

Reasoning about Aggregate Queries

Werner Nutt

Faculty of Computer Science

Free University of Bozen-Bolzano

Joint work with Sara Cohen and Yehoshua Sagiv

Outline

2

- *Introduction*
 - motivation
 - examples and definitions
- *Equivalence*
 - general results
 - reminder on reasoning techniques for non-aggregate queries
 - reduction of aggregate equivalence
 - to properties of non-aggregate queries
- *Containment*
 - reduction to equivalence
- *Rewriting queries using views*
 - reduction to equivalence and containment

Aggregate Queries

3

Aggregate queries are queries that output the result of an aggregation function (and perhaps additional values)

Examples:

- “What is the **average** salary of the employees of each department?”
- “What is the **minimum** price at which bananas are sold?”
- “**How many** people voted for each of the candidates?”

When are Aggregate Queries Used?

4

- **Data warehouses**
 - store huge amounts of historical data
 - *Example:* all sales of products in all branches of a store
- **Stream data**
 - data that is continuously created
 - *Example:* information about telephone calls
- **Sensor networks**
 - virtual “database” made up of sensors that are physically scattered
 - *Example:* count the number of empty parking spaces per lot

Aggregate Queries are Expensive

5

- They must “touch” many items in order to return few items
- They require sorting, if there are grouping attributes
- Example:
 - “For each city, what was the average length of telephone calls to that city, in the month of July?”

The Need to Optimize

6

- Since aggregate queries are **common** and **expensive**, it is important to optimise such queries
- Two steps in optimisation
 - Logical Plan Space:
Find equivalent queries that might be cheaper to execute
 - Physical Plan Space:
Find different physical ways to execute the query

Our work deals with the first step

Two Approaches to Optimisation

7

When given a query q , try to find a query q' that is equivalent to q and is cheaper to execute, where

1. q' uses only the database relations
2. q' might use (precomputed) previous query results
(*“Query rewriting using views”*)

In order to implement these techniques, we must be able to determine when two queries are equivalent

Example 1: Advisors and PhD Students

8

Professors **advise** students, and people **live** in cities.

Suppose a database records this in the following **tables**:

```
advises(prof, student)
```

```
lives_in(person, city)
```

Query: “What is the *total number* of students for each *professor*?”

```
SELECT    advises.prof, COUNT(*)
FROM      advises
GROUP BY  advises.prof
```

Advisors and PhD Students (cntd.)

9

Suppose we have materialised a **view** `bozen_advisors`, with answers to the query

“What is the *total number* of students for each *professor who lives in Bozen*?”

```
CREATE VIEW bozen_advisors(prof, no_of_students) AS
SELECT      advises.prof, COUNT(*)
FROM        advises, lives_in
WHERE       advises.prof = lives_in.person AND
           lives_in.city = 'Bozen'
GROUP BY   advises.prof
```

Are the answers in the view also answers to our original query?

Advisors and PhD Students (cntd.)

10

Suppose we have materialised another **view** `trento_advisors`, with answers to the query

“What is the *total number of students from Trento* for each *professor*?”

```
CREATE VIEW trento_advisors(prof, no_of_students) AS
SELECT      advises.prof, COUNT(*)
FROM        advises, lives_in
WHERE       advises.student = lives_in.person AND
            lives_in.city = 'Trento'
GROUP BY   advises.prof
```

Are the answers in the view again answers to the original query?

Example 2: Salaries for Teaching Assistants at Hebrew Univ

11

Each TA has a **job type** in the **course** he assists:

$\text{job_type} = \{ \textit{marking}, \textit{tutoring}, \textit{instructing a lab} \}$

Teaching assistants are **financed** by different **sources**:

$\text{sponsorship} = \{ \textit{science foundation}, \textit{university} \}$

For each job type, each **sponsor** gives a fixed **amount**.

Tables ta, salary:

ta(name, course_name, job_type)

salary(job_type, sponsorship, amount)

Salaries for Teaching Assistants at Hebrew Univ (cntd.)

