\{R,GL,A,Y\}}$$

where R can be any of the base or view relations labeling a node in the query graph of the query, the grouping list GL is a subset of the attributes of relation R, A is one of the aggregation functions (`MIN`, `MAX`, `SUM`, `AVG`, `COUNT`) applied to a single attribute Y of relation R.

If we were to introduce every one of these function symbols in the constraint language, we would have a large number of functional terms. Therefore, it important to restrict the functional terms we consider. Recall that in the end, we remove all of the predicates involving functional terms, and leave only the ordinary predicates (predicates that do not involve functional terms) that were derived using the functional terms. Therefore, we need only consider functional terms that are useful in deriving ordinary predicates. Below we show how to identify the functional terms that are relevant to deriving ordinary predicates. A functional term involving a relation R is useful only if the query does an aggregation on R or on a view derived from R. The grouping lists and the aggregated attributes can be restricted in a similar fashion:

- Let $\mathcal{R}$ be the set of base and view relation names derived as follows:
  - If R is the name of a base relation or the label of a node in the query graph that feeds into a groupby node, then it is in $\mathcal{R}$.
  - If $R1 \in \mathcal{R}$, $R1$ is the label of a node in the query graph, and $R2$ is the name of a base relation or the label of a node in the query graph that feeds into the node labeled $R1$, then $R2$ is in set $\mathcal{R}$.
- For every relation $R \in \mathcal{R}$, derive a set GL_R of pairs (grouping lists, aggregated column) as follows:
  - If R feeds into a groupby node, and GL is the groupby list used in the groupby node, then for each column Y aggregated in the groupby node, place (GL, Y) into GL_R.
  - If $(GL, Y) \in GL_R$, and relation S feeds into the node labeled R, and relation S provides the attribute Y into R, then include $(\pi_S(GL), Y)$ into GL_S. $\pi_S(GL)$ is the projection of the groupby list GL onto the attributes that appear in relation S.

The only functional terms that are relevant to query optimization are those with a relation name $R \in \mathcal{R}$, and with a grouping list and aggregation column in the set GL_R. Thus, the grouping lists and aggregation columns to be considered by the inference algorithm is restricted to the lists and columns actually used in the query, or derived from the query. In fact the number of resulting functional terms is linear in the size of the query.

Example 2. For the marketing query Q of Section 2:
$\mathcal{R} = \{\texttt{pt0}, \texttt{calls}, \texttt{customers}\}$
$GL_{\texttt{pt0}} = \{(\{\text{AC, Tel, ToAC}\}, \text{Length}), (\{\text{AC, Tel, ToAC}\}, \text{Date})\}$
$GL_{\texttt{calls}} = \{(\{\text{FromAC, FromTel, ToAC}\}, \text{Length}),$
$(\{\text{FromAC, FromTel, ToAC}\}, \text{Date}), (\{\text{ToAC}\}, \text{Length})\}$
$GL_{\texttt{customers}} = \{\}$

The following functional terms, one for each possible aggregation function A, are thus the only ones needed for query optimization:

$f_{\{\texttt{pt0}, \{\text{AC, Tel, ToAC}\}, A, \text{Length}\}}$
$f_{\{\texttt{pt0}, \{\text{AC, Tel, ToAC}\}, A, \text{Date}\}}$
$f_{\{\texttt{calls}, \{\text{FromAC, FromTel, ToAC}\}, A, \text{Length}\}}$
$f_{\{\texttt{calls}, \{\text{FromAC, FromTel, ToAC}\}, A, \text{Date}\}}$
$f_{\{\texttt{calls}, \{\text{ToAC}\}, A, \text{Length}\}}$

5.2 Inferring predicates during optimization

We illustrate the use of the inference rules of Section 4 for optimization through the marketing query Q of Section 2. The functional terms of interest were listed in Example 2.

