Rewriting Aggregate Queries using Description Logic

David DeHaan, David Toman, and Grant Weddell
School of Computer Science, University of Waterloo
{dedehaan, david, gweddell}@uwaterloo.ca

Abstract

This paper presents an application of a DL reasoner to the optimization of an object-relational query language. Queries containing aggregate functions are difficult to optimize because care must be taken to ensure that the output value of the aggregate function is not affected. We present a mapping from an object-relational aggregate query to a DL implication problem such that satisfaction of the implication is a sufficient condition for the correctness of a particular logical rewrite rule with respect to a given schema.

1 Introduction

An aggregate function is a function that takes as input a bag of values and outputs a single value. An aggregate query is a query that contains an aggregate function. Aggregate queries are a relatively unexplored area of relational query optimization. However, aggregate functions may occur frequently in certain application domains; for example, query workloads in OLAP and data warehousing environments make substantial use of aggregation [1, 6]. The most commonly discussed aggregate functions are likely those that are defined by SQL—`Sum`, `Average`, `Maximum`, `Minimum`, and `Count`. However, many other aggregates are possible: `Median`, `Standard Deviation`, `Parity`, `Product`, just to name a few with mathematical significance.

Description Logics can be used as a representation for the preconditions of the rules of a query optimizer, as well as for the constraints of an object-relational database schema. One advantage of this approach is that all of the reasoning about the applicability of optimization rules can be reduced to instances of logical implication. The rest of this paper describes a particular query rewrite rule for aggregate queries and how Description Logics with an ability to express uniqueness constraints can be used to reason about the applicability of the rule. This builds directly upon the work in [7, 8].

$$
\begin{array}{rll@{\qquad}rl}
D ::= & C & & (D)^{\mathcal{I}} = & (C)^{\mathcal{I}} \subseteq \Delta \\
| & \top & & & \Delta \\
| & \bot & & & \emptyset \\
| & D_1 \sqcap D_2 & & & (D_1)^{\mathcal{I}} \cap (D_2)^{\mathcal{I}} \\
| & D_1 \sqcup D_2 & & & (D_1)^{\mathcal{I}} \cup (D_2)^{\mathcal{I}} \\
| & \forall f.D & & & \{x : (f)^{\mathcal{I}}(x) \in (D)^{\mathcal{I}}\} \\
| & \neg D & & & \Delta \setminus (D)^{\mathcal{I}} \\
E ::= & D & & (E)^{\mathcal{I}} = & (D)^{\mathcal{I}} \\
| & E_1 \sqcap E_2 & & & (E_1)^{\mathcal{I}} \cap (E_2)^{\mathcal{I}} \\
| & (\mathrm{Pf}_1 = \mathrm{Pf}_2) & & & \{x : (\mathrm{Pf}_1)^{\mathcal{I}}(x) = (\mathrm{Pf}_2)^{\mathcal{I}}(x)\} \\
| & D\{\mathrm{Pf}_1, \ldots, \mathrm{Pf}_k\} \to \mathrm{Pf} & & & \{x : \forall y \in (D)^{\mathcal{I}}. \bigwedge_{i=1}^{k} (\mathrm{Pf})^{\mathcal{I}}(x) = (\mathrm{Pf})^{\mathcal{I}}(y) \\
 & & & & \qquad \Rightarrow (\mathrm{Pf})^{\mathcal{I}}(x) = (\mathrm{Pf})^{\mathcal{I}}(y)\}
\end{array}
$$

Figure 1: Syntax and Semantics of $\mathcal{DLFDE}$

2 Definitions

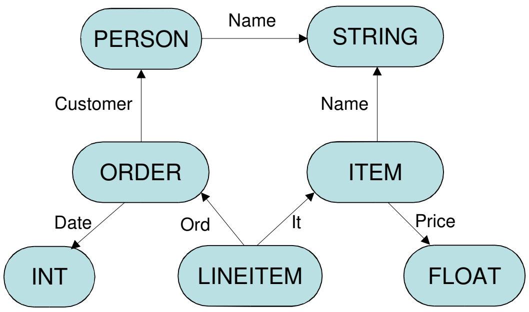
In this section we define a Description Logic called $\mathcal{DLFDE}$ and an object-relational query language called $\mathcal{QLA}$. Both of these languages are extensions of similar languages found in [8].

