An Ordered Bag Semantics for SQL

by

Hamid R. Chinaei

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Semantic query optimization is an important issue in many contexts of databases including information integration, view maintenance and data warehousing and can substantially improve performance, especially in today's database systems which contain gigabytes of data. A crucial issue in semantic query optimization is query containment. Several papers have dealt with the problem of conjunctive query containment [Cha92, KV98, LS97]. In particular, some of the literature admits SQL like query languages with aggregate operations such as sum/count [Coh05, NSS98]. Moreover, since real SQL requires a richer semantics than set semantics, there has been work on bag-semantics for SQL, essentially by introducing an interpreted column. One important technique for reasoning about query containment in the context of bag semantics is to translate the queries to alternatives using aggregate functions and assuming set semantics.

Furthermore, in SQL, *order by* is the operator by which the results are sorted based on certain attributes and, clearly, ordering is an important issue in query optimization. As such, there has been work done in support of ordering based on the application of the domain. However, a final step is required in order to introduce a rich semantics in support.

In this work, we integrate set and bag semantics to be able to reason about real SQL queries. We demonstrate an ordered bag semantics for SQL using a relational algebra with aggregates. We define a set algebra with various expressions of interest, then define syntax and semantics for bag algebra, and finally extend these definitions to ordered bags. This is done by adding a pair of additional interpreted columns to computed relations in which the first column is used in the standard fashion to capture duplicate tuples in query results, and the second adds an ordering priority to the output. We show that the relational algebra with aggregates can be used to compute these interpreted columns with sufficient flexibility to work as a semantics for standard SQL queries, which are allowed to include *order by* and *duplicate preserving select* clauses. The reduction of a workable ordered bag semantics for SQL to the relational algebra with aggregates - as we have developed it - can enable existing query containment theory to be applied in practical query containment.

Acknowledgments

I would like to express my deep and sincere gratitude to people, without whom this thesis would not have been possible.

... my supervisor Professor Grant Weddell for leading and supporting this research, and for being both an advisor and a friend.

... my readers Ihab Francis Ilyas and David Toman for their valuable comments and suggestions.

... my siblings here Amir and Leila for their guidance and support, and the rest of my family for their never-ending encouragement and support.

... my friends all for helping me out throughout this work.

To all of them and the ones I did not mention explicitly, many, many thanks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A crucial issue in semantic query optimization is query containment. Several papers have dealt with the problem of conjunctive query containment [Cha92, KV98, LS97]. In particular, some of literature admits SQL-like query languages with aggregate operations such as sum/count [Coh05, NSS98]. Moreover, since real SQL requires a richer semantics than set semantics, there have been works on bag-semantics for SQL, essentially by introducing an interpreted column. One important way of reasoning about query containment in the context of bag semantics is to translate to alternative queries using aggregate functions and assuming set semantics.

Furthermore, in SQL, *order by* is the operator by which the results are sorted based on certain attributes and clearly ordering is an important issue in query optimization. As such, there have been some works done for support of ordering based on the application of the domain, for instance those for order optimization in IBM's DB2/CS introduced in [SSM96] and those of supporting top-k queries in [LCIS05, LSCI05]. However, a final step for introducing some rich semantics for support of ordering is demanded. In this work, we define an ordered bag semantics for first order queries using a relational language with aggregates.

Moreover, database queries are investigated in set semantics, bag semantics, and combined semantics. In set semantics, the results of queries are sets as well as the databases. In bag semantics, both queries and databases are bags. In combined semantics, the semantics is based on a combination of set and bag semantics. Since our defined order bag semantics is based on set semantics, it is a support for query

containment under combined semantics which is a fascinating topic in semantic query optimization.

1.1 Our Contribution

In this work, similar to [CW93], based on directed graph models, we introduce three domains of set, bag, and ordered bags. First, a syntax and semantics of a set algebra are introduced. Then, syntax is extended for bag and ordered bag queries. This is done by adding a pair of additional interpreted columns to computed relations in which the first column is used in the standard fashion to capture duplicate tuples in query results, and the second adds an ordering priority to output. To be able to derive the semantics of bag and ordered bag domains, we introduce two mapping functions $\mathcal{B}$ and $\mathcal{OB}$, which map bag and ordered bag queries to those of set domain, respectively. As a result, the semantics for bag and ordered bag domain are derived from that of set domain.

1.2 Related Work

Simmen, et al., introduced the *Reduce* technique for order optimization [SSM96]. The reduction method is used to avoid sorting whenever possible because of keys, functional dependencies, indexes, or predicates. Moreover, the optimizer is able to realize pushed down sorts in order to avoid insufficient sorts.

Li, et al., using aggregates, developed a framework for support of ranking (top-k) queries as a first level expression [LCIS05]. Top-k queries are queries that provide only the first k query results. For instance, in a simple scenario, selecting the minimum value for an attribute such as A in a table such as R, is a form of top-k query where we are interested in finding only the top 1 value for the attribute A in the table R. The rules defined in [LCIS05] can be used by a query optimizer to do an ordering operation interleaved with other expressions, rather than doing the ordering uniformly after other operations. These defined rules can lead to more efficient processing of order queries. Using a subset of first order queries in our syntax, we are able to capture a subset of top-k queries, including the query mentioned above, and

therefore a formal semantics for such queries can be derived in set semantics.

Coburn and Weddell introduced an algebra for representing SQL queries [CW93], which they used to explore both high level queries (non- procedural queries), as well as lower level query plans (procedural queries). Their work is based on directed graph models, where vertices are used for representing objects of the domain, and names and arcs for attributes of objects. Coburn and Weddell demonstrated the method of using sets of lists for representing results of queries, where lists show the order of tuples. More precisely, the order of appearance of the tuples in the list shows the order of the results, and different lists in a set show the possible ordering of the results. Moreover, Coburn and Weddell define a calculus for query rewriting for the purpose of query containment. They show that their rules for query rewriting are sound.

Also, there have been works in bag semantics in the context of multi-set algebra. In particular, Grefen and de By introduced a practical theoretical approach for bag semantics in Relational Databases [GdB94] . They defined relational algebra expressions of union, minus, cross product, selection, projection, intersection and join, as well as some aggregate functions such as count, sum, average, min and max for multi-sets. Moreover, the authors showed some expression equivalences in query rewriting, as required for multi-set relational programs.

Furthermore, there have been also a number of works on combining set and bag semantics. Cohen argued that real SQL queries combine set and bag-set[1] semantics and investigated query containment for combined semantics based on homomorphism method for different class of queries, such as conjunctive and quasilinear queries.

1.3 Thesis Overview

The outline of the rest of the thesis is as follows. In Chapter 2, we briefly describe the background knowledge needed for this work. Notions of concepts, attributes, and databases are introduced. Moreover, we define well-formed and finite set, bag, and ordered bag queries and outline the syntax for well-formed set, bag, and ordered bag queries. Chapter 3 is the core of the thesis where we describe a semantics for well-

[1]In bag-set semantics, while the databases are sets, the result of quires are bags.

formed set queries followed by two mapping functions for mapping of bag and ordered bag queries to set queries. Ordered bag translation of order by and order preserving select to the set semantic domain are described in detail in the last three subsections of this chapter. In Chapter 4, we discuss our contribution followed by addressing the future directions.

Chapter 2

Syntax

In this chapter, we describe three algebras, $\mathcal{Q}$, $\mathcal{BQ}$, and $\mathcal{OBQ}$, for set, bag, and ordered bag domains, respectively. We introduce our underlying domain of work and describe notions of well-formed and finite queries to be able to outline the syntax of well-formed queries for $\mathcal{Q}$, $\mathcal{BQ}$, and $\mathcal{OBQ}$. Lastly, we demonstrate that our defined syntax is capable of capturing relational algebra and expressing SQL-like queries.

2.1 Concepts, Attributes, and Databases

We take the natural numbers, $\mathbb{N} = \{1, 2, \dots\}$, as a common universal domain. We refer to the element of our underlying domain as objects which can be concepts of the form C_i, each of which can have some attributes of the form A, B, A_i or B_i. The two sets of $\mathbf{C}$ and $\mathbf{A}$ represent primitive concepts and attributes:

$$\mathbf{C} = \{C_1, C_2, \dots\}$$

$$\mathbf{A} = \{A, B, A_1, B_1, A_2, B_2, \dots\} \cup \{\texttt{Cnt}, \texttt{Ord}, \texttt{Id}\}$$

$$Pf = \texttt{Id} \mid A_i.Pf$$

The notation Pf, used for path functions, is a finite number of attribute compositions where `Id` is a reserved attribute for the identification of objects. Moreover, `Cnt` and `Ord` are two reserved attributes that are used for count and order of results of a query, which we describe more in detail later in this work.

Table 2.1: Employee Table 1, EMP

	Eno	Sal
n_1	3158	39000
n_2	2467	39000

A database $\mathcal{I}$ is an interpretation function over concepts, attributes, and path functions. We use, e.g., the notation $(C)^{\mathcal{I}}$ vs. $(C)(\mathcal{I})$ to refer to the interpretation of C over database $\mathcal{I}$:

$$(C_i)^{\mathcal{I}} \subseteq \mathbb{N}, \text{ in particular } (C_i)^{\mathcal{I}} \text{ is a finite subset of } \mathbb{N}$$

$$(A_i)^{\mathcal{I}} : \mathbb{N} \to \mathbb{N}, \text{ in particular } (A_i)^{\mathcal{I}} \text{ is a total function over } \mathbb{N}$$

$$(\texttt{Id})^{\mathcal{I}} \equiv \{(e, e) | e \in \mathbb{N}\}$$

$$(A.Pf)^{\mathcal{I}} \equiv \{(e_1, e_2) | (Pf)^{\mathcal{I}}((A)^{\mathcal{I}}(e_1)) = e_2\}$$

In Chapter 3, we extend the definition of $\mathcal{I}$ to apply to queries.

Example 2.1.1 *The Relation* $\texttt{EMP}(\texttt{Eno}, \texttt{Sal})$ *and two tuples* n_1 *and* n_2 *in Table 2.1 can be encoded in our domain as in Figure 2.1. We have the following:*

$$\{n_1, n_2\} \subseteq (\texttt{EMP})^{\mathcal{I}},$$
$$(\texttt{Eno.Id})^{\mathcal{I}}(n_1) = 3158,$$
$$(\texttt{Sal.Id})^{\mathcal{I}}(n_2) = 39000.$$

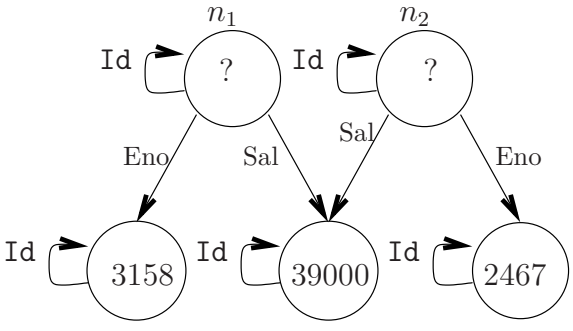

Figure 2.1: Employee Example

Table 2.2: EMP

	Eno	Sal
n_1	3158	39000
n_2	2467	39000
n_3	5688	30000

Table 2.3: WORKSIN

	Eno	Dno
n_4	3158	20
n_5	3158	30
n_6	2467	10
n_7	5688	20

Table 2.4: DEPT

	Dno	Loc
n_8	20	3
n_9	30	1
n_{10}	10	3

Question marks stand for some (unknown) natural numbers which, in turn, stand for employee objects.

Example 2.1.2 *Consider a database with relations* EMP(Eno, Sal), WorksIn(Eno, Dno), *and* Dept(Dno, Loc) *and the database instance in Tables 2.2, 2.3, and 2.4. In our domain the three relations can be thought of as three concepts, and for this database instance, we have the followings for the interpretations of these three concepts:*

$$\{n_1, n_2, n_3\} \subseteq (\texttt{EMP})^{\mathcal{I}},$$
$$\{n_4, n_5, n_6, n_7\} \subseteq (\texttt{WORKSIN})^{\mathcal{I}},$$
$$\{n_8, n_9, n_{10}\} \subseteq (\texttt{DEPT})^{\mathcal{I}}.$$

For interpretations of the attributes, for instance we have the followings:

$$(\texttt{Eno.Id})^{\mathcal{I}}(n_1) = 3158,$$
$$(\texttt{Sal.Id})^{\mathcal{I}}(n_1) = 39000,$$
$$(\texttt{Eno.Id})^{\mathcal{I}}(n_4) = 3158,$$
$$(\texttt{Dno.Id})^{\mathcal{I}}(n_4) = 20,$$
$$(\texttt{Dno.Id})^{\mathcal{I}}(n_8) = 20,$$
$$(\texttt{Loc.Id})^{\mathcal{I}}(n_8) = 3.$$

As the two examples reveal, we are able to encode relational databases and instances in our domain based on the defined underlying domain that is based on directed graph models. In this model, relations are encoded in concepts and attributes in path functions of size two, i.e., attributes composed with the Id attribute. From now on, whenever it is simpler to understand the syntax, we disregard the attribute Id.