12

Two materialized **views**:

pos_per_type: “What is the *number of positions* of each *type*?”

```
CREATE VIEW pos_per_type(job_type, no_of_pos) AS
SELECT      job_type, COUNT(*)
FROM        ta
GROUP BY    job_type
```

ta_job_salary: “What is the *total salary* for each *type* of position?”

```
CREATE VIEW ta_job_salary(job_type, total_salary) AS
SELECT      job_type, SUM(amount)
FROM        salary
GROUP BY    job_type
```

Salaries for Teaching Assistants at Hebrew Univ (cntd.)

13

Query: “What is the *total amount* of money for each *job type*?” :

```
SELECT    ta.job_type, SUM(amount)
FROM      ta, salary
WHERE     ta.job_type = salary.job_type
GROUP BY ta.job_type.
```

Can we use the views to answer the query?

Salaries for Teaching Assistants at Hebrew Univ (cntd.)

14

For each type of job:

total amount spent = salary per position × number of pos of that type

⇒ *Combine the answer from the two views*

```
SELECT    ta_job_salary.job_type,  
          total_salary * no_of_pos  
FROM      ta_job_salary, pos_per_type  
WHERE     ta_job_salary.job_type = pos_per_type.job_type  
GROUP BY ta_job_salary.job_type.
```

How can a query optimizer find out about this?

What do Non-Aggregate Queries Look Like?

15

Instead of SQL, we use a datalog like notation:

- **Conjunctive Queries:** “Professors advising students from Trento”

$$q(p) \leftarrow \text{advises}(p, s) \wedge \text{lives_in}(s, \text{trento})$$

- **Conjunctive Queries with Comparisons:** “Teaching assistants and the respective amount, if they receive more than 500 Euros from some sponsor for one of their jobs”

$$q(n, a) \leftarrow \text{job_type}(n, c, j) \wedge \text{sponsorship}(j, sp, a) \wedge s > 500$$

What do Non-Aggregate Queries Look Like? (cntd.)

16

- **Disjunctive Queries:** “Advisors living in Bozen or advising students from Trento”

$$q(p) \leftarrow (\text{advises}(p, s1) \wedge \text{lives_in}(p, \text{bozen})) \vee \\ (\text{advises}(p, s2) \wedge \text{lives_in}(s2, \text{trento}))$$

- **Queries with Negated Atoms:** “Professors advising a student who does not live in Bozen”

$$q(p) \leftarrow \text{advises}(p, s) \wedge \neg \text{lives_in}(s, \text{bozen})$$

Aggregate Functions

17

Aggregate functions

- are applied to **bags** (multisets) **of values**
- return a **single value** for each bag.

Some common examples:

- count
- sum
- average
- product
- maximum

What do Aggregate Queries Look Like?

18

The *head* contains: **grouping variables**
aggregation term(s)

$$q(\boxed{x, y}, \boxed{\text{sum}(z)}) \leftarrow \boxed{p_1(x, c) \wedge p_2(x, y, w) \wedge p_3(w, z) \wedge y \geq 5}$$

The *body* is similar to the body
of non-aggregate queries

- We assume w.l.o.g. that there is only one aggregation term
- Bodies may involve comparisons, disjunction and negated atoms
- To ease the presentation, we will mostly consider conjunctive queries

Aggregate Query Examples

19

- **Count Query:** “How many students from each city does Rossi advise?”

$$q(c, count) \leftarrow \text{advices}(\text{rossi}, s) \wedge \text{lives_in}(s, c)$$

- **Sum Query:** “What is the total salary for each TA of Database Systems?”

$$q(n, \text{sum}(a)) \leftarrow \text{job_type}(n, \text{db_sys}, j) \wedge \text{sponsorship}(j, sp, a)$$

How are Non-Aggregate Queries Evaluated?

20

$$q(n, a) \leftarrow \text{job_type}(n, \text{db_sys}, j) \wedge \text{sponsorship}(j, sp, a)$$

1. Find satisfying assignments for the body
2. Create a tuple from each satisfying assignment by projecting on the head of the query

Notation: Applying \boxed{q} to database $\boxed{\mathcal{D}}$ gives the result set $\boxed{q^{\mathcal{D}}}$

How are Aggregate Queries Evaluated?