Definition 1 (Syntax and Semantics of $\mathcal{DLFDE}$)
Let F be a set of attribute names $\{f_1, f_2, \ldots\}$. A path expression *is defined by the grammar "$\mathsf{Pf} ::= f.\,\mathsf{Pf} \mid Id$" for $f \in F$. Let C be a primitive concept description. We define derived* concept descriptions *by the grammar in Figure 1. An* inclusion dependency *is an expression of the form $D \sqsubseteq E$.*
The semantics *of expressions is defined with respect to a structure $(\Delta, \cdot^{\mathcal{I}})$, where Δ is a domain of "objects" $\{e_1, e_2, \ldots\}$ and $(\cdot)^{\mathcal{I}}$ an interpretation function that fixes the interpretations of primitive concepts C to be subsets of Δ and primitive attributes f to be total functions $(f)^{\mathcal{I}} : \Delta \to \Delta$. The interpretation is extended to path expressions, $(Id)^{\mathcal{I}} = \lambda x.x$, $(f.\,\mathsf{Pf})^{\mathcal{I}} = (\mathsf{Pf})^{\mathcal{I}} \circ (f)^{\mathcal{I}}$ and derived concept descriptions D and E as defined in Figure 1. An interpretation* satisfies *an* inclusion dependency *$D \sqsubseteq E$ if $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$. A* terminology *$\mathcal{T}$ consists of a finite set of inclusion dependencies. The* logical implication problem *asks if $\mathcal{T} \models D \sqsubseteq E$ holds; that is, if all interpretations that satisfy all constraints in $\mathcal{T}$ also satisfy $(D)^{\mathcal{I}} \subseteq (E)^{\mathcal{I}}$ (the posed question).*

As usual, a $\mathcal{DLFDE}$ terminology can be used to represent the schema of an object-relational database. In particular, each object type corresponds to a primitive concept. Also, in order for the implication problems to be decidable in $\mathcal{DLFDE}$, certain stratification requirements over equational constraints must be met [8]. In this case, the $\mathcal{DLFDE}$ implication problem is DEXPTIME-complete.

The following example demonstrates how a $\mathcal{DLFDE}$ terminology can be used to represent an object-relational schema. This schema will be used as a running example throughout the paper.

Example 1 Figure 2 illustrates a schema for a simple purchase order database along with the corresponding $\mathcal{DLFDE}$ terminology. The constraints in $\mathcal{T}$ induce



$$
\begin{aligned}
\mathcal{T} = \{ \quad & \mathrm{PERSON} \sqsubseteq \forall Name.\mathrm{STRING} \\
& \quad \sqcap \mathrm{PERSON}\{Name\} \to Id, \\
& \mathrm{ORDER} \sqsubseteq \forall Customer.\mathrm{PERSON} \\
& \quad \sqcap \forall Date.\mathrm{INT} \\
& \quad \sqcap \mathrm{ORDER}\{Customer, Date\} \to Id, \\
& \mathrm{ITEM} \sqsubseteq \forall Name.\mathrm{STRING} \\
& \quad \sqcap \forall Price.\mathrm{FLOAT} \\
& \quad \sqcap \mathrm{ITEM}\{Name\} \to Id, \\
& \mathrm{LINEITEM} \sqsubseteq \forall Ord.\mathrm{ORDER} \\
& \quad \sqcap \forall It.\mathrm{ITEM}\}
\end{aligned}
$$