2.2 Syntax of set Queries ($\mathcal{Q}$), Bag Queries ($\mathcal{BQ}$), and Ordered Bag Queries ($\mathcal{OBQ}$)

Conjunctive queries, a basic class of queries for relational databases, are a simple class of queries that have properties of decidability for query containment [CGV05]. First order queries are a more general class of queries which can express negation in addition to what can be expressed by conjunctive queries. We study three object-relational languages: $\mathcal{Q}$, $\mathcal{BQ}$, and $\mathcal{OBQ}$, for set, bag, and ordered bag first order queries, respectively. Figure 2.2 shows domains of these three dialects where $\mathcal{Q}$ is a subset of $\mathcal{BQ}$, and $\mathcal{BQ}$, in turn is a subset of $\mathcal{OBQ}$.

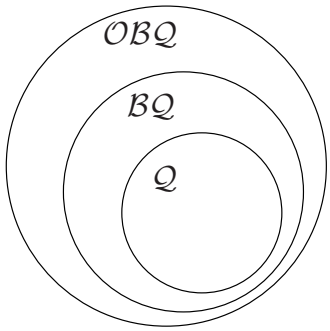

Figure 2.2: $\mathcal{Q}$, $\mathcal{BQ}$, AND $\mathcal{OBQ}$ DOMAINS

In the following three subsections, we define well-formed and finite queries of $\mathcal{Q}$, $\mathcal{BQ}$, and $\mathcal{OBQ}$, respectively. Figure 2.3 shows the syntax of three dialects of well-formed queries for $\mathcal{Q}$, $\mathcal{BQ}$, and $\mathcal{OBQ}$. Checkmarks show the productions that form the grammar for each of the dialects.

2.2.1 Well-formed and Finite $\mathcal{Q}$

In this subsection, we provide mutual definitions for well-formed and finite queries. Well-formed queries are a subset of queries which are allowed in the syntax, i.e., there is a semantics for a well-formed query. Finite queries are a subset of queries which produces finite number of elements as results. The set of well-formed queries ($\mathcal{WQ}$) and that of finite queries ($\mathcal{FQ}$) is defined in Definitions 2.2.1 and 2.2.2, respectively.

Definition 2.2.1 *The set of well-formed queries, denoted $\mathcal{WQ}$, is the smallest set satisfying the following conditions:*

		Q ::=		$\mathcal{Q}$	$\mathcal{BQ}$	$\mathcal{OBQ}$	$\alpha(Q)$
1.	*(reference)*		C `as` A	✓	✓	✓	$\{A\}$
2.	*(selection)*	|	$A_1.Pf_1 = A_2.Pf_2$	✓	✓	✓	$\{A_1, A_2\}$
3.	*(projection)*	|	`elim` $\{A_1, \ldots, A_n\}$ Q	✓	✓	✓	$\{A_1, \ldots, A_n\}$
4.	*(null tuple)*	|	`true`	✓	✓	✓	$\emptyset$
5.	*(natural join)*	|	`from` Q_1, Q_2	✓	✓	✓	$\alpha(Q_1) \cup \alpha(Q_2)$
6.	*(empty set)*	|	`empty` $\{A_1, \ldots, A_n\}$	✓	✓	✓	$\{A_1, \ldots, A_n\}$
7.	*(union)*	|	Q_1 `union` Q_2	✓	✓	✓	$\alpha(Q_1)$
8.	*(difference)*	|	Q_1 `minus` Q_2	✓	✓	✓	$\alpha(Q_1)$
9.	*(count aggregate)*	|	`cagg` A $\{A_1, \ldots, A_n\}$ Q	✓	✓	✓	$\{A_1, \ldots, A_n\} \cup \{A\}$
10.	*(sum aggregate)*	|	`sagg` A $\{A_1, \ldots, A_n\}$ A' Q	✓	✓	✓	$\{A_1, \ldots, A_n\} \cup \{A\}$
11.	*(sum of two attributes)*	|	`plus` A $\{A_1, A_2\}$ Q	✓	✓	✓	$\alpha(Q) \cup \{A\}$
12.	*(product of two attributes)*	|	`times` A $\{A_1, A_2\}$ Q	✓	✓	✓	$\alpha(Q) \cup \{A\}$
13.	*(explicit precedence)*	|	(Q)	✓	✓	✓	$\alpha(Q)$
14.	*(select)*	|	`select` $\{A_1, \ldots, A_n\}$ Q		✓	✓	$\{A_1, \ldots, A_n\}$
15.	*(union all)*	|	Q_1 `union all` Q_2		✓	✓	$\alpha(Q_1)$
16.	*(difference all)*	|	Q_1 `minus all` Q_2		✓	✓	$\alpha(Q_1)$
17.	*(intersect all)*	|	Q_1 `intersect all` Q_2		✓	✓	$\alpha(Q_1)$
18.	*(order by)*	|	Q_1 `order by` A `asc`|`desc`			✓	$\alpha(Q_1)$

Figure 2.3: Syntax of $\mathcal{WQ}$, $\mathcal{WBQ}$, and $\mathcal{WOBQ}$

1. *Any query of the form "* C `as` A *", "* $A_1.Pf_1 = A_2.Pf_2$ *", "* `empty` $\{A_1, \ldots, A_n\}$ *", and "* `true` *" occurs in* $\mathcal{WQ}$.

2. *If* Q, Q_1, *and* $Q_2 \in \mathcal{WQ}$, *then* $\mathcal{WQ}$ *also includes:*

   (a) `elim` $\{A_1, \ldots, A_n\}$ Q, if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$,

   (b) `from` Q_1, Q_2 ,

   (c) Q_1 `union` Q_2, if $\alpha(Q_1) = \alpha(Q_2)$,

   (d) Q_1 `minus` Q_2, if $\alpha(Q_1) = \alpha(Q_2)$,

   (e) `cagg` A $\{A_1, \ldots, A_n\}$ Q, if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$ and $Q \in \mathcal{FQ}$,

   (f) `sagg` A $\{A_1, \ldots, A_n\}$ A' Q, if $\{A_1, \ldots, A_n, A'\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$ and $Q \in \mathcal{FQ}$,

   (g) `plus` A $\{A_1, A_2\}$ Q, if $\{A_1, A_2\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$,

   (h) `times` A $\{A_1, A_2\}$ Q, if $\{A_1, A_2\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$.

Definition 2.2.2 *The set of finite queries, denoted $\mathcal{FQ}$, is the smallest set satisfying the following conditions:*

1. *Any query of the form "* C `as` A *", "* `empty` $\{A_1, \ldots, A_n\}$ *", and "* `true` *" occurs in $\mathcal{FQ}$.*

2. *If Q, Q_1, and $Q_2 \in \mathcal{FQ}$, and $Q_3 \in \mathcal{WQ}$, then $\mathcal{FQ}$ also includes:*

   (a) `elim` $\{A_1, \ldots, A_n\}$ Q, if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$,

   (b) *i.* `from` Q_1, Q_2,

   ii. `from` Q, Q_3, if $\alpha(Q_3) \subseteq \alpha(Q)$,

   iii. `from` $Q, (A_1.Pf = A_2.\texttt{Id})$, if $A_1 \in \alpha(Q)$,

   (c) Q_1 `union` Q_2, if $\alpha(Q_1) = \alpha(Q_2)$,

   (d) Q `minus` Q_3, if $\alpha(Q) = \alpha(Q_3)$,

   (e) `cagg` A $\{A_1, \ldots, A_n\}$ Q, if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$,

   (f) `sagg` A $\{A_1, \ldots, A_n\}$ A' Q, if $\{A_1, \ldots, A_n, A'\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$,

   (g) `plus` A $\{A_1, A_2\}$ Q, if $\{A_1, A_2\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$,

   (h) `times` A $\{A_1, A_2\}$ Q, if $\{A_1, A_2\} \subseteq \alpha(Q)$ and $A \notin \alpha(Q)$.

Proposition 2.2.3 $\mathcal{FQ} \subseteq \mathcal{WQ}$.

Proof: To prove that finite queries are also well-formed, we should show that any query of $\mathcal{FQ}$ also occurs in $\mathcal{WQ}$. We prove by induction, i.e., start with simple smaller cases and based on those gradually build up more complex cases.

Item *1* in Definition 2.2.2 shows that queries, "C `as` A", "`empty` $\{A_1, \ldots, A_n\}$", and "`true`" are finite queries. Accordingly, Item *1* in Definition 2.2.1 shows that these queries are also well-formed.

Item *2(a)* in Definition 2.2.2 shows that "`elim` $\{A_1, \ldots, A_n\}$ Q" is finite, if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$. This query is also in the set of well-formed queries with the same condition (Item *2(a)* of Definition 2.2.1).

Item *2(b)i.* in Definition 2.2.2 demonstrates that query "`from` Q_1, Q_2" is in $\mathcal{FQ}$ if Q_1 and Q_2 are finite. Q_1 and Q_2 are smaller finite queries and by induction those

are well-formed. Thus, based on Item *2(b)* in Definition 2.2.1, the query in Item *2(b)i.* of Definition 2.2.2 is well-formed.

Similarly, Item *2(b)ii.* demonstrates that query " `from` Q_3, Q " is in $\mathcal{FQ}$, when Q_3 is well-formed and Q is finite, and if $\alpha(Q_3)$ is contained in $\alpha(Q)$. Since Q is a smaller finite query, consequently by induction it is well-formed. Also Q_3 is well-formed, and consequently the query in Item *2(b)ii.* is well-formed. Similarly, Item *2(b)iii.* shows that " `from` $Q, (A_1.Pf = A_2.\texttt{Id})$ " is in $\mathcal{FQ}$ if $\{A_1\}$ is contained in $\alpha(Q)$ and Q is finite. Since $A_1.Pf = A_2.\texttt{Id}$ is well-formed and Q is a smaller finite query, and consequently by induction is well-formed, the query " `from` $Q, (A_1.Pf_1 = A_2.Pf_2)$ " is a form of query *2(b)* in Definition 2.2.1. Thus, the query of Item *2(b)iii.* is well-formed as well.

Item *2(c)* in Definition 2.2.2 shows that " Q_1 `union` Q_2 " is finite when Q_1 and Q_2 are finite and if $\alpha(Q_1) = \alpha(Q_2)$. Obviously the condition $\alpha(Q_1) = \alpha(Q_2)$ is the condition of Item *2(c)* in Definition 2.2.1. Also, Q_1 and Q_2 are smaller finite queries and therefore by induction are well-formed. So, the query in Item *2(c)* of Definition 2.2.2 is well-formed.

Item *2(d)* of Definition 2.2.2, " Q `minus` Q_3 ", is similar to Item *2(c)*, but Q_3 suffices to be well-formed. So, this query is well-formed as well based on Item *2(d)* of Definition 2.2.1.

Items *2(e)* and *2(f)* of Definitions 2.2.1 and 2.2.2 demonstrate that count and sum aggregate queries occur in both $\mathcal{WQ}$ and $\mathcal{FQ}$ with the same conditions.

Items *2(g)* and *2(h)* in Definition 2.2.2 show that queries " `plus` A $\{A_1, A_2\}$ Q " and " `times` A $\{A_1, A_2\}$ Q " are finite, if $\{A_1, A_2\}$ is contained in $\alpha(Q)$, and A is not in $\alpha(Q)$ and Q is finite. Items *2(g)* and *2(h)* in Definition 2.2.1 show that these two queries also occur in $\mathcal{WQ}$ since the only difference of conditions for these queries to be well-formed is that Q should be well-formed. As Q builds up from smaller finite queries, these queries are also well-formed.

So, any query of $\mathcal{FQ}$ also occurs in $\mathcal{WQ}$ and it implies that $\mathcal{FQ} \subseteq \mathcal{WQ}$.

□

Figure 2.4 shows the domain of $\mathcal{Q}$, well-formed $\mathcal{Q}$, and finite $\mathcal{Q}$ where finite $\mathcal{Q}$ is under well-formed $\mathcal{Q}$ and well-formed $\mathcal{Q}$ is under $\mathcal{Q}$.