Using inference rules $L3$, $L4$, $L7$, and $L8$ in Section 4.1, we derive:

$$(c1):\ (\forall X,Y,Z) f_{\{\texttt{pt0},\{AC,Tel,ToAC\},\texttt{MAX},Length\}}(X,Y,Z) \\ \leq f_{\{\texttt{calls},\{FromAC,FromTel,ToAC\},\texttt{MAX},Length\}}(X,Y,Z).$$

$$(c2):\ (\forall X,Y,Z) f_{\{\texttt{pt0},\{AC,Tel,ToAC\},\texttt{MIN},Date\}}(X,Y,Z) \\ \geq f_{\{\texttt{calls},\{FromAC,FromTel,ToAC\},\texttt{MIN},Date\}}(X,Y,Z).$$

Using inference rules $L22$ and $L23$ in Section 4.2, we derive:

$$(c3):\ (\forall X,Y,Z) f_{\{\texttt{calls},\{ToAC\},\texttt{MAX},Length\}}(Z) \\ \geq f_{\{\texttt{calls},\{FromAC,FromTel,ToAC\},\texttt{MAX},Length\}}(X,Y,Z).$$

$$(c4):\ (\forall X,Y,Z) f_{\{\texttt{calls},\{ToAC\},\texttt{MIN},Date\}}(Z) \\ \leq f_{\{\texttt{calls},\{FromAC,FromTel,ToAC\},\texttt{MIN},Date\}}(X,Y,Z).$$

Using inference rule $L17$ in Section 4.1, we derive:

$$(c5):\ (\forall \texttt{p} \in \texttt{ptCustomers}) \\ (\texttt{p}.MaxLen = f_{\{\texttt{pt0},\{AC,Tel,ToAC\},\texttt{MAX},Length\}}(\texttt{p}.AC, \texttt{p}.Tel, \texttt{p}.ToAC)).$$

$$(c6):\ (\forall \texttt{p} \in \texttt{ptCustomers}) \\ (\texttt{p}.MinDate = f_{\{\texttt{pt0},\{AC,Tel,ToAC\},\texttt{MIN},Date\}}(\texttt{p}.AC, \texttt{p}.Tel, \texttt{p}.ToAC)).$$

$$(c7):\ (\forall \texttt{w} \in \texttt{wellcalled})(\texttt{w}.MaxLen = f_{\{\texttt{calls},\{ToAC\},\texttt{MAX},Length\}}(\texttt{w}.ToAC)).$$

From inferences $c1$ and $c3$, it follows that

$$(c8):\ (\forall X,Y,Z)(f_{\{\texttt{calls},\{ToAC\},\texttt{MAX},Length\}}(Z) \\ \geq f_{\{\texttt{pt0},\{AC,Tel,ToAC\},\texttt{MAX},Length\}}(X,Y,Z).$$

From inferences $c5$, $c7$, and $c8$ it follows that

$$(c9):\ (\forall \texttt{p} \in \texttt{ptCustomers}, \texttt{w} \in \texttt{wellcalled}) \\ ((\texttt{w}.ToAC = \texttt{p}.ToAC) \Rightarrow (\texttt{w}.MaxLen \geq \texttt{p}.MaxLen)).$$

Since it is known that $\texttt{w}.ToAC = \texttt{p}.ToAC$, the predicate $\texttt{w}.MaxLen \geq \texttt{p}.MaxLen$ can be inferred in the query. This then derives the predicate

$$(c10):\ (\forall \texttt{w} \in \texttt{wellcalled})\ \texttt{w}.MaxLen > 10.$$

We are also given the predicates

$(c11)$: $(\forall \mathtt{p} \in \mathtt{ptCustomers})\ \mathtt{p}.MaxLen > 10$.
$(c12)$: $(\forall \mathtt{p} \in \mathtt{ptCustomers})\ \mathtt{p}.MinDate < 1\_Apr\_1994$.

Now by applying the pushdown inference algorithm in Section 4.3, to the predicates $c10 - c12$ above, we derive the predicate

$(c13)$: $(\forall \mathtt{t} \in \mathtt{calls})\ \mathtt{t}.Length > 10$.

in the view `wellcalled`, and the predicate

$(c14)$: $(\forall \mathtt{p0} \in \mathtt{pt0})\ (\mathtt{p0}.Length > 10 \ \mathtt{OR}\ \mathtt{p0}.Date < 1\_Apr\_1994)$.

on the node `ptCustomers`, which further derives

$(c15)$: $(\forall \mathtt{t} \in \mathtt{calls})\ (\mathtt{t}.Length > 10 \ \mathtt{OR}\ \mathtt{t}.Date < 1\_Apr\_1994)$.

in the view `pt0`.

The exact place where the above inferences are made depends on the optimization framework within which the aggregation reasoning is incorporated. As an example, the predicate movearound [LMS94] technique works in four phases, and would make the following inferences in each:

- (*Initialization Phase*): Initialize predicates into each box. Infer $c1 - c4$, $c8$, and $c11 - c12$.
- (*Pull-up phase*): Infer predicates in each box and pull them up into the parent box. Infer $c5 - c7$.
- (*Push-down phase*): Infer predicates in each box and push them down into each child box. Infer $c9 - c10$, $c13 - c15$.
- (*Cleanup phase*): Remove predicates involving functional terms and predicates that are guaranteed to be true.