Figure 2: Purchase Order Database Schema

$$
\begin{array}{lll}
Q ::= & D \ \texttt{as} \ a & [\![Q]\!]\mathcal{I} \equiv \ \{\langle a\!:\!e\rangle : e \in (D)^{\mathcal{I}}\} \\
\mid & (a_1.\mathrm{Pf}_1 = a_2.\mathrm{Pf}_2) & \{\langle a_1\!:\!e_1, a_2\!:\!e_2\rangle : (\mathrm{Pf}_1)^{\mathcal{I}}(e_1) = (\mathrm{Pf}_2)^{\mathcal{I}}(e_2)\} \\
\mid & \texttt{true} & \{\langle\ \rangle\} \\
\mid & \texttt{empty} \ a_1, \ldots, a_n & \emptyset \\
\mid & \texttt{select} \ a_1, \ldots, a_n \ Q_1 & \{\!|\langle a_1\!:\!t@a_1, \ldots, a_n\!:\!t@a_n\rangle : t \in [\![Q_1]\!]\mathcal{I}|\!\} \\
\mid & \texttt{elim} \ a_1, \ldots, a_n \ Q_1 & \{\langle a_1\!:\!t@a_1, \ldots, a_n\!:\!t@a_n\rangle : t \in [\![Q_1]\!]\mathcal{I}\} \\
\mid & \texttt{agg} \ a_1, \ldots, a_n, \ \beta(a) \ \texttt{as} \ a_\beta \ Q_1 & \{t_1 \cup \{a_\beta\!:\!\mathrm{AV}_{t_1}\}, \ldots, t_k \cup \{a_\beta\!:\!\mathrm{AV}_{t_k}\} : \\
 & & \quad \{t_1, \ldots, t_k\} = [\![\texttt{elim} \ a_1, \ldots, a_n \ Q_1]\!]\mathcal{I}\} \\
 & & \quad \text{where } \mathrm{AV}_t = \beta(\{\!|e \in \Delta : t \cup \{a\!:\!e\} \in \\
 & & \qquad\qquad [\![\texttt{select} \ a_1, \ldots, a_n, a \ Q_1]\!]\mathcal{I}|\!\}) \\
\mid & \texttt{from} \ Q_1, \ Q_2 & [\![Q_1]\!]\mathcal{I} \bowtie [\![Q_2]\!]\mathcal{I} \\
\mid & Q_1 \ \texttt{union all} \ Q_2 & [\![Q_1]\!]\mathcal{I} \uplus [\![Q_2]\!]\mathcal{I}
\end{array}
$$

Figure 3: Syntax and Semantics of $\mathcal{QLA}$

the primitive concepts PERSON, ORDER, ITEM, LINEITEM, STRING, INT, and FLOAT.

Definition 2 (Object-Relational Query Language $\mathcal{QLA}$)
Let $V \subseteq F$ be a set of query variables $\{a_1, a_2, \ldots\}$.[1]

The syntax and semantics of $\mathcal{QLA}$ are shown in Figure 3. The semantics of a query Q, denoted $[\![Q]\!]$, is a function that maps interpretations to bags of tuples. Query variables occurring in a given query are assumed to satisfy standard conditions of well-formedness, and are never reused within the query. Also, for a given tuple $t = \langle a_1\!:\!e_1, \ldots, a_n\!:\!e_n\rangle \in [\![Q]\!]\mathcal{I}$ computed by query Q over interpretation $\mathcal{I}$, we write $t@a_i$ to denote e_i.

Although the agg operator defined for $\mathcal{QLA}$ is restricted to taking only a single aggregate function β, this does not limit the expressive power of the language because any agg operator containing multiple aggregate functions can be rewritten as the natural join of multiple agg operators, each taking a single aggregate

[1] We have selected V as a subset of attribute names in order to reduce the need for additional notational baggage. In particular, query variables "become" attribute names in our abstraction of queries as $\mathcal{DLFDE}$ concept descriptions.

function. The following two definitions state properties of aggregate functions that are exploited by the proposed query rewrite rule.

Definition 3 (Duplicate Insensitive Aggregate Function) *An aggregate function β is* duplicate insensitive *if and only if for any bag of values B,* $\beta(B) = \beta(\{e : e \in B\})$.

Definition 4 (Cleanly Composable Aggregate Functions) *Aggregate functions β and β'* cleanly compose *if and only if for any two bags of values B_1 and B_2, $\beta(\{\!|\beta'(B_1 \uplus B_2)|\!\}) = \beta(\{\!|\beta'(B_1), \beta'(B_2)|\!\})$.*

3 Related Work

Testing the equivalence of arbitrary aggregate queries has been characterized for conjunctive [9], positive [4], and disjunctive queries with limited negation [3]. However, this body of work does not consider schema information, and it only considers queries where the aggregation operator is at the root of the query tree.