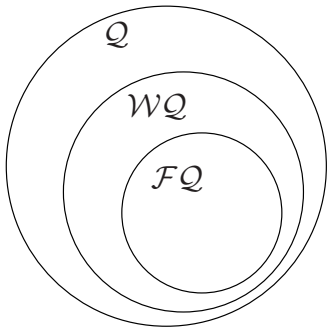

Figure 2.4: $\mathcal{Q}$, $\mathcal{WQ}$, AND $\mathcal{FQ}$ DOMAINS

2.2.2 Well-formed and Finite $\mathcal{BQ}$

In this subsection, we extend our definitions of well-formed and finite queries to bag queries.

Definition 2.2.4 *The set of well-formed bag queries, denoted $\mathcal{WBQ}$, is the smallest set satisfying the following conditions:*

1. *Any query of $\mathcal{WQ}$ also occurs in $\mathcal{WBQ}$*
2. *If Q_1 and $Q_2 \in \mathcal{WBQ}$ and $Q \in \mathcal{FBQ}$, then $\mathcal{FBQ}$ also includes:*
   - *(a)* `select` $\{A_1, \ldots, A_n\}$ Q, if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$,
   - *(b)* Q_1 `union all` Q_2, if $\alpha(Q_1) = \alpha(Q_2)$,
   - *(c)* Q_1 `minus all` Q_2, if $\alpha(Q_1) = \alpha(Q_2)$,
   - *(d)* Q_1 `intersect all` Q_2, if $\alpha(Q_1) = \alpha(Q_2)$.

Definition 2.2.5 *The set of finite bag queries, denoted $\mathcal{FBQ}$, is the smallest set satisfying the following conditions:*

1. *Any query of $\mathcal{FQ}$ also occurs in $\mathcal{FBQ}$*

2. If Q, Q_1, and $Q_2 \in \mathcal{FBQ}$, and $Q_3 \in \mathcal{WBQ}$, then $\mathcal{FBQ}$ also includes:

(a) $\texttt{select}\ \{A_1, \ldots, A_n\}\ Q$, if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$,

(b) $Q_1\ \texttt{union all}\ Q_2$, if $\alpha(Q_1) = \alpha(Q_2)$,

(c) $Q\ \texttt{minus all}\ Q_3$, if $\alpha(Q) = \alpha(Q_3)$,

(d) $Q\ \texttt{intersect all}\ Q_3$, if $\alpha(Q) = \alpha(Q_3)$.

Proposition 2.2.6 $\mathcal{FBQ} \subseteq \mathcal{WBQ}$

Proof: We should prove that any queries of $\mathcal{FBQ}$ occurs in $\mathcal{WBQ}$. Similar to Proposition 2.2.3, we prove this Proposition by induction.

From Definitions 2.2.4 and 2.2.5, it is straitforward that $\mathcal{WBQ}$ and $\mathcal{FBQ}$ have all queries of $\mathcal{WQ}$ and $\mathcal{FQ}$, respectively, as well as some other queries: `select`, `union all`, `minus all` and `intersect all`. Clearly, we need to prove only that `select`, `union all`, `minus all` and `intersect all` queries of $\mathcal{FBQ}$ also occur in $\mathcal{WBQ}$.

Query " $\texttt{select}\ \{A_1, \ldots, A_n\}\ Q$ " is in $\mathcal{FBQ}$ if $\{A_1, \ldots, A_n\} \subseteq \alpha(Q)$, and when Q is in $\mathcal{FBQ}$. Since Q is constructed from smaller finite cases and by induction is well-formed, query `select` of $\mathcal{FBQ}$ also occurs in $\mathcal{WBQ}$. Similarly, queries `union all` , `minus all` and `intersect all` are well-formed as well.

So, any query of $\mathcal{FBQ}$ also occurs in $\mathcal{WBQ}$ and it implies that $\mathcal{FBQ} \subseteq \mathcal{WBQ}$. □

Figure 2.5 shows the domain of $\mathcal{BQ}$, $\mathcal{WBQ}$, and $\mathcal{FBQ}$ where $\mathcal{FBQ}$ is under $\mathcal{WBQ}$ and $\mathcal{WBQ}$ is under $\mathcal{BQ}$.

2.2.3 Well-formed and Finite $\mathcal{OBQ}$

In this subsection, we extend our definitions of well-formed and finite queries to ordered bag queries.

Definition 2.2.7 *The set of well-formed ordered bag queries, denoted $\mathcal{WOBQ}$, is the smallest set satisfying the following conditions:*

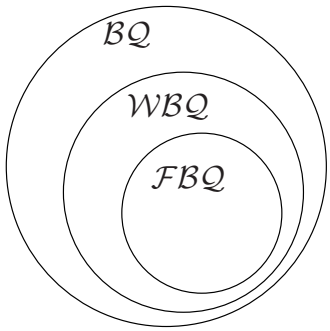

Figure 2.5: $\mathcal{BQ}$, $\mathcal{WBQ}$, AND $\mathcal{FBQ}$ DOMAINS

1. *Any query of $\mathcal{WBQ}$ also occurs in $\mathcal{WOBQ}$,*
2. *If $Q \in \mathcal{WOBQ}$, then $\mathcal{WOBQ}$ also includes:*
   (a) Q `order by` A `asc|desc`, *if* $A \in \alpha(Q)$.

Definition 2.2.8 *The set of finite ordered bag queries, denoted $\mathcal{FOBQ}$, is the smallest set satisfying the following conditions:*

1. *Any query of $\mathcal{FBQ}$ also occurs in $\mathcal{FOBQ}$,*
2. *If $Q \in \mathcal{FOBQ}$, then $\mathcal{FOBQ}$ also includes:*
   (a) Q `order by` A `asc|desc`, *if* $A \in \alpha(Q)$.

Proposition 2.2.9 $\mathcal{FOBQ} \subseteq \mathcal{WOBQ}$

Proof: From the Definitions 2.2.7 and 2.2.8, it is straitforward that $\mathcal{WOBQ}$ and $\mathcal{FOBQ}$ have all elements of $\mathcal{WBQ}$ and $\mathcal{FBQ}$, respectively, and a common element, that is, the order query. So, since from Proposition 2.2.6 $\mathcal{WBQ}$ includes $\mathcal{FBQ}$, it is straitforward that also $\mathcal{WOBQ}$ includes $\mathcal{FOBQ}$ and it implies that $\mathcal{FOBQ} \subseteq \mathcal{WOBQ}$. □

Proposition 2.2.10 $\mathcal{WQ} \subseteq \mathcal{WBQ} \subseteq \mathcal{WOBQ}$

Proof: From the Definitions 2.2.1, 2.2.4, and 2.2.7, it is straitforward that $\mathcal{WOBQ}$ includes all elements of $\mathcal{WBQ}$ and $\mathcal{WBQ}$, in turn, includes all of the elements of $\mathcal{WQ}$. □

Proposition 2.2.11 $\mathcal{FQ} \subseteq \mathcal{FBQ} \subseteq \mathcal{FOBQ}$

Proof: From Definitions 2.2.2, 2.2.5, and 2.2.8, it is straitforward that $\mathcal{FOBQ}$ includes all elements of $\mathcal{FBQ}$ and $\mathcal{FBQ}$, in turn, includes all elements of the $\mathcal{FQ}$. □

Figure 2.6 shows the domain of $\mathcal{WQ}$, $\mathcal{WBQ}$, and $\mathcal{WOBQ}$ where $\mathcal{WQ}$ is under $\mathcal{WBQ}$ and $\mathcal{WBQ}$ is under $\mathcal{WOBQ}$.

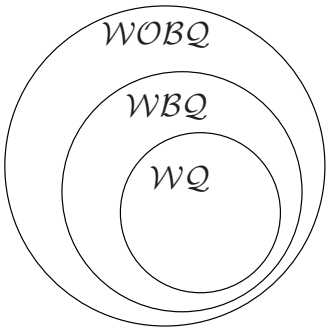

Figure 2.6: $\mathcal{WQ}$, $\mathcal{WBQ}$, AND $\mathcal{WOBQ}$

Figure 2.6 suggest that $\mathcal{WQ}$ is the smallest domain of well-formed queries. Well-formed queries are the smallest set of queries for which there is a semantics. From now on, we focus on well-formed queries, and we may occasionally refer to a well-formed query simply as a query. As we introduced earlier in this chapter, Figure 2.3 shows the syntax of three dialects, $\mathcal{WQ}$, $\mathcal{WBQ}$, and $\mathcal{WOBQ}$. Checkmarks show the productions that form the grammar for each of the dialects. In this work, our main contribution is to map the queries of $\mathcal{WBQ}$ and $\mathcal{WOBQ}$ to those of $\mathcal{WQ}$ in order to derive the semantics for queries of $\mathcal{WBQ}$ and $\mathcal{WOBQ}$ from the defined semantics of $\mathcal{WQ}$.

Syntactic Sugar

For the purpose of readability of queries, in this subsection we define some syntactic sugar:

Relational Algebra	$\mathcal{Q}$
R	Q
$\pi_{\{A_1,\ldots,A_n\}}(Q)$	$\texttt{elim}\ \{A_1,\ldots,A_n\}\ Q$
$\sigma_{A_1=A_2}(Q)$	$\texttt{from}\ Q, (A_1.\texttt{Id} = A_2.\texttt{Id})$
$Q_1 \times Q_2$	$\texttt{from}\ Q_1, Q_2$
$Q_1 \cup Q_2$	$Q_1\ \texttt{union}\ Q_2$
$Q_1 - Q_2$	$Q_1\ \texttt{minus}\ Q_2$
$\rho_{A_1 \to B_1}(Q)$	$\texttt{rename}\ A_1\ \texttt{as}\ B_1\ Q$

Figure 2.7: Relational Algebra Equivalent Expressions in $\mathcal{Q}$

1.	$\texttt{select} * Q$	$\equiv$	Q
2.	$\texttt{select distinct}\ A_1,\ldots,A_n Q$	$\equiv$	$\texttt{elim}\ \{A_1,\ldots,A_n\}\ Q$
3.	$Q_1\ \texttt{where}\ Q_2$	$\equiv$	$\texttt{from}\ Q_1, Q_2$
4.	$Q_1\ \texttt{and}\ Q_2$	$\equiv$	$\texttt{from}\ Q_1, Q_2$
5.	$\texttt{from}$	$\equiv$	$\texttt{true}$
6.	$\texttt{from}\ Q_1,\ldots,Q_n$	$\equiv$	$\texttt{from}\ (\texttt{from}\ Q_1, Q_2, \ldots), Q_n$
7.	$\texttt{rename}\ A_1\ \texttt{as}\ B_1\ Q$	$\equiv$	$\texttt{elim}\ \alpha(Q) \cup \{B_1\} - \{A_1\}\ (\texttt{from}\ Q, (B_1.\texttt{Id} = A_1.\texttt{Id}))$
8.	$\texttt{dom}\ A$	$\equiv$	$(A.\texttt{Id} = A.\texttt{Id})$
9.	$\texttt{plustab}\ \{A_1, A_2, A_3\}$	$\equiv$	$\texttt{plus}\ A_3\ \{A_1, A_2\}\ (\texttt{from dom}\ A_1, \texttt{dom}\ A_2)$
10.	$\texttt{lesstab}\ \{A_1, A_2\}$	$\equiv$	$\texttt{elim}\ \{A_1, A_2\}\ \texttt{plustab}\ \{A_1, A, A_2\}$
11.	$\texttt{eqtab}\ \{A_1, A_2\}$	$\equiv$	$\texttt{from dom}\ A_1, A_2.\texttt{Id} = A_1.\texttt{Id}$
12.	$\texttt{lesseqtab}\ \{A_1, A_2\}$	$\equiv$	$(\texttt{from lesstab}\ \{A_1, A_2\}\ )\ \texttt{union}\ (\texttt{eqtab}\ \{A_1, A_2\}\ )$
13.	$\texttt{minus}\ A_3\ \{A_1, A_2\}\ Q$	$\equiv$	$\texttt{from}\ Q, \texttt{plustab}\ \{A_3, A_2, A_1\}$
14.	$\texttt{mintab}\ \{A_1, A_2, A_3\}$	$\equiv$	$(\texttt{from eqtab}\ \{A_1, A_2\}\ A_3.\texttt{Id} = A_1.\texttt{Id})\texttt{union}$
			$(\texttt{from lesstab}\ \{A_1, A_2\}\ , A_3.\texttt{Id} = A1.\texttt{Id})\texttt{union}$
			$(\texttt{from lesstab}\ \{A_2, A_1\}\ , A_3.\texttt{Id} = A_2.\texttt{Id})$

The first few expressions on the left side of the equivalences are similar to SQL expressions. For instance, Rule 2 make it clear that `select distinct` expression is equivalent to `elim` in our syntax, i.e., selecting distinct values for attributes $A_1, \ldots, A_n$.