21

$$q(n, j, \text{sum}(a)) \leftarrow \text{job_type}(n, \text{db_sys}, j) \wedge \text{sponsorship}(j, \text{sp}, a)$$

1. Find satisfying assignments for the body
2. Group assignments according to values of grouping variables (here (n, j))
3. For each group, collect the multiset of values assigned to the aggregation variables (here $\gamma(a)$)
4. Apply the aggregation function sum to those multisets

Notation: Again, applying q to \mathcal{D} gives $q^{\mathcal{D}}$

Containment and Equivalence

22

Let q, q' be queries (aggregate or non-aggregate)

- q is **contained** in q' iff for all \mathcal{D}

$$q^{\mathcal{D}} \subseteq q'^{\mathcal{D}}$$

- q is **equivalent** to q' iff for all \mathcal{D}

$$q^{\mathcal{D}} = q'^{\mathcal{D}}$$

Notation: “ $q \subseteq q'$ ”, “ $q \equiv q'$ ”

Also: containment and equivalence w.r.t. integrity constraints

Rewriting a Query Using Views

23

Given

- a set $\boxed{\mathcal{V}} = \{v_1, \dots, v_n\}$ of queries, called *views*
- another *query* \boxed{q}

Find

- a query \boxed{r} that uses *database* and *view predicates* such that
 - evaluating q , and
 - evaluating first \mathcal{V} and then ryields the same results over *all databases*.

Then r is an (equivalent) **rewriting** of q using \mathcal{V}

Also: contained rewriting

“Query Rewriting” is a Fundamental Problem in Databases

24

- **Query Optimisation**

If a costly operation (join, aggregation) has already been executed, one may save executing it a second time.