Khizder et al [7] use Description Logic to deduce the correctness (with respect to a schema) of a rewrite rule moving subqueries out of the scope of the `elim` operator. Unfortunately, their construction requires converting the query to a normal form that is only possible because their query language is conjunctive and does not contain aggregate functions. Liu et al [8] extend this to positive queries by introducing the concept of *query context* which eliminates the reliance on a normal form.

Definition 5 (Query Context) *A* query context $Q[\,]$ *is a $\mathcal{QLA}$ expression containing a single instance of the special terminal symbol $[\,]$. For $Q' \in \mathcal{QLA}$, the expression $Q[Q']$ denotes the syntactic substitution of Q' for $[\,]$.*

Yan and Larson [10, 11] consider a rewrite that commutes a top-level aggregation operator with the join of two base relations. Let C denote the base relation pulled out of the aggregate block, V the set of attributes in the selection list of the aggregation operator, and R the attributes from C that form the join key with the rest of the aggregation block. They show that a necessary and sufficient condition for the correctness of the rewrite with respect to schema $\mathcal{T}$ (assuming an arbitrary aggregate function) is that both the functional dependencies $V \to R$ and $V \cup R \to \mathrm{Key}(C)$ are implied by the constraints in $\mathcal{T}$ and the rest of the query. This condition has been used within the context of a system for answering queries using aggregate views [5], but it no longer remains necessary when the aggregation block is not at the root of the query tree. Similar rewrites have been considered in [2, 6].

4 DL Reasoning for Rewriting Aggregate Queries

In this section we show how a reasoner for $\mathcal{DLFDE}$ terminologies can be applied to rewriting aggregate queries expressed in $\mathcal{QLA}$.

The rewrite rule examined by Yan and Larson [10] and described in the previous section can be generalized in several ways. First of all, the two functional dependencies in the condition can be combined into the single equivalent dependency $V \rightarrow Id_C$, where Id_C is a unique tuple identifier for tuples from relation C (follows since by definition $Id_C \rightarrow R$). Next, C can be allowed to be an arbitrary subquery. Finally, rather than considering only top-level aggregation operators, one can allow the aggregation operator to occur at arbitrary positions within a larger containing query. One impact of this last generalization is that the functional dependency is no longer a necessary condition for correctness of the rewrite; it is possible for the overall queries to be equivalent without the aggregate subqueries being equivalent. This can happen when the rewritten aggregate block is nested within another aggregation operator whose aggregate function β' ignores or compensates for the changes introduced by the rewrite.

Example 2 The following queries are equivalent even though the dependency $\{\mathsf{Date}\} \rightarrow o.Id$ does not hold for all databases.

```
E1: agg Sum(ItemCnt) as TotalItemCnt
    [ agg Date, Count(l) as ItemCnt
      from ORDER as o, LINEITEM as l
        (o = l.Ord), (o.Date = Date) ]

E2: agg Sum(ItemCnt) as TotalItemCnt
    [ select Date, ItemCnt
      from (from ORDER as o, (o.Date = Date)),
        (agg Count(l) as ItemCnt
        from LINEITEM as l, (o = l.Ord)) ]
```

Example 3 Modify E_1 and E_2 from Example 2 by replacing the "Sum(ItemCnt) as TotalItemCnt" with "Max(Date) as MaxDate"; the queries remain equivalent.

We design a rewrite rule that captures the above situations, as well as others similar to them.

Definition 6 (Nearest Enclosing Aggregation Operator)
Given a query context $Q[\,]$ *the* nearest enclosing aggregation operator *to* $[\,]$ *(if one exists) is the* `elim` *or* `agg` *operator such that* $Q[\,] = Q''[\texttt{elim}\ V'\ Q'[\,]]$ *or* $Q[\,] = Q''[\texttt{agg}\ V',\ \beta'(a_\beta)\ \texttt{as}\ a_{\beta'}\ Q'[\,]]$ *where* Q' *does not contain* `elim` *or* `agg`.