Example 2.2.1 *Figure 2.7 also shows common expressions of relational algebra such as selection, projection, cross product, minus, union, and renaming and their translation in our syntax.*

The first row of the table shows the query Q in our language which is correspondent to results of an arbitrary query R in relational databases. Assume results of query R

where $\alpha(R) = \{A_1, \ldots, A_n\}$*, tuples in the results of* R *can be referred to using the query* $(R$ `as` $A)$*:*

$Q \equiv (\texttt{elim}\ \{A_1, \ldots, A_n\}\ (\texttt{from}\ (R\ \texttt{as}\ A), (A.A_1.\texttt{Id} = A_1.\texttt{Id}, \ldots, A.A_n.\texttt{Id} = A_n.\texttt{Id}).$

Example 2.2.2 *Consider the database in Example 2.1.2. Suppose the query of deriving salaries of Employees who work in Location 3, ordered by salaries ascending. As the database instance in Example 2.1.2 implicitly shows that employees may work in several departments, but they have a single salary and also each department is located in a single place. The query should return the correct number of results of duplicates for the purpose of aggregate queries such as average (if there is a duplicate salary in the result that is from different employees). One possible query for this example, using some syntactic sugar rules, is as follows:*

```
Q ≡ select {Sal, P}
    from (select distinct {E, Sal, P}
          from EMP as E, WorksIn as W, DEPT as D
          where E.Eno = W.Eno
          and   W.Dno = D.Dno
          and   D.Loc = P
          and   E.Sal = Sal)
     order by Sal asc
```

P in the above query is an input parameter, which is used to parameterize `Loc` *attribute. For instance, in this example P is assigned to 3.*

This is an example of a query in which $\mathcal{WQ}$, $\mathcal{WBQ}$ and $\mathcal{WOBQ}$ expressions are combined. However, later in the next chapter, we see that the semantics for the arbitrary combined queries are derived from the basic $\mathcal{WQ}$ semantics.

Chapter 3

Semantics

In this chapter, we describe the semantics of $\mathcal{WQ}$, $\mathcal{WBQ}$, and $\mathcal{WOBQ}$, introduced in the last chapter. First, the semantics for $\mathcal{WQ}$ is declared which provides a background for defining the semantics for $\mathcal{WBQ}$ and $\mathcal{WOBQ}$. The semantics for two domains of $\mathcal{WBQ}$ and $\mathcal{WOBQ}$ are derived through two mapping functions $\mathcal{B}$ and $\mathcal{OB}$. As a matter of fact, these functions map the queries of $\mathcal{WBQ}$ and $\mathcal{WOBQ}$ to those of $\mathcal{WQ}$, which in turn provide the semantics for $\mathcal{WBQ}$ and $\mathcal{WOBQ}$ from that of $\mathcal{WQ}$.

$$
\begin{array}{rcl}
\alpha(t) & \equiv & \textit{set of attributes occurring in } t \\
t@A & \equiv & \textit{element } e \in \mathbb{N}, \textit{ such that ``A:e'' occurs in } t \\
 & & \textit{defined only when } A \in \alpha(t) \\
t[A_1, A_2, ..., A_n] & \equiv & \{A_1 : t@A_1, A_2 : t@A_2, \ldots, A_n : t@A_n\} \\
 & & \textit{defined only when } \{A_1, A_2, \ldots, A_n\} \subseteq \alpha(t)
\end{array}
$$

Figure 3.1: Operation on Tuples

3.1 Semantics of $\mathcal{WQ}$

This section demonstrates what each component of $\mathcal{WQ}$ means. Formally, we define a tuple t with attribute bindings for attributes $\{A_1, \ldots, A_n\}$ over the database $\mathcal{I}$ as the general form $\{A_1 : e_1, \ldots, A_n : e_n\}$ where $\{e_1, \ldots, e_n\} \subseteq \mathbb{N}$. For the purpose of operations on tuples, the operators α, @, and $t[\ ]$ have been defined (Figure 3.1). Intuitively, $\alpha(t)$ returns the schema of the tuple t. As Figure 3.1 shows, operator @

1. $(C \texttt{ as } A)^{\mathcal{I}} \equiv \{\ \{A : e\}\ \mid e \in (C)^{\mathcal{I}}\}$

2. $(A_1.Pf_1 = A_2.Pf_2)^{\mathcal{I}} \equiv \{\ \{A_1 : e_1, A_2 : e_2\}\ \mid e_1 \in \mathbb{N}, e_2 \in \mathbb{N} \wedge (Pf_1)^{\mathcal{I}}(e_1) = (Pf_2)^{\mathcal{I}}(e_2)\}$

3. $(\texttt{elim}\ \{A_1, \ldots, A_n\}\ Q)^{\mathcal{I}} \equiv \{\ t[A_1, \ldots, A_n] \mid t \in (Q)^{\mathcal{I}}\}$

4. $(\texttt{true})^{\mathcal{I}} \equiv \{\emptyset\}$

5. $(\texttt{from}\ \ Q_1, Q_2)^{\mathcal{I}} \equiv \{\ t \mid \alpha(t) = \alpha(Q_1) \cup \alpha(Q_2) \wedge \exists\, t_1 \in (Q_1)^{\mathcal{I}}, t_2 \in (Q_2)^{\mathcal{I}} :\ t[\alpha(t_1)] = t_1\ \wedge\ t[\alpha(t_2)] = t_2\}$

6. $(\texttt{empty}\ \ \{A_1, \ldots, A_n\}\ )^{\mathcal{I}} \equiv \emptyset$

7. $(Q_1 \texttt{ union }\ Q_2)^{\mathcal{I}} \equiv (Q_1)^{\mathcal{I}} \cup (Q_2)^{\mathcal{I}}$

8. $(Q_1 \texttt{ minus }\ Q_2)^{\mathcal{I}} \equiv (Q_1)^{\mathcal{I}} - (Q_2)^{\mathcal{I}}$

9. $(\texttt{cagg}\ A\ \{A_1, \ldots, A_n\}\ Q)^{\mathcal{I}} \equiv \{\ t\ \uplus\ \{A : c\} \mid t \in (\texttt{elim}\ \{A_1, \ldots, A_n\}\ \ Q)^{\mathcal{I}}\ \wedge\ \ c = |\ \ \{t' \mid t' \in (Q)^{\mathcal{I}} \wedge\ \bigwedge_{1 \le i \le n} t'@A_i = t@A_i\}\ \ |\ \}$

10. $(\texttt{sagg}\ A\ \{A_1, \ldots, A_n\}\ \ A'\ Q)^{\mathcal{I}} \equiv \{\ t\ \uplus\ \{A : s\}\ \mid t \in (\texttt{elim}\ \{A_1, \ldots, A_n\}\ \ Q)^{\mathcal{I}}\ \wedge\ \ s = \sum_{\substack{t' \in\ \{t'' \mid t'' \in (Q)^{\mathcal{I}} \wedge \\ \bigwedge_{1 \le i \le n} t''@A_i = t@A_i\}}} (t'@A')\ \}$

11. $(\texttt{plus}\ A\ \ \{A_1, A_2\}\ \ Q)^{\mathcal{I}} \equiv \{\ t\ \uplus\ \{A : (t@A_1 + t@A_2)\}\ \mid t \in (Q)^{\mathcal{I}}\}$

12. $(\texttt{times}\ A\ \ \{A_1, A_2\}\ \ Q)^{\mathcal{I}} \equiv \{\ t\ \uplus\ \{A : (t@A_1 \times t@A_2)\}\ \mid t \in (Q)^{\mathcal{I}}\}$

13. $((Q))^{\mathcal{I}} \equiv (Q)^{\mathcal{I}}$

Figure 3.2: Semantics of $\mathcal{WQ}$

in $t@A$ is defined only when A is an attribute in the schema of tuple t and returns the value of attribute A for the tuple t. Also Figure 3.1 shows that $t[A_1, \ldots, A_n]$ is defined only when $\{A_1, \ldots, A_n\}$ occurs in the schema of tuple t.

We generalize the definition of $\mathcal{I}$ for defining a set semantics for queries. The interpretation of query Q under set semantics, $(Q)^{\mathcal{I}}$, is a function which maps databases to sets of tuples (Figure 3.2). Informally, the meaning of each query is as follows:

Query 1, " C `as` A ", reference, creates a view with schema A for objects in the

interpretation of C i.e., $(C)^{\mathcal{I}}$.

Query 2, " $A_1.Pf_1 = A_2.Pf_2$ ", selection, returns tuple of the form $\{A_1 : e_1, A_2 : e_2\}$ where applying path function Pf_1 on e_1, and Pf_2 on e_2 leads to the same object.

Query 3, " `elim` $\{A_1, \ldots, A_n\}$ Q ", projection, projects out the attribute bindings of attributes $\{A_1, \ldots, A_n\}$ of tuples of Q.

Query 4, " `true` " is a singleton set consisting of the null tuple.

Query 5, " `from` Q_1, Q_2 ", has the property of cross product if $\alpha(Q_1) \cap \alpha(Q_2) = \emptyset$, and projection, otherwise.

Query 6, " `empty` $\{A_1, \ldots, A_n\}$ ", is an empty set, yet with schema $\{A_1, \ldots, A_n\}$.

Query 7 and 8, " Q_1 `union` Q_2 " and " Q_1 `minus` Q_2 ", are union and difference queries, respectively.

Query 9, count aggregate query, groups tuples of Q by attributes $\{A_1, \ldots, A_n\}$ and counts the number of existing tuples for each group. Similarly, in 10, sum aggregate query groups tuples of Q by attributes $\{A_1, \ldots, A_n\}$, sums up the values of attribute A' of all tuples in each group and names the accumulated value under the new attribute A.

Queries 11 and 12, aggregate queries `plus` and `times`, are used for adding and multiplying two attribute values and naming the result under another new attribute.

Query 13 shows the explicit precedence of query Q.

Queries 14 to 18 are queries of domains $\mathcal{BQ}$ and $\mathcal{OBQ}$ which will be discussed later.

Example 3.1.1 *Consider query* `dom` $A \equiv A.\texttt{Id} = A.\texttt{Id}$ *introduced in syntactic sugar in the last chapter. It creates query results with schema* $\{A\}$ *and infinite number of tuples (all natural numbers), i.e.,* $\{\{A : 1\}, \{A : 2\}, \ldots\}$.

Table 3.1: `plustab`, `lesstab`, `lesseqtab`, AND `mintab`

A1	A2	A3	A1	A2	A1	A2	A1	A2	A3
1	1	2	1	2	1	1	1	1	1
1	2	3	1	3	2	2	2	2	2
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
2	1	3	2	3	1	2	1	2	1
2	2	4	2	4	1	3	1	3	1
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

Example 3.1.2 *The query,* `from` A_1.`Id` = A_2.`Id`*, creates results with schema* $\{A_1, A_2\}$*, and infinite number of tuples, each of which has the same attribute bindings for attributes* A_1 *and* A_2*, i.e.,* $\{\{A_1 : 1, A_2 : 1\}, \{A_1 : 2, A_2 : 2\}, \ldots\}$.

Example 3.1.3 *Consider queries* `plustab` $\{A_1, A_2, A_3\}$ *and* `lesstab` $\{A_1, A_2\}$ *introduced in syntactic sugar.*

The results of these two queries can be thought of as relations of the form `plustab`(`A1`, `A2`, `A3`) *and* `lesstab`(`A1`, `A2`) *in the relational model. An instance of the relation* `plustab` *consists of a countably infinite number of tuples where all natural numbers occur in columns* `A1` *and* `A2`*, permutations of two natural numbers, and their summation in column* `A3` *(Table 3.1). Similarly, an instance of the relation* `lesstab` *consists of all pairs of natural numbers in which the first element is less than the second (Table 3.1).*

Table 3.1 also shows similar results for queries `lesseqtab` $\{A_1, A_2\}$ and `mintab` $\{A_1, A_2, A_3\}$.

Proposition 3.1.1 *The semantics of* $\mathcal{WQ}$ *is well-founded.*

Proof: The only real issue is to show that the semantics of queries 9 and 10 (count and sum aggregate) are well-founded.