6 Related Work

The types of inferences we describe in this paper can be incorporated into several existing query rewrite techniques (e.g., predicate pushdown [Ull89], predicate move-around [LMS94]) and into rule based query optimizers [PHH92] and optimizer generators [GM93] fairly easily and modularly.

The predicate move-around algorithm [LMS94] provides a framework in which predicates are moved in a query graph, and we showed, in Section 5.2 how various aggregation inferences can be made within this framework. Some simple versions of the inferences described in Sections 4.2 and 4.3 were mentioned in [LMS94] as a way of showing the generality of the predicate move-around framework. Functional terms of a different type than the ones in this paper were also used in [LMS94] to reason with functional dependencies.

Ross et al. [RSSS94] considered a subset of our problem, namely a subset of the intra-relation inferences, when different aggregation functions are performed on the *same* relation, attribute and grouping columns (Section 4.2). For the case of a single groupby operation in a view, they studied predicates that can contain arbitrary linear constraints, and gave a complete inference procedure. They considered view-to-relation inferences (Section 4.3), but only when there is a single aggregation function in a grouping operation. The constraint language and the reasoning framework in [RSSS94] was not expressive enough to represent the types of predicates needed for the query optimization shown in Sections 2 and 5.

Gupta et al. [?] use a generalized projection operator to show that aggregation is similar to duplicate elimination in SQL, and that optimizations for the SQL distinct operator can be applied to aggregations. They do not give any rules to infer aggregation predicates between different types of aggregations or between views and relations.

There has been a lot of work on optimizing queries with aggregation in correlated subqueries by way of decorrelation (converting the subqueries into views) [Kim82, GW87, Day87, Mur92], with perhaps a magic-sets transformation to follow [MFPR90, MP94]. A different type of optimization involving aggregation was described by Chaudhuri and Shim [CS94] and Yan and Larson [?]. The main observation in that work is that often it is possible to perform a grouping operation *before* a join or selection operation in the same query block. Doing so may result in more efficient query plans. Our approach is orthogonal to that of decorrelation and commuting groupings with joins since the goal there is to change the structure of the query graph given a set of predicates, while ours is to infer predicates in the query graph without changing the graph. The problem of optimizing queries with aggregation by exploiting materialized views is considered in [?, ?].

7 Conclusions

We have developed a framework in which a system can do reasoning with aggregation constraints. We identified a constraint language that lets us reason with aggregation. The key feature of the language is introduction of functional symbols of the form $f_{\{R,GL,A,Y\}}$ that are identified by the relation, the grouping list, the aggregation function, and the aggregated column. Use of such function symbols lets us relate aggregations done in different parts of the query. We then identified three types of inferencing that needs to be done: (1) intra-relation Inference: Inferencing predicates on functional terms with the same relation, (2) relation-to-view Inference: Inferencing predicates on functional terms on a relation and a view derived from the relation, and (3) view-to-relation Inference: Inferencing ordinary predicates on relations that define an aggregation view from predicates on the view. We presented a set of sound inference rules for each type of inference, and detailed an inference procedure that works in time linear in the number of rules and functional terms. We have several more rules that are similar or a special case of the rules presented here. However, even with all these rules, the inference procedure is not complete. In fact, it follows from [vdM92, MS95] that the satisfiability problem for queries with aggregation is undecidable. Thus, there cannot exist a complete inference procedure for aggregation constraints.

Aggregation constraints are very important in large database applications, where complex decision-support queries rely on reducing data by several different types of aggregation on several combinations of a small number of base tables, and then apply a large number of predicates on the aggregation views to study different fragments of the data and to test different hypotheses. Such queries can greatly benefit from optimization using the type of reasoning outlined above. In this paper we show how the aggregation reasoning framework can be incorporated into a database optimizer. A crucial observation is that the functional terms over which we need to reason is linear in the size of the query.

Beside query optimization, their are several other domains where reasoning with aggregation can be used – logic programming, constraint programming, constraint databases [KKR90, BK95], and global information systems [LSK95].

As future work, we would like to permit an aggregation function to be applied to more than one aggregated column, e.g. $\mathtt{MAX}(Y_1 + Y_2)$. Though the general inference procedure on aggregation constraints is undecidable, it may be possible to identify fragments that are decidable, as done in [RSSS94] for the subcase considered there. Finally, this paper focussed mostly on the logic behind the inference rules. We plan to further explore the problem of controlling the application of the inference rules (as in Section 5).