To utilize a DL reasoner for testing correctness of the rewrite we need to represent in $\mathcal{DLFDE}$ the constraints induced by the query structure. For subqueries Q and query context $Q[\,]$, we introduce the inclusion dependency $C_Q \sqsubseteq (E_Q \sqcap E_{Q[\,]})$, where C_Q is a primitive concept corresponding to the result of the query. Figure 4 defines E_Q and $E_{Q[\,]}$, as well as α_Q and $\alpha_{Q[\,]}$ which are the sets of variable names that occur in Q and $Q[\,]$, respectively. Note that each unique object identifier Id_Q is an artifact of the translation to $\mathcal{DLFDE}$ and does not occur as an actual attribute within the object-relational system.

To identify compensation by nested aggregation operators, we first extend the terminology $\mathcal{T}$ to capture certain properties of each aggregate function β; specifically, whether β is duplicate insensitive (c.f. Definition 3) or cleanly composable with another function β' (c.f. Definition 4). Assume that for each aggregate function β known to the system we have a special primitive concept C_β. If β is duplicate insensitive, we extend $\mathcal{T}$ with the constraint $C_\beta \sqsubseteq C_{\mathrm{DI}}$. If two aggregate functions β and β' are cleanly composable, we extend $\mathcal{T}$ with the

Q	α_Q	E_Q
D as a	$\{a\}$	$(\forall a.D) \sqcap \top\{a\} \rightarrow Id_Q$
$(a_1.\mathrm{Pf}_1 = a_2.\mathrm{Pf}_1)$	$\{a_1, a_2\}$	$(a_1.\mathrm{Pf}_1 = a_2.\mathrm{Pf}_1) \sqcap \top\{a_1, a_2\} \rightarrow Id_Q$
true	$\{\}$	$\top$
empty V	V	$\perp$
select V Q_1	V	$E_{Q_1} \sqcap \top\{Id_{Q_1}\} \rightarrow Id_Q$
elim V Q_1	V	$E_{Q_1} \sqcap \top\{V\} \rightarrow Id_Q$
agg V, $\beta(a)$ as a_β Q_1	$V \cup \{a_\beta\}$	$E_{Q_1} \sqcap \forall a_\beta.C_\beta \sqcap \top\{V\} \rightarrow Id_Q$
from Q_1, Q_2	$\alpha_{Q_1} \cup \alpha_{Q_2}$	$E_{Q_1} \sqcap E_{Q_2} \sqcap \top\{Id_{Q_1}, Id_{Q_2}\} \rightarrow Id_Q$
Q_1 union all Q_2	$\alpha_{Q_1} \cup \alpha_{Q_2}$	$E_{Q_1} \sqcup E_{Q_2}$

$Q[]$	$\alpha_{Q[]}$	$E_{Q[]}$	$\gamma_{Q[]}$	$A_{Q[]}$
$[]$	$\{\}$	$\top$	$\{\}$	$\top$
Q_1[from Q_2, $[]$] or Q_1[from $[]$, Q_2]	$\alpha_{Q_1[]} \cup \alpha_{Q_2}$	$E_{Q_1[]} \sqcap E_{Q_2}$	$\gamma_{Q_1[]}$	$A_{Q_1[]}$
Q_1[select V $[]$]	$\alpha_{Q_1[]} \cup V$	$E_{Q_1[]}$	$\gamma_{Q_1[]}$	$A_{Q_1[]}$
Q_1[elim V $[]$]	$\alpha_{Q_1[]} \cup V$	$E_{Q_1[]}$	$\{\}$	C_{elim}
Q_1[agg V, $\beta(a)$ as a_β $[]$]	$\alpha_{Q_1[]} \cup V$	$E_{Q_1[]}$	$\{a\}$	$C_{\mathrm{agg}} \sqcap \forall\delta.C_\beta$
Q_1[Q_2 union all $[]$] or Q_1[$[]$ union all Q_2]	$\alpha_{Q_1[]}$	$E_{Q_1[]}$	$\gamma_{Q_1[]}$	$A_{Q_1[]}$

Figure 4: Capturing Structural Constraints

constraint $C_\beta \sqsubseteq \forall\beta'.C_{\mathrm{CC}}$. These constraints require that the name β' does not appear as an attribute name anywhere in $\mathcal{T}$ or in the query, and that C_{DI} and C_{CC} are reserved primitive concept names.