To show that the semantics of Query 9, " (`cagg` A $\{A_1, \ldots, A_n\}$ $Q)^{\mathcal{I}}$ ", is well-founded, we should prove that constant c in the definition of the semantics for Query

9 is finite. c is the number of elements of the set " $\{t' \mid t' \in (Q)^{\mathcal{I}} \wedge \bigwedge_{1 \leq i \leq n} t'@A_i = t@A_i\}$ ". It is straitforward that the number of elements of the set is at most the number of elements of $(Q)^{\mathcal{I}}$. Q is finite by the definition of well-formed queries. So, c is finite.

To show that the semantics of Query 10, " $\texttt{sagg}\ A\ \{A_1, \ldots, A_n\}\ \ A'\ Q)^{\mathcal{I}}$ ", is well-founded, we should prove that constant s in the definition of the semantics for Query 10 is finite. s is equal to " $\sum_{\substack{t' \in \ \{t'' \mid t'' \in (Q)^{\mathcal{I}} \wedge \\ \wedge_{1 \leq i \leq n} t''@A_i = t@A_i\}}} (t'@A')$ ". It is straitforward that s is at most equal to " $\sum_{t' \in \ \{t'' \mid t'' \in (Q)^{\mathcal{I}}\}} (t'@A')$ ". Since Q is finite by the definition of well-formed queries, s is finite. $\square$

3.2 Semantics of $\mathcal{WBQ}$

The difference between a bag and a set query is that the interpretation of queries under bag semantics, preserves duplicates; whereas, the interpretation of queries under set semantics ignores duplicates. In this section, we extend our semantics for bag algebra. As we saw in the last chapter, the syntax for bag algebra consists of all queries in the set algebra syntax, as well as a few more queries useful for work on duplicates (Figure 2.3). These queries are `union all`, `minus all`, `intersect all`, and duplicate preserving `select`. We define a function, $\mathcal{B}$, $\mathcal{B} : \mathcal{WBQ} \rightarrow \mathcal{WQ}$ which maps a bag algebra query to one of an equivalent set algebra. By this method, the semantics for the queries in $\mathcal{WBQ}$ is derived by means of the semantics of $\mathcal{WQ}$. The mapping function $\mathcal{B}$ preserves the number of tuples of the results of queries in the auxiliary attribute, `Cnt`.

Figure 3.3 shows outputs of the mapping function $\mathcal{B}$ on each expression of $\mathcal{WBQ}$. For instance, consider the result of applying the mapping function $\mathcal{B}$ on " C `as` A ". As the figure shows we will have " `from` $(C$ `as` $A), \mathcal{B}($`true`$)$ ". To derive the semantics for this result, we use the defined semantics in Figure 3.2 for set algebra, therefore:

1. $\mathcal{B}(C \texttt{ as } A) = \texttt{from } (C \texttt{ as } A), \mathcal{B}(\texttt{true})$

2. $\mathcal{B}(A_1.Pf_1 = A_2.Pf_2) = \texttt{from } (A_1.Pf_1 = A_2.Pf_2), \mathcal{B}(\texttt{true})$

3. $\mathcal{B}(\texttt{elim } \{A_1, \ldots, A_n\}\ Q) = \texttt{from } (\texttt{elim } \{A_1, \ldots, A_n\}\ \mathcal{B}(Q)), \mathcal{B}(\texttt{true})$

4. $\mathcal{B}(\texttt{true}) = \texttt{cagg Cnt } \{\} \ (\texttt{true})$

5. $\mathcal{B}(\texttt{from } Q_1, Q_2) =$
$\texttt{elim } \alpha(Q_1) \cup \alpha(Q_2) \cup \{\texttt{Cnt}\}$
$(\texttt{times Cnt } \{\texttt{Cnt1}, \texttt{Cnt2}\}$
$(\texttt{from } (\texttt{rename Cnt as Cnt1 } \mathcal{B}(Q_1)),$
$\quad (\texttt{rename Cnt as Cnt2 } \mathcal{B}(Q_2))))$

6. $\mathcal{B}(\texttt{empty } \{A_1, \ldots, A_n\}\ ) = \texttt{empty } \{A_1, \ldots, A_n, \texttt{Cnt}\}$

7. $\mathcal{B}(Q_1 \texttt{ union } Q_2) = \texttt{from } ((\texttt{elim } \alpha(Q_1)\ \mathcal{B}(Q_1)) \texttt{ union } (\texttt{elim } \alpha(Q_2)\ \mathcal{B}(Q_2))), \mathcal{B}(\texttt{true})$

8. $\mathcal{B}(Q_1 \texttt{ minus } Q_2) = \texttt{from } ((\texttt{elim } \alpha(Q_1)\ \mathcal{B}(Q_1)) \texttt{ minus } (\texttt{elim } \alpha(Q_2)\ \mathcal{B}(Q_2))), \mathcal{B}(\texttt{true})$

9. $\mathcal{B}(\texttt{cagg } A\ \{A_1, \ldots, A_n\}\ Q) = \texttt{from } (\texttt{sagg } A\ \{A_1, \ldots, A_n\}\ \texttt{Cnt } \mathcal{B}(Q)), \mathcal{B}(\texttt{true})$

10. $\mathcal{B}(\texttt{sagg } A\ \{A_1, \ldots, A_n\}\ A'\ Q) = \texttt{from } (\texttt{sagg } A\ \{A_1, \ldots, A_n\}\ A''\ (\texttt{times } A''\ \{A', \texttt{Cnt}\}\ \mathcal{B}(Q))), \mathcal{B}(\texttt{true})$
$\textit{where } A'' \notin \alpha(\mathcal{B}(Q))$

11. $\mathcal{B}(\texttt{plus } A\ \{A_1, A_2\}\ Q) = \texttt{plus } A\ \{A_1, A_2\}\ \mathcal{B}(Q)$

12. $\mathcal{B}(\texttt{times } A\ \{A_1, A_2\}\ Q) = \texttt{times } A\ \{A_1, A_2\}\ \mathcal{B}(Q)$

13. $\mathcal{B}((Q)) = \mathcal{B}(Q)$

14. $\mathcal{B}(\texttt{select } \{A_1, \ldots, A_n\}\ Q) =$
$\texttt{rename Cnt1 as Cnt}$
$\texttt{elim } \{A_1, \ldots, A_n, \texttt{Cnt1}\}\ (\texttt{sagg Cnt1 } \{A_1, \ldots, A_n\}\ \texttt{Cnt } \mathcal{B}(Q))$

15. $\mathcal{B}(Q_1 \texttt{ union all } Q_2) =$
$\texttt{from } (\texttt{elim } \alpha(Q_1)\ \mathcal{B}(Q_1 \texttt{ minus } Q_2)), \mathcal{B}(Q_1)$
$\texttt{union}$
$\texttt{from } (\texttt{elim } \alpha(Q_2)\ \mathcal{B}(Q_2 \texttt{ minus } Q_1)), \mathcal{B}(Q_2)$
$\texttt{union}$
$(\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}$
$\texttt{plus Cnt } \{\texttt{Cnt1}, \texttt{Cnt2}\}$
$\texttt{from } (\texttt{rename Cnt1 as Cnt } \mathcal{B}(Q_1)), (\texttt{rename Cnt2 as Cnt } \mathcal{B}(Q_2))$

16. $\mathcal{B}(Q_1 \texttt{ minus all } Q_2) =$
$\texttt{from } (\texttt{elim } \alpha(Q_1)\ \mathcal{B}(Q_1 \texttt{ minus } Q_2)),\ \mathcal{B}(Q_1)$
$\texttt{union}$
$\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}$
$(\texttt{minus Cnt } \{\texttt{Cnt1}, \texttt{Cnt2}\} \texttt{ from}$
$(\texttt{rename Cnt as Cnt1 } \mathcal{B}(Q_1)), (\texttt{rename Cnt as Cnt2 } \mathcal{B}(Q_2)))$

17. $\mathcal{B}(Q_1 \texttt{ intersect all } Q_2) =$
$\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}$
$(\texttt{mintab } \{\texttt{Cnt1}, \texttt{Cnt2}, \texttt{Cnt}\} \texttt{ from}$
$(\texttt{rename Cnt as Cnt1 } \mathcal{B}(Q_1)), (\texttt{rename Cnt as Cnt2 } \mathcal{B}(Q_2)))$

Figure 3.3: Semantics of $\mathcal{WBQ}$

$$
\begin{aligned}
(\texttt{from}\ (C\ \texttt{as}\ A), \mathcal{B}(\texttt{true}))^{\mathcal{I}} \quad \equiv \quad & \{\ t \mid \alpha(t) = \alpha(C\ \texttt{as}\ A) \cup \alpha(\mathcal{B}(\texttt{true}))\ \wedge \\
& \quad \exists\ t_1 \in (C\ \texttt{as}\ A)^{\mathcal{I}}, t_2 \in (\mathcal{B}(\texttt{true}))^{\mathcal{I}} : \\
& \quad t[\alpha(t_1)] = t_1\ \wedge\ t[\alpha(t_2)] = t_2 \\
& \}
\end{aligned}
$$

which is equal to:

$$
\begin{aligned}
(\texttt{from}\ (C\ \texttt{as}\ A), \mathcal{B}(\texttt{true}))^{\mathcal{I}} \quad \equiv \quad & \{\ t \mid \alpha(t) =\ \{A\}\ \cup\ \{\texttt{Cnt}\}\ \wedge \\
& \quad \exists\ t_1 \in (C\ \texttt{as}\ A)^{\mathcal{I}}, t_2 \in (\mathcal{B}(\texttt{true}))^{\mathcal{I}} : \\
& \quad t[A] = t_1\ \wedge\ t[\texttt{Cnt}] = t_2 \\
& \}
\end{aligned}
\tag{3.1}
$$

Example 3.2.1 *Suppose the interpretation of concept C under database $\mathcal{I}$ is $\{2, 4, 6\}$. The interpretation of $(C\ \texttt{as}\ A)$ under set semantics would be $\{\ \{A:2\}\ ,\ \{A:4\}\ ,\ \{A:6\}\ \}$. To derive the interpretation of $(C\ \texttt{as}\ A)$ in bag semantics, the interpretation function $()^{\mathcal{I}}$ is applied to mapping function $\mathcal{B}$ on $(C\ \texttt{as}\ A)$. From Equation 3.1 results can be derived which is as follows: $\{\ \{A:2, \texttt{Cnt}:1\}\ ,\ \{A:4, \texttt{Cnt}:1\}\ ,\ \{A:6, \texttt{Cnt}:1\}\ \}$.*

In the following, we informally describe the output of mapping function $\mathcal{B}$ for each query of $\mathcal{WBQ}$ of Figure 2.3.

The outputs of $\mathcal{B}$ in queries 2 and 3 are similar to that of 1 described above.

The output of query 4 is $\{\ \{\texttt{Cnt}:1\}\ \}$.

In query 5, `Cnt` of $\mathcal{B}(Q_1)$ and $\mathcal{B}(Q_2)$ are renamed `Cnt1` and `Cnt2` respectively. Then, for each tuple, the results of the multiplication of `Cnt1` and `Cnt2` are stored under attribute `Cnt`. Finally, the desired results, projecting $\alpha(Q_1)$, $\alpha(Q_2)$, and `Cnt` out.

The output of query 6 is the empty set, yet with schema $\{A_1, \ldots, A_n, \texttt{Cnt}\}$.

$\mathcal{B}$ acts similarly for queries 7 and 8. For instance, in 7, first `Cnt`(s) of $\mathcal{B}(Q_1)$ and

$\mathcal{B}(Q_2)$ are dropped and union of the remained is taken; the results are grouped by $\alpha(Q_1)$, or equivalently $\alpha(Q_2)$, and lastly their cross product with $\mathcal{B}(\texttt{true})$ is taken which adds count of 1 to each tuple.

$\mathcal{B}$ acts similarly for queries 9 and 10 as well. For instance, in query 9, the sum aggregate of attribute `Cnt` for each group $\{A_1, \ldots, A_n\}$ in $\mathcal{B}(Q)$ is taken and named under the attribute A and cross product of the results from last step with $\mathcal{B}(\texttt{true})$ is taken.

In queries 11 and 12, the sum and product of attributes A_1 and A_2 in $\mathcal{B}(Q)$ are taken, respectively, and the results are stored under attribute A.

Query 13, explicit precedence which is straitforward.