References

[BK95] A. Brodsky and Y. Kornatzky. The lyric language: Querying constraint objects. In *Proceedings of ACM SIGMOD 1995 International Conference on Management of Data*, San Jose, CA, May 23-25 1995.

[CS94] Surajit Chaudhuri and Kyuseok Shim. Including groupby in query optimization. In Proceedings of VLDB–94, pages 354–366.

[DJLS95] Shaul Dar, H. V. Jagadish, Alon Y. Levy and Divesh Srivastava. *Answering SQL Queries with Aggregation Using Materialized Views.* Working notes of the Post-ILPS95 Workshop on Constraints, Databases and Logic Programming.

[Day87] Umeshwar Dayal. Of nests and trees: A unified approach to processing queries that contain nested subqueries, aggregates, and quantifiers. In *Proceedings of the Thirteenth International Conference on Very Large Databases (VLDB)*, pages 197–208, Brighton, England, September 1-4 1987.

[GM93] Goetz Graefe and William J. McKenna. The volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the Ninth IEEE International Conference on Data Engineering*, Vienna, Austria, April 1993.

[GW87] Richard A. Ganski and Harry K. T. Wong. Optimization of nested SQL queries revisited. In *Proceedings of ACM SIGMOD 1987 International Conference on Management of Data*, pages 23–33, San Francisco, CA, May 1987.

[GHQ95] A. Gupta, V. Harinarayan and D. Quass. *Generalized Projections: A Powerful Approach to Aggregation.* In Proceedings of VLDB–95.

[Hel94] Joseph M. Hellerstein. Practical predicate placement. In Proceedings of SIGMOD–94.

[HS93] Joseph M. Hellerstein and Michael Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of ACM SIG-*

MOD 1993 International Conference on Management of Data, pages 267–276, Washington, DC, May 26-28 1993.

[Kim82] Won Kim. On optimizing an SQL-like nested query. *ACM Transactions on Database Systems*, 7(3), September 1982.

[KKR90] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint query languages. In *Proceedings of the Ninth Symposium on Principles of Database Systems (PODS)*, pages 299–313, Nashville, TN, April 2-4 1990.

[LMS94] Alon Levy, Inderpal Singh Mumick, and Yehoshua Sagiv. Query optimization by predicate movearound. In Proceedings of VLDB–94, pages 96–107.

[LS92] Alon Levy and Yehoshua Sagiv. Constraints and redundancy in datalog. In *Proceedings of the Eleventh Symposium on Principles of Database Systems (PODS)*, pages 67–80, San Diego, CA, June 2-4 1992.

[LSK95] Alon Y. Levy, Divesh Srivastava, and Thomas Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5(2), September, 1995.

[MFPR90] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic is relevant. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 247–258, Atlantic City, NJ, May 23-25 1990.

[MP94] Inderpal Singh Mumick and Hamid Pirahesh. Implementation of magic in starburst. In Proceedings of SIGMOD–94.

[MS95] Inderpal Singh Mumick and Oded Shmueli. How expressive is stratified aggregation. *To Appear in Annals of Mathematics and Artificial Intelligence*, 1995.

[Mur92] M. Muralikrishna. Improved unnesting algorithms for join aggregate SQL queries. In *Proceedings of the Eighteenth International Conference on Very Large Databases (VLDB)*, pages 91–102, Vancouver, Canada, August 23-27 1992.

[PHH92] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/rule based query rewrite optimization in Starburst. In *Proceedings of ACM SIGMOD 1992 International Conference on Management of Data*, pages 39–48, San Diego, CA, June 2-5 1992.

[RSSS94] Kenneth Ross, Divesh Srivastava, Peter Stuckey, and S. Sudarshan. Foundations of aggregation constraints. In Alan Borning, editor, *Principles and Practice of Constraint Programming*, 1994. LNCS 874.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volumes 1 and 2*. Computer Science Press, 1989.

[vdM92] Ronald van der Meyden. *The Complexity of Querying Indefinite Information: Defined Relations, Recursion, and Linear Order.* PhD thesis, Rutgers, The State University of New Jersey, New Brunswick, NJ, October 1992.

[YL95] Weipeng P. Yan and Per-Åke Larson. *Eager Aggregation and Lazy Aggregation.* In Proceedings of VLDB–95.