The type of the nearest enclosing aggregation operator (`agg` or `elim`) is captured by extending the terminology with dependencies of the form $C_Q \sqsubseteq A_{Q[]}$ where $A_{Q[]}$ is defined as in Figure 4. Note that $A_{Q[]}$ requires reserved primitive concept names C_{agg} and C_{elim} and reserved attribute name δ. Figure 4 also defines $\gamma_{Q[]}$ which is the set containing the variable name of the input argument to the aggregate function of the nearest enclosing aggregation operator in $Q[]$.

Theorem 1 *The queries*

E_1: $Q[$`agg` V, $\beta(a)$ `as` a_β `from` Q_1, $Q_2]$, and
E_2: $Q[$`select` V, a_β `from` Q_1, (`agg` W, $\beta(a)$ `as` a_β Q_2)$]$

where $W = (V \cup \alpha_{Q_1}) \cap \alpha_{Q_2}$ are equivalent, with respect to $\mathcal{T}$, if any of the following conditions are satisfied.

1. $\mathcal{T} \cup \{C_Q \sqsubseteq E_{Q[]} \sqcap E_{Q_1} \sqcap E_{Q_2}\} \models C_Q \sqsubseteq C_Q\{V \cup \alpha_{Q[]}\} \rightarrow Id_{Q_1}$
2. $a_\beta \notin \alpha_{Q[]}$, $a_\beta \in \gamma_{Q[]}$, *and*

$$\mathcal{T} \cup \{C_Q \sqsubseteq A_{Q[]}\} \models C_Q \sqsubseteq C_{agg} \sqcap \forall\delta.\forall\beta.C_{CC}$$

3. $a_\beta \notin \alpha_{Q[]}$, *and*

$$\mathcal{T} \cup \{C_Q \sqsubseteq A_{Q[]}\} \models C_Q \sqsubseteq C_{elim} \sqcup (C_{agg} \sqcap \forall\delta.C_{DI})$$

Proof (sketch): Let E_1' and E_2' denote the contents of $[\,]$ for E_1 and E_2.

Condition 1: Suppose that $E_1 \not\equiv E_2$ Then there exists some interpretation $\mathcal{I}$ with a fixed valuation of $\alpha_{Q[]}$ such that $[\![E_1']\!]\mathcal{I} \not\equiv [\![E_2']\!]\mathcal{I}$. This is only possible if the interpretation $[\![E_2']\!]\mathcal{I}$ includes two distinct tuples which agree on attributes $V \cup \alpha_{Q[]}$. However, these tuples can be used to create an interpretation $\mathcal{I}$ containing only two objects $s, t \in (C_{E_2'})^{\mathcal{I}}$ that agree on attributes $V \cup \alpha_{Q[]}$ but disagree on $Id_{E_2'}$. Define $Q_2' = (\text{agg } W,\, \beta(a) \text{ as } a_\beta\ Q_2)$. As these objects satisfy the left-hand side of the constructed implication, the E_Q construction for the `select` and `from` operators dictates that s and t must differ on either Id_{Q_1} or Id_{Q_2}. However, by the E_Q rule for the `agg` operator, disagreement on Id_{Q_2} implies disagreement on $W \subseteq (V \cup \alpha_{Q_1})$ which implies disagreement on Id_{Q_1}. Therefore, we have

$$\mathcal{T} \cup \{C_Q \sqsubseteq E_{Q[]} \sqcap E_{Q_1} \sqcap E_{Q_2}\} \not\models C_Q \sqsubseteq C_Q\{V \cup \alpha_{Q[]}\} \to Id_{Q_1}.$$

Condition 2: By construction, this condition is only true when $[\,]$ falls into the scope of an aggregation operator and the nearest such operator to $[\,]$ is $Q''[\text{agg } V',\, \beta'(a_\beta) \text{ as } a_{\beta'}\ Q'[\,]]$ such that β' and β are cleanly composable. Since $a_\beta \notin \alpha_{Q[]}$, every tuple in $[\![\text{from } Q_1,\, Q_2]\!]\mathcal{I}$ corresponding to the pair (Id_{Q_1}, Id_{Q_2}) that contributes to a single group of values for V' in $[\![\text{agg } V',\, \beta'(a_\beta) \text{ as } a_{\beta'}\ Q'[E_1']]\!]\mathcal{I}$ also contributes to the same group in $[\![\text{agg } V',\, \beta'(a_\beta) \text{ as } a_{\beta'}\ Q'[E_2']]\!]\mathcal{I}$ and vice-versa. It follows from Definition 4 that the values of $a_{\beta'}$ for each group are unchanged, so $E_1 \equiv E_2$.