Query 14 is the most important expression in $\mathcal{WBQ}$. `select` query returns tuples with the schema of $\{A_1, \ldots, A_n\}$ in Q together with their counts. To do so, the sum aggregate of `Cnt` for each group of $\{A_1, \ldots, A_n\}$ in $\mathcal{B}(Q)$ should be taken and named under attribute `Cnt`. However, in order to write a well-formed definition for Query 13, first the sum aggregate of `Cnt` for each group of $\{A_1, \ldots, A_n\}$ in $\mathcal{B}(Q)$ is taken and named under attribute `Cnt1`. After projecting $\{A_1, \ldots, A_n\} \cup \{\texttt{Cnt1}\}$ out, `Cnt1` is renamed `Cnt`.

$\mathcal{B}$ also acts similarly for queries 15, 16 and 17. For instance, in query 15, union all is taken in three steps. In step one and two, tuples which occur in just one of Q_1 or Q_2 are taken together with their counts. In step three, the tuples which occur in both Q_1 and Q_2 are taken, their counts summed together, and their union with results of step one and two is taken.

Proposition 3.2.1 *For all* $Q \in \mathcal{WBQ}$,
1) $\{\texttt{Cnt}\} \notin \alpha(Q)$, *and*
2) $\alpha(\mathcal{B}(Q)) = \alpha(Q) \ \cup \ \{\texttt{Cnt}\}$.

Proof: For the first part, by the convention stated in the last chapter, the reserved attribute `Cnt` does not occur in the schema of set queries. For the second part, it is obvious that `Cnt` is in schema query 6. Also, as Figure 3.3 shows, query 4 produces $\{\texttt{Cnt} : 1\}$ which has the schema $\{\texttt{Cnt}\}$. Moreover, in the definition of queries 1, 2, 3,

7, 8, 9, and 10, the cross product of the queries with query 4 is taken and consequently `Cnt` occurs in their schema. Queries 5, 11, 12, 13, 14, 15, 16, and 17 are constructed from small cases consists of $\mathcal{B}(Q)$ and by induction have `Cnt` in their schema. □

3.3 Semantics of $\mathcal{WOBQ}$

In this section, we introduce the mapping function $\mathcal{OB}$, which maps the domain $\mathcal{WOBQ}$ to $\mathcal{WQ}$. Tuples of the domain $\mathcal{WOBQ}$ have two additional reserved attributes, `Cnt` and `Ord`, for the sake of preserving the number of duplicates and order of attributes, respectively. As we saw in the last chapter, the main expression of $\mathcal{WOBQ}$ is "Q `order by` A `asc` | `desc`". This expression orders tuples of Q by attribute A either ascending or descending, maintaining ordering in `Ord` attribute. The mapping function $\mathcal{OB}$, $\mathcal{OB} : \mathcal{WOBQ} \rightarrow \mathcal{WQ}$, allows us to derive the semantics for the queries in $\mathcal{WOBQ}$ using the semantics of $\mathcal{WQ}$.

Figures 3.4 shows the definition of function $\mathcal{OB}$ for all query inputs other than `order by` and `select`. We describe details of $\mathcal{OB}$ on these two queries in the following subsections.

For instance, consider the result of applying the mapping function $\mathcal{OB}$ on " C `as` A ". As Figure 3.4 shows, we have "`from` $(C$ `as` $A), \mathcal{OB}($`true`$)$". To derive the semantics for this result, we use the defined semantics for set algebra as follows:

$$
\begin{aligned}
\texttt{from}\ (C\ \texttt{as}\ A), \mathcal{OB}(\texttt{true}) \quad \equiv \quad & \{\ t \mid \alpha(t) = \alpha(C\ \texttt{as}\ A) \cup \alpha(\mathcal{OB}(\texttt{true})) \wedge \\
& \quad \exists\ t_1 \in (C\ \texttt{as}\ A)^{\mathcal{I}}, t_2 \in (\mathcal{OB}(\texttt{true}))^{\mathcal{I}} : \\
& \quad t[\alpha(t_1)] = t_1\ \wedge\ t[\alpha(t_2)] = t_2 \\
& \}
\end{aligned}
$$

which is equal to:

$$
\begin{aligned}
(\texttt{from}\ (C\ \texttt{as}\ A), \mathcal{OB}(\texttt{true}))^{\mathcal{I}} \ \equiv \ & \{\ t \mid \alpha(t) = \ \{A\} \ \cup \ \{\texttt{Cnt}, \texttt{Ord}\} \wedge \\
& \quad \exists\ t_1 \in (C\ \texttt{as}\ A)^{\mathcal{I}}, t_2 \in (\mathcal{OB}(\texttt{true}))^{\mathcal{I}} : \\
& \quad\ t[A] = t_1\ \wedge\ t[\texttt{Cnt}] = t_2 \\
& \}
\end{aligned}
\tag{3.2}
$$

Example 3.3.1 *Consider concept C from Example 3.2.1. As Example 3.2.1 shows, the interpretation of "C as A" under set and bag semantics are* $\{\ \{A:2\},\ \{A:4\},\ \{A:6\}\ \}$, *and* $\{\ \{A:2, \texttt{Cnt}:1\},\ \{A:4, \texttt{Cnt}:1\},\ \{A:6, \texttt{Cnt}:1\}\ \}$, *respectively. To derive the interpretation of "C as A" under ordered bag semantics, from Equation 3.3, we will have:*
$\{\ \{A:2, \texttt{Cnt}:1, \texttt{Ord}:1\},\ \{A:4, \texttt{Cnt}:1, \texttt{Ord}:1\},\ \{A:6, \texttt{Cnt}:1, \texttt{Ord}:1\}\ \}$.

As the above example demonstrates, the order of tuples are maintained in the auxiliary attribute `Ord`. The example also reveals that the ordering of the tuples for the reference query is destructive. As a matter of fact, ordering is stable for the sum and product of two attributes (`plus` and `times` queries), as well as `select` and `order by` queries, however, it is destructive for all other queries. In the following, we describe the output of mapping function $\mathcal{OB}$ for each query of $\mathcal{WOBQ}$ of Figure 2.3.

The definition of $\mathcal{OB}$ in queries 2 and 3 are similar to that of query 1 explained above.

The output of query 4 would be $\{\ \{\texttt{Cnt}:1, \texttt{Ord}:1\}\ \}$.

$\mathcal{OB}$'s behavior on query 5 is similar to that of $\mathcal{B}$ on query 5 (Figure 3.3), however, it takes an additional step. That is cross product of the results with "(`cagg Ord {} true`)" which adds $\{\texttt{Ord}:1\}$ to tuples.

The output of query 6 is the empty set, yet with schema $\{A_1, \ldots, A_n, \texttt{Cnt}, \texttt{Ord}\}$.

$\mathcal{OB}$, like $\mathcal{B}$, behaves similarly for queries 7 and 8, however, it takes the cross product of the intermediate results with $\mathcal{OB}(\texttt{true})$.

$\mathcal{OB}$, like $\mathcal{B}$, acts similarly for queries 9 and 10, however, it takes the cross product of the intermediate results with $\mathcal{OB}$(`true`).

In queries 11 and 12, the sum and product of attributes A_1 and A_2 in $\mathcal{OB}(Q)$ are taken, respectively, and the results are stored under attribute A.

`select` is one of the two most important queries in $\mathcal{WOBQ}$. The output of $\mathcal{OB}$ on `select` is described in detail in Subsections 3.3.3.

$\mathcal{OB}$'s behavior on queries 15, 16 and 17 is similar to mapping function $\mathcal{B}$ on these queries. However, the definition of $\mathcal{OB}$ for queries 15 and 16, first drops `Ord` attribute generated in intermediate results and, at the end , the highest operation adds the `Ord` attribute with the value of 1 to the final results.

Query `order by` is the other important query in $\mathcal{WOBQ}$. The output of $\mathcal{OB}$ on `order by` is described in Subsections 3.3.1 and 3.3.2.

1.	$\mathcal{OB}(C \texttt{ as } A)$	=	$\texttt{from } (C \texttt{ as } A), \mathcal{OB}(\texttt{true})$
2.	$\mathcal{OB}(A_1.Pf_1 = A_2.Pf_2)$	=	$\texttt{from } (A_1.Pf_1 = A_2.Pf_2), \mathcal{OB}(\texttt{true})$
3.	$\mathcal{OB}(\texttt{elim } \{A_1, \ldots, A_n\}\ Q)$	=	$\texttt{from } (\texttt{elim } \{A_1, \ldots, A_n\}\ \mathcal{OB}(Q)), \mathcal{OB}(\texttt{true})$
4.	$\mathcal{OB}(\texttt{true})$	=	$\texttt{cagg Ord } \{\texttt{Cnt}\}\ (\texttt{cagg Cnt } \{\}\ (\texttt{true}))$
5.	$\mathcal{OB}(\texttt{from } Q_1, Q_2)$	=	$\texttt{from } (\texttt{cagg Ord } \{\}\ \texttt{true}),$ $\texttt{elim } \alpha(Q_1) \cup \alpha(Q_2) \cup \{\texttt{Cnt}\}$ $(\texttt{times Cnt } \{\texttt{Cnt1}, \texttt{Cnt2}\}$ $(\texttt{from } (\texttt{rename Cnt as Cnt1 } \mathcal{OB}(Q_1)),$ $\quad (\texttt{rename Cnt as Cnt2 } \mathcal{OB}(Q_2))))$
6.	$\mathcal{OB}(\texttt{empty } \{A_1, \ldots, A_n\}\ )$	=	$\texttt{empty } \{A_1, \ldots, A_n, \texttt{Cnt}, \texttt{Ord}\}$
7.	$\mathcal{OB}(Q_1 \texttt{ union } Q_2)$	=	$\texttt{from } ((\texttt{elim } \alpha(Q_1)\ \mathcal{OB}(Q_1)) \texttt{ union } (\texttt{elim } \alpha(Q_2)\ \mathcal{OB}(Q_2))), \mathcal{OB}(\texttt{true})$
8.	$\mathcal{OB}(Q_1 \texttt{ minus } Q_2)$	=	$\texttt{from } ((\texttt{elim } \alpha(Q_1)\ \mathcal{OB}(Q_1)) \texttt{ minus } (\texttt{elim } \alpha(Q_2)\ \mathcal{OB}(Q_2))), \mathcal{OB}(\texttt{true})$
9.	$\mathcal{OB}(\texttt{cagg } A\ \{A_1, \ldots, A_n\}\ Q)$	=	$\texttt{from } (\texttt{sagg } A\ \{A_1, \ldots, A_n\}\ \texttt{Cnt}\ \mathcal{OB}(Q)), \mathcal{OB}(\texttt{true})$
10.	$\mathcal{OB}(\texttt{sagg } A\ \{A_1, \ldots, A_n\}\ A'\ Q)$	=	$\texttt{from } (\texttt{sagg } A\ \{A_1, \ldots, A_n\}\ A''\ (\texttt{times } A''\ \{A', \texttt{Cnt}\}\ \mathcal{OB}(Q))), \mathcal{OB}(\texttt{true})$ $where\ A'' \notin \alpha(\mathcal{OB}(Q))$
11.	$\mathcal{OB}(\texttt{plus } A\ \{A_1, A_2\}\ Q)$	=	$\texttt{plus } A\ \{A_1, A_2\}\ (\mathcal{OB}(Q))$
12.	$\mathcal{OB}(\texttt{times } A\ \{A_1, A_2\}\ Q)$	=	$\texttt{times } A\ \{A_1, A_2\}\ (\mathcal{OB}(Q))$
13.	$\mathcal{OB}((Q))$	$\equiv$	$\mathcal{OB}(Q)$
15.	$\mathcal{OB}(Q_1 \texttt{ union all } Q_2)$	=	$\texttt{from } (\texttt{cagg Ord } \{\}\ \texttt{true}),$ $\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}$ $(\texttt{from } (\texttt{elim } \alpha(Q_1)\ \mathcal{OB}(Q_1 \texttt{ minus } Q_2)), \mathcal{OB}(Q_1)$ $\texttt{union}$ $\texttt{from } (\texttt{elim } \alpha(Q_2)\ \mathcal{OB}(Q_2 \texttt{ minus } Q_1)), \mathcal{OB}(Q_2))$ $\texttt{union}$ $(\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}$ $\texttt{plus Cnt } \{\texttt{Cnt1}, \texttt{Cnt2}\}$ $\texttt{from } (\texttt{rename Cnt1 as Cnt } \mathcal{OB}(Q_1)), (\texttt{rename Cnt2 as Cnt } \mathcal{OB}(Q_2))$
16.	$\mathcal{OB}(Q_1 \texttt{ minus all } Q_2)$	=	$\texttt{from } (\texttt{cagg Ord } \{\}\ \texttt{true}),$ $\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}\ \texttt{from}$ $(\texttt{elim } \alpha(Q_1)\ \mathcal{OB}(Q_1 \texttt{ minus } Q_2)),\ \mathcal{OB}(Q_1)$ $\texttt{union}$ $\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}$ $(\texttt{minus Cnt } \{\texttt{Cnt1}, \texttt{Cnt2}\}\ \texttt{from}$ $(\texttt{rename Cnt as Cnt1 } \mathcal{OB}(Q_1)), (\texttt{rename Cnt as Cnt2 } \mathcal{OB}(Q_2)))$
17.	$\mathcal{OB}(Q_1 \texttt{ intersect all } Q_2)$	=	$\texttt{from } (\texttt{cagg Ord } \{\}\ \texttt{true}),$ $\texttt{elim } \alpha(Q_1) \cup \{\texttt{Cnt}\}$ $(\texttt{mintab } \{\texttt{Cnt1}, \texttt{Cnt2}, \texttt{Cnt}\}\ \texttt{from}$ $(\texttt{rename Cnt as Cnt1 } \mathcal{OB}(Q_1)), (\texttt{rename Cnt as Cnt2 } \mathcal{OB}(Q_2)))$

Figure 3.4: SEMANTICS OF $\mathcal{WOBQ}$

Table 3.2: ORDER BY EXAMPLE, $\mathcal{OB}(Q)$

A	A_2	Cnt	Ord
1	1	2	1
1	2	2	2
2	1	2	1
2	2	2	2

Table 3.3: RESULTS OF $\mathcal{OB}(Q$ order by $\{A\}$ asc$)$

A	A_2	Cnt	Ord
1	1	2	1
1	2	2	2
2	1	2	3
2	2	2	4

3.3.1 Translation of Q order by A asc

In this subsection, we show in detail the ordered bag translation of query order by, $\mathcal{OB}(Q$ order by A asc$)$. To clearly illustrate this translation, we give an example and apply it in progressive subexpressions of $\mathcal{OB}$ on order by .

Example 3.3.2 *Assume database $\mathcal{I}$ interprets the concept C as $\{1, 2\}$. Now, consider the following query:*

$$Q \equiv ((\texttt{from}\ C\ \texttt{as}\ A, C\ \texttt{as}\ A_2)\ \texttt{union all}\ (\texttt{from}\ C\ \texttt{as}\ A, C\ \texttt{as}\ A_2))\ \texttt{order by}\ A_2\ \texttt{asc}$$

It turns out that $(\mathcal{OB}(Q))^{\mathcal{I}}$ computes the results shown in Table 3.2. As we define $\mathcal{OB}(Q$ order by A asc$)$, we illustrate the definitions by incrementally applying the subexpressions that progressively define $\mathcal{OB}(Q$ order by A asc$)$ on query Q of this example. The desired final results are shown in Table 3.3.

Query "Q order by A asc" performs an ascending ordering on results of query Q. We see that our definition of "Q order by A asc" accomplishes a stable sort on Q by a major sort on attribute A and a minor sort on attribute Ord inherited to query Q, i.e., whenever attribute A values of two objects are the same, they are sorted based on their Ord values. The behavior of function $\mathcal{OB}(Q$ order by A asc$)$ can be described in six phases below. In our definitions, we assume temporary variables B, *CM*, and *NOrd* do not occur in the schema of Q.

Phase 1:

In phase 1, $\mathcal{OB}$ groups Q by attributes A and for each group (with a distinct value for attribute A, say c), the number of A values of the base table which are less than

Table 3.4: $Q4$

A	C
1	1
2	2

Table 3.5: $Q9$

M
2

Table 3.6: $Q10$

A	CM
1	2
2	4

or equal to c is counted and named under attribute C. This can be derived by the following four progressive subexpressions of $\mathcal{OB}$ function:

$$
\begin{aligned}
Q1 &\equiv \texttt{elim}\ \{A\}\ \mathcal{OB}(Q) \\
Q2 &\equiv \texttt{rename}\ A\ \texttt{as}\ B\ Q1 \\
Q3 &\equiv \texttt{from}\ Q1, Q2, \texttt{lesseqtab}\ \{B, A\} \\
Q4 &\equiv \texttt{cagg}\ C\ \{A\}\ Q3
\end{aligned}
$$

Applying these four subexpressions on query Q in Example 3.2 reveals the results shown in Table 3.4.

Phase 2:

In phase 2, the maximum `Ord` value of the base table (Table3.2) is found via the following five subexpressions:

$$
\begin{aligned}
Q5 &\equiv \texttt{elim}\ \{\texttt{Ord}\}\ \mathcal{OB}(Q) \\
Q6 &\equiv \texttt{rename}\ \texttt{Ord}\ \texttt{as}\ B\ Q5 \\
Q7 &\equiv \texttt{from}\ Q5, Q6, \texttt{lesstab}\ \{\texttt{Ord}, B\} \\
Q8 &\equiv Q5\ \texttt{minus}\ (\texttt{elim}\ \{\texttt{Ord}\}\ Q7) \\
Q9 &\equiv \texttt{rename}\ \texttt{Ord}\ \texttt{as}\ M\ Q8
\end{aligned}
$$

Applying these five subexpressions incrementally on results derived from the last phase turns out results shown in Table 3.5.

Phase 3:

In phase 3, the maximum `Ord` value derived in the last phase (M) is multiplied by C value of each group in $Q4$ and named under attribute CM:

Table 3.7: $Q11$

A	A_2	`Cnt`	`Ord`	CM	$NOrd$
1	1	2	1	2	3
1	2	2	2	2	4
2	1	2	1	4	5
2	2	2	2	4	6

Table 3.8: $Q12$

A	A_2	`Cnt`	$NOrd$
1	1	2	3
1	2	2	4
2	1	2	5
2	2	2	6

$$Q10 \equiv \texttt{elim}\ \{A, CM\}\ (\texttt{times}\ CM\ \{C, M\}\ (\texttt{from}\ Q4, Q9))$$

This phase opens up enough room between major sort values for the purpose of adding minor sort values in the following phase. Applying subexpression $Q10$ to the results in Table 3.5 reveals the results shown in Table 3.6.

Phase 4:

In phase 4, a new ordering is derived by adding major sort attribute values (CM) and minor sort attribute values (`Ord`):

$$Q11 \equiv \texttt{plus}\ NOrd\ \{\texttt{Ord}, CM\}\ (\texttt{from}\ \mathcal{OB}(Q), Q10)$$

Applying this subexpression to the results in Table 3.6 reveals the results shown in Table 3.7.

Phase 5:

In phase 5, interesting attributes are retained:

$$Q12 \equiv \texttt{elim}\ \alpha(Q)\ \cup\ \{\texttt{Cnt}, NOrd\}\ Q11$$

Applying this subexpression to the results in Table 3.7 reveals the results shown in Table 3.8.

Phase 6:

In the last phase, the new *ord* values are normalized so that results have orderings which start from 1 and successively increase to the maximum value. This is done using the following five subexpressions:

Table 3.9: $Q17$

A	A_2	Cnt	Ord
1	1	2	1
1	2	2	2
2	1	2	3
2	2	2	4

$$
\begin{aligned}
Q13 &\equiv \texttt{elim}\ \{NOrd\}\ Q12 \\
Q14 &\equiv \texttt{rename}\ NOrd\ \texttt{as}\ B\ Q13 \\
Q15 &\equiv \texttt{from}\ Q13, Q14, \texttt{lesseqtab}\ \{NOrd, B\} \\
Q16 &\equiv \texttt{cagg}\ \texttt{Ord}\ \{NOrd\}\ Q15 \\
Q17 &\equiv \texttt{elim}\ \alpha(Q) \cup\ \{\texttt{Cnt}, \texttt{Ord}\}\ (\texttt{from}\ Q12, Q16)
\end{aligned}
$$

Applying these five subexpressions to the results in Table 3.8 turns out the final results shown in Table 3.9.

3.3.2 Translation of Q order by A desc

$\mathcal{OB}(Q$ order by A desc$)$ performs a descending ordering on results of query Q. The operation of $\mathcal{OB}$ for " Q order by A desc " is similar to what we presented in the previous subsection. However, there is a slight change in step 1, that is, $\mathcal{OB}$ groups Q by attributes A and for each group (with a distinct value for attribute A, say c), the number of A values of the base table which are greater than c is counted and named under attribute C. So, the only modification is in $Q3$ in which we change " from $Q1, Q2,$ lesseqtab $\{B, A\}$ " to " from $Q1, Q2,$ lesseqtab $\{A, B\}$ ".

For example , consider the results of the query Q in Example 3.3.3. The results of $\mathcal{OB}(Q)$ are presented in Table 3.10. The desired final results of $\mathcal{OB}(Q$ order by A desc$)$ are shown in Table 3.11. By applying this example in progressive subexpressions of $\mathcal{OB}(Q$ order by A desc$)$, we get Tables 3.12, 3.13, 3.14, 3.15, 3.16 and the final results in 3.17.

Table 3.10: ORDER BY EXAMPLE, $\mathcal{OB}(Q)$

A	A_2	`Cnt`	`Ord`
1	1	2	1
1	2	2	2
2	1	2	1
2	2	2	2

Table 3.11: RESULTS OF $\mathcal{OB}(Q$ `order by` $\{A\}$ `desc`$)$

A	A_2	`Cnt`	`Ord`
1	1	2	3
1	2	2	4
2	1	2	1
2	2	2	2

Table 3.12: $Q4$

A	C
1	2
2	1

Table 3.13: $Q9$

M
2

Table 3.14: $Q10$

A	CM
1	4
2	2

Table 3.15: $Q11$

A	A_2	`Cnt`	`Ord`	CM	$NOrd$
1	1	2	1	4	5
1	2	2	2	4	6
2	1	2	1	2	3
2	2	2	2	2	4

Table 3.16: $Q12$

A	A_2	`Cnt`	$NOrd$
1	1	2	5
1	2	2	6
2	1	2	3
2	2	2	4

Table 3.17: $Q17$

A	A_2	`Cnt`	`Ord`
1	1	2	3
1	2	2	4
2	1	2	1
2	2	2	2

Table 3.18: `select` Example, $\mathcal{OB}(Q)$

	B	A_1	`Cnt`	`Ord`
n_1	1	1	2	1
n_2	2	2	2	2
n_3	3	2	1	3
n_4	4	2	2	4
$^*n_5$	5	3	2	5
n_6	5	2	1	5
n_7	6	2	2	6

Table 3.19: Results of $\mathcal{OB}$(`select` $\{A_1\}$ Q)

	A_1	`Cnt`	`Ord`
g_1	1	2	1
g_2	2	5	2
g_3	2	1	3
g_4	2	2	4
g_5	3	2	3

3.3.3 Translation of `select` $\{A_1, \ldots, A_n\}$ Q

In this subsection, we show the ordered bag translation of expression `select` in detail. To illustrate this translation, we provide the following example and apply it in progressive subexpressions of $\mathcal{OB}$ on the expression $\mathcal{OB}$(`select` $\{A_1, \ldots, A_n\}$ Q).

Example 3.3.3 *Assume that* $(\mathcal{OB}(Q))^{\mathcal{I}}$ *computes the results in Table 3.18. For instance, the first row can be derived from the following query:*

((`from` C_1 `as` B, C_1 `as` A)
`union all`
(`from` C_1 `as` B, C_1 `as` A))
`order by` B

where $(C_1)^{\mathcal{I}} = \{1\}$.

Other rows of the table also can be derived from similar queries.