Condition 3: By construction, this condition is only true when the nearest enclosing aggregation operator to $[\,]$ is either `elim` or `agg` with a duplicate insensitive aggregate function. However, the output of this aggregation operator is not affected by any duplicate tuples introduced by the rewrite, and since $a_\beta \notin \alpha_{Q[]}$, the value of a_β is never used (including in the selection list of the aggregation operator) so a_β does not affect the output of the query. Thus, $E_1 \equiv E_2$.

The rewrite rule from Theorem 1 is useful for pushing aggregation operators below joins, which reduces the cost of calculating the aggregate function. It can also be used to pull aggregation operators above a join, primarily for the purpose of making the query match the definition of a materialized aggregate view.

Example 4 The following queries can be reasoned equivalent by Condition 1 of Theorem 1.

E_1:
```
select Name, Date, LineCnt
  from (select Name from PERSON as p,
          (p.Name = Name)),
    [ (agg Date, Count(l) as LineCnt
        from ORDER as o, LINEITEM as l,
          (o.Date = Date), (l.Ord = o),
          (o.Customer.Name = Name)) ]
```

E_2:
```
select Name, Date, LineCnt
  from (select Name from PERSON as p,
          (p.Name = Name)),
    [ (select Date, LineCnt
        from ORDER as o, (o.Date = Date),
          (o.Customer.Name = Name),
          (agg o, Count(l) as LineCnt
           from LINEITEM as l, (l.Ord = o))) ]
```

The queries given in Examples 2 and 3 can be reasoned equivalent using Conditions 2 and 3, respectively.

5 Summary

We have presented an application of a DL reasoner in testing sufficient conditions for the correctness of a query optimization rule for the object-relational query language $\mathcal{QLA}$ which includes aggregation and union operators. One novel feature of the constructed implication problems is the capturing of functional dependencies over virtual object identifiers induced by the structure of the query. Future work includes establishing completeness results for logical rewrites involving aggregation operators.

References

[1] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.

[2] S. Chaudhuri and K. Shim. Including group-by in query optimization. In *Proc. 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 354–366. Morgan Kaufmann, 1994.

[3] S. Cohen, W. Nutt, and Y. Sagiv. Equivalences among aggregate queries with negation. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 215–226. ACM Press, 2001.

[4] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. 18th Symposium on Principles of Database Systems (PODS 1999)*, pages 155–166. ACM Press, 1999.

[5] J. Goldstein and P.-Å. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Proc. 2001 ACM SIGMOD International Conference on Management of Data*, pages 331–342. ACM Press, 2001.

[6] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proc. 21th International Conference on Very Large Data Bases (VLDB'95)*, pages 358–369. Morgan Kaufmann, 1995.

[7] V. L. Khizder, D. Toman, and G. Weddell. Reasoning about duplicate elimination with description logic (preliminary report). In *Computational Logic (CL/DOOD 2000)*, volume 1861 of *Lecture Notes in Computer Science*, pages 1017–1032. Springer, 2000.

[8] H. Liu, D. Toman, and G. Weddell. Fine grained information integration with description logics. In *Proc. 2002 International Workshop on Description Logics (DL2002)*, volume 53, 2002.

[9] W. Nutt, Y. Sagiv, and S. Shurin. Deciding equivalences among aggregate queries. In *Proc. 17th Symposium on Principles of Database Systems (PODS 1998)*, pages 214–223. ACM Press, 1998.

[10] W. P. Yan and P.-Å. Larson. Performing group-by before join. In *Proc. 10th International Conference on Data Engineering (ICDE'94)*, pages 89–100. IEEE Computer Society, 1994.

[11] W. P. Yan and P.-Å. Larson. Eager aggregation and lazy aggregation. In *Proc. 21st International Conference on Very Large Data Bases (VLDB'95)*, pages 345–357. Morgan Kaufmann, 1995.