As we define the translation of $\mathcal{OB}$ *on* `select`*, we illustrate the definitions by incrementally applying subexpressions that progressively define* $\mathcal{OB}$(`select` $\{A_1, \ldots, A_n\}$ Q) *on query* Q *in this example. The desired final results are shown in Table 3.19.*

As the above example shows, $\mathcal{OB}$(`select` $\{A_1, \ldots, A_n\}$ Q) selects groups of tuples with schema $\{A_1, \ldots, A_n\}$ from Q together with their counts and orderings that have been derived from a stable sort on Q. For instance, in the above example, tuples n_2, n_3, and n_4 are grouped in group g_2 in the final results (Table 3.19) since they

have the same value for attribute A_1 and consecutive `Ord` values. Nevertheless, tuple n_5 causes tuple n_6 to be situated in g_3 (a different group from g_2). The behavior of function $\mathcal{OB}(\texttt{select}\ \{A_1, \ldots, A_n\}\ Q)$ can be described in five phases:

Phase 1:

Phase 1 is an intermediate state in which tuples with the potential of situating in one group are placed in pairs together with their `Ord` values. That is, tuples with the same $A_1, \ldots, A_n$ values are set in pairs which is done by the following three subexpressions.

$$
\begin{aligned}
Q1 &\equiv \texttt{elim}\ \{A_1, \ldots, A_n, \texttt{Ord}\}\ \mathcal{OB}(Q) \\
Q2 &\equiv \texttt{rename}\ A_1\ \texttt{as}\ B_1\ (\ldots(\texttt{rename}\ A_n\ \texttt{as}\ B_n(\texttt{rename}\ \texttt{Ord}\ \texttt{as}\ \mathrm{BOrd}(Q1)))\ldots) \\
Q3 &\equiv \texttt{from}\ Q1, Q2, \texttt{lesseqtab}\ \{\texttt{Ord}, \mathrm{BOrd}\}, A_1.\texttt{Id} = B_1.\texttt{Id}, \ldots, A_n.\texttt{Id} = B_n.\texttt{Id}
\end{aligned}
$$

Since, for each pair, one `Ord` value is needed to build up the final ordering (in our definitions, the maximum `Ord` value of each pair), `lesseqtab` $\{\texttt{Ord}, BOrd\}$, in subexpression three, removes extra pairs in which ordering of the first element (value of attribute `Ord` in subexpression three) is less than that of the second (value of attribute BOrd) since in our definitions the final ordering is derived from *BOrd* attribute.

Applying the four subexpressions on query Q in Example 3.3.3 reveals the results shown in Table 3.20.

Phase 2:

In phase 2, some pairs from the results of $Q3$ have to be dropped, and they are the following: 1) Those tuples, t, such that there exists a tuple with different A_1 value and greater/equal `Ord` value, which is also greater than the `Ord` value of some other tuple with the same A_1 value as t (i.e., a tuple in the same group as t). 2) Those tuples t such that there exists a tuple with different A_1 value and greater `Ord` value, which is also greater/equal to the `Ord` value of some other tuple with the same A_1 value as t. This is done via the following five subexpressions where $Q7$ selects the tuples which have to be removed based on the two conditions above. More precisely the first `from`

Table 3.20: $Q3$

	A_1	Ord	B_1	BOrd
	1	1	1	1
	2	2	2	2
	2	2	2	3
	2	2	2	4
*	2	2	2	5
*	2	2	2	6
	2	3	2	3
	2	3	2	4
*	2	3	2	5
*	2	3	2	6
	2	4	2	4
*	2	4	2	5
*	2	4	2	6
	2	5	2	5
*	2	5	2	6
	2	6	2	6
	3	5	3	5

expression in $Q7$ is for the condition 1 above and the second for condition 2.

$$
\begin{aligned}
Q4 &\equiv \texttt{rename}\ A_1\ \texttt{as}\ D_1\ (\ldots(\texttt{rename}\ A_n\ \texttt{as}\ D_n(\texttt{rename}\ \texttt{Ord}\ \texttt{as}\ DOrd\ (Q1)))\ldots)\\
Q5 &\equiv \texttt{from}\ Q3, Q4\\
Q6 &\equiv (Q5\ \texttt{minus}\ (\texttt{from}\ Q5, A_1.\texttt{Id} = D_1.\texttt{Id}))\\
&\quad\ \texttt{union}\\
&\quad\ \ldots\\
&\quad\ \texttt{union}\\
&\quad\ (Q5\ \texttt{minus}\ (\texttt{from}\ Q5, A_n.\texttt{Id} = D_n.\texttt{Id}))\\
Q7 &\equiv \texttt{elim}\ \alpha(Q3)(\\
&\quad\ (\texttt{from}\ Q5, Q6, \texttt{lesstab}\ \{\texttt{Ord}, DOrd\}, \texttt{lesseqtab}\ \{DOrd, \text{BOrd}\})\\
&\quad\ \texttt{union}\\
&\quad\ (\texttt{from}\ Q5, Q6, \texttt{lesseqtab}\ \{\texttt{Ord}, DOrd\}, \texttt{lesstab}\ \{DOrd, \text{BOrd}\}))\\
Q8 &\equiv Q3\ \texttt{minus}\ Q7
\end{aligned}
$$

Table 3.21: $Q8$

	A_1	`Ord`	B_1	BOrd
	1	1	1	1
+	2	2	2	2
+	2	2	2	3
*	2	2	2	4
+	2	3	2	3
*	2	3	2	4
	2	4	2	4
	2	5	2	5
	2	6	2	6
	3	5	3	5

To illustrate this phase more visibly, we apply the five subexpressions incrementally on results derived from the last phase in Table 3.20. As the example shows tuples marked by "$*$" in Table 3.20 are removed because of the tuple marked by "$*$" in Table 3.18.

Phase 3:

As mentioned in Phase 1, for each group of tuples, the maximum ordering (in our definitions maximum of *BOrd*) each group builds up the final ordering. So, in this phase, the following three subexpressions remove tuples of each group which have *BOrd* values less than the maximum *BOrd* of the group.

$$
\begin{aligned}
Q9 &\equiv \texttt{rename}\ B_1\ \texttt{as}\ D_1\ (\ldots(\texttt{rename}\ B_n\ \texttt{as}\ D_n(\texttt{rename}\ \mathit{BOrd}\ \texttt{as}\ \mathit{DOrd}(Q8)))\ldots)\\
Q10 &\equiv \texttt{elim}\ \alpha(Q3)(\texttt{from}\ Q8, Q9, \texttt{lesstab}\ \{\mathit{BOrd}, \mathit{DOrd}\})\\
Q11 &\equiv Q8\ \texttt{minus}\ Q10
\end{aligned}
$$

Applying these three subexpressions to the results in Table 3.21 reveals the results shown in Table 3.22. As these two tables show, tuples marked with "+" in Table 3.21 are removed because of the tuples marked with "$*$" in the same table.

Phase 4:

At this point, we have the groups of tuples with their relative orderings. This phase joins the derived results from the last phase with the base table in order to compute

Table 3.22: $Q11$

A_1	Ord	B_1	$BOrd$
1	1	1	1
2	2	2	4
2	3	2	4
2	4	2	4
2	5	2	5
2	6	2	6
3	5	3	5

Table 3.23: $Q14$

A_1	Cnt	$BOrd$
1	2	1
2	5	4
2	1	5
2	2	6
3	2	5

Table 3.24: $Q18$

A_1	Cnt	Ord
1	2	1
2	5	2
2	1	3
2	2	4
3	2	5

the count of each group. This is done via the following three subexpressions.

$$
\begin{aligned}
Q12 &\equiv \texttt{from}\ \mathcal{OB}(Q), Q11 \\
Q13 &\equiv \texttt{sagg}\ NCnt\ \{A_1, \ldots, A_n, BOrd\}\ \texttt{Cnt}\ Q12 \\
Q14 &\equiv \texttt{rename}\ NCnt\ \texttt{as}\ \texttt{Cnt}\ Q13
\end{aligned}
$$

Applying these three subexpressions to the results in Table 3.22 yields the results shown in Table 3.23.

Phase 5:

In this phase, similar to the last phase in the last subsection, the new $BOrd$ values are normalized and stored under attribute Ord. Consequently, the results have orderings which start from 1 and successively increase to the maximum value. This is done

using the following five subexpressions:

$$
\begin{aligned}
Q15 &\equiv \texttt{elim}\ \{BOrd\}\ Q14 \\
Q16 &\equiv \texttt{rename}\ BOrd\ \texttt{as}\ \mathrm{DOrd}\ Q15 \\
Q17 &\equiv \texttt{from}\ Q15, Q16, \texttt{lesseqtab}\ \{DOrd,\ BOrd\} \\
Q18 &\equiv \texttt{cagg}\ \texttt{Ord}\ \{BOrd\}\ Q17 \\
Q19 &\equiv \texttt{elim}\ \alpha(Q)\ \cup\ \{\texttt{Cnt}, \texttt{Ord}\}\ (\texttt{from}\ Q14, Q18)
\end{aligned}
$$

Applying these five subexpressions to the results in Table 3.23 turns out the final results shown in Table 3.24.

Chapter 4

Conclusion

We demonstrated a practical semantics for ordered bag SQL queries using a relational algebra with aggregates. We introduced the domain of finite and well-formed queries and continued our work in the domain of well-formed queries and showed that our defined semantics for well-formed queries is well-founded. Our method of defining semantics for well-formed bag and ordered bag queries, introduced in Chapter 3, enables us to reason about ordered bag queries using basic set semantics. To do so, we needed to add two interpreted columns to the results for the count and order of the results. We saw that relational algebra with aggregates can be used to compute the interpreted columns with sufficient flexibility to be used as a semantics for standard SQL-like queries, which may include order by and order preserving select clauses. In fact, our method is able to derive the semantics of more complex queries composed of any combination of set, bag, and order clauses from the simpler set semantics. The reduction of a workable ordered bag semantics for SQL to the relational algebra with aggregates can enable existing query containment theory, so that algebra can be employed to reason about practical query containment.

4.1 Future Work

In this work, we observed that our defined syntax and semantics are capable of capturing SQL-like queries with order by and duplicate preserving select clauses. However, our method for translation of ordered duplicate preserving select is not transparent. An area of future work could be to investigate alternative ways of defining semantics for such expressions capable of more efficiently applying query containment in prac-

Table 4.1: Outer Join Table

A	`Cnt`	`Ord`
1	1	1
2	1	1

Table 4.2: Inner Join Table

B	`Cnt`	`Ord`
1	1	1
2	1	2

Table 4.3: Nested Loop Join Results 1

	A	B
n_1	1	1
n_2	1	2
n_3	2	1
n_4	2	2

Table 4.4: Nested Loop Join Results 2

	A	B
n_3	2	1
n_4	2	2
n_1	1	1
n_2	1	2

tical query optimization.

In addition, using the introduced method, we are not able to capture lower query language expressions such as *nested loop joins*. For instance, consider a nested loop join which joins Table 4.1 and Table 4.2. Since the two tuples in the outer join table (Table 4.1) have an equal ordering, there are two possible orderings (Table 4.3 and Table 4.4). However, it is not possible to show these two possible orderings in a table using the attribute `Ord`. To see this, for example, we pick the tuple n_3. It occurs after n_2 in Table 4.3, while it occurs before n_2 in Table 4.4. Clearly, it is not possible to show these two possible orderings in one table using the `Ord` attribute. An interesting future investigation would be to consider possible ways of resolving this problem.

Bibliography

[CGV05] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Decidable containment of recursive queries. *Theoretical Computer Science*, 336(1):33–56, 2005.

[Cha92] Edward P. F. Chan. Containment and minimization of positive conjunctive queries in oodb's. In *Principles Of Database Systems*, pages 202–211, 1992.

[Coh05] Sara Cohen. Containment of aggregate queries. *Special Interest Group on Management Of Data Conference Rec.*, 34(1):77–85, 2005.

[CW93] Neil Coburn and Grant E. Weddell. A logic for rule-based query optimization in graph-based data models. In *Deductive and Object-Oriented Databases*, pages 120–145, 1993.

[GdB94] Paul W. P. J. Grefen and Rolf A. de By. A multi-set extended relational algebra - a formal approach to a practical issue. In *Proceedings of the Tenth International Conference on Data Engineering, February 14-18, 1994, Houston, Texas, USA*, pages 80–88. IEEE Computer Society, 1994.

[KV98] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. In *Proceedings of the Seventeenth Association for Computing Machinery Special Interest Group on Algorithms and Computation Theory-Special Interest Group on Management Of Data Conference-Special Interest Group on Artificial Intelligence Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 205–213. ACM Press, 1998.

[LCIS05] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. Ranksql: Query algebra and optimization for relational top-k queries. In

Special Interest Group on Management Of Data Conference, pages 131–142, 2005.

[LS97] Alon Y. Levy and Dan Suciu. Deciding containment for queries with complex objects. In *Principles Of Database Systems*, pages 20–31, 1997.

[LSCI05] Chengkai Li, Mohamed A. Soliman, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Ranksql: Supporting ranking queries in relational database management systems. In *Very Large Data Bases*, pages 1342–1345, 2005.

[NSS98] Werner Nutt, Yehoshua Sagiv, and Sara Shurin. Deciding equivalences among aggregate queries. In *Principles Of Database Systems*, pages 214–223, 1998.

[SSM96] David E. Simmen, Eugene J. Shekita, and Timothy Malkemus. Fundamental techniques for order optimization. In *Special Interest Group on Management Of Data Conference*, pages 57–67, 1996.