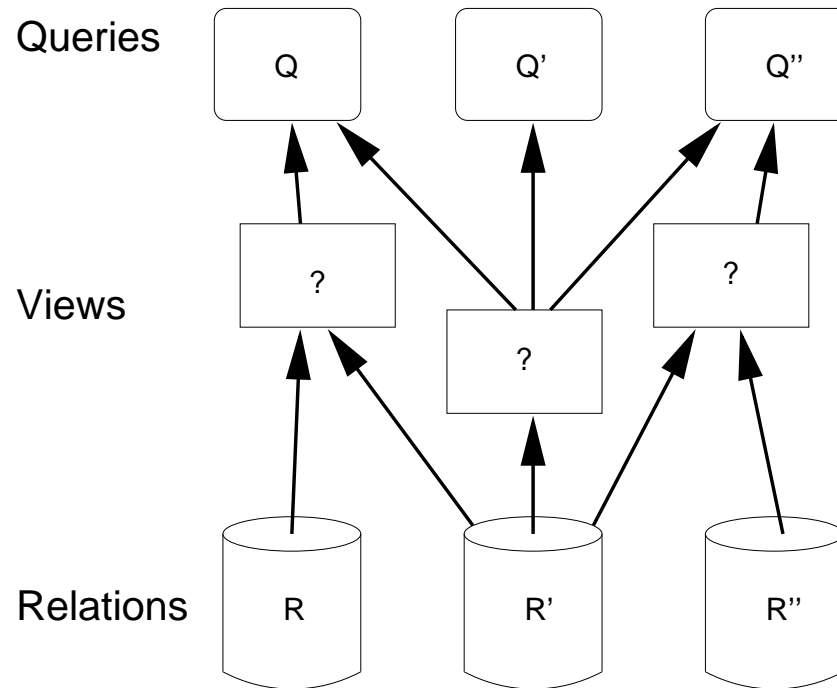
- **Data Security**

Check that it is *impossible* to answer a query using certain views

- **Design Optimisation**

- **Information Integration**

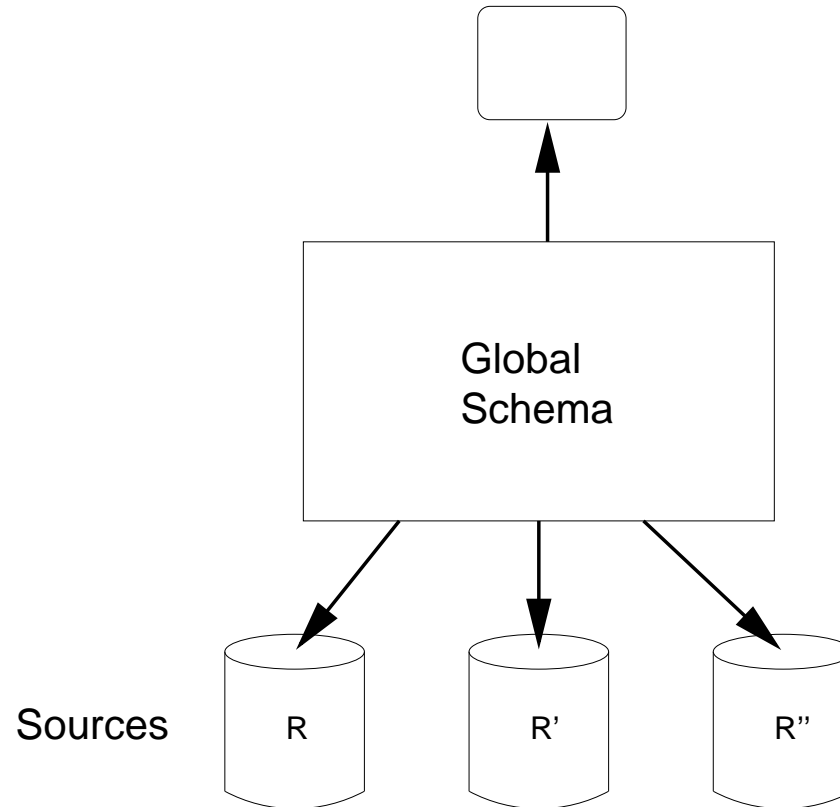
Design Optimisation



Compute views that can serve as intermediate results
when answering the queries

Integrated Access to Heterogeneous Data Sources

26



Consider sources as “views” on a global schema

Dimensions of Reasoning Problems

27

Aggregation functions in queries and views:

min, max, top2, count, parity, sum, prod, ...

Queries (and views)

- **conjunctive** or **disjunctive**
- without or with **comparisons** $<$, \leq , \geq , $>$
- comparisons over **rational numbers** or **integers**

Rewritings

- **conjunctive** or **disjunctive**
- **partial** or **complete**

~> Different techniques for different combinations

Equivalence Checking for Non-Aggregate Queries

28

- Much research during the past 30 years
- Approach by reduction to containment

$$q \equiv q' \iff q \subseteq q' \text{ and } q' \subseteq q$$

- **Basic idea for containment checking:**

Given q, q' , find several small databases $\mathcal{D}_1, \dots, \mathcal{D}_n$ such that

$$q \subseteq q' \iff q^{\mathcal{D}_j} \subseteq q'^{\mathcal{D}_j} \text{ for } j = 1, \dots, n$$

- Also formulated in terms of homomorphisms between queries

Checking Equivalence of Aggregate Queries

29

It is not at all clear that this approach is applicable to aggregate queries

- The result of an aggregate query is highly dependent on the exact values that appear in the database
- Even if it is enough to consider *small* databases, there are infinitely many different small databases.

Another Problem: Aggregation Function Quirks

30

Each aggregation function has its own quirk which requires considering special boundary cases:

Count: standard function, counts values

Max: ignores repeated values

Sum: ignores 0 values

Prod: ignores 1 values, turns to 0, when there is a 0 in the bag

Equivalence is Often Decidable

31

With generic results for classes of aggregation functions α we could show:

Theorem: *Equivalence of **disjunctive** aggregate queries is **decidable** for the operators*

min, max, top2, count, parity, sum, prod.

*This holds even for queries with **negated atoms**.*

[TOCL 05]

Equivalence of α -Queries: Proof Idea

32

Distinguish between three Problems:

- **Equivalence:** Given q, q' ,
— is $q^{\mathcal{D}} = q'^{\mathcal{D}'}$ for all databases \mathcal{D} ?
- **Bounded Equivalence:** Given q, q' and $N > 0$,
— is $q^{\mathcal{D}} = q'^{\mathcal{D}'}$ for all databases \mathcal{D} with at most N constants?
- **Local Equivalence:** Given q, q' ,
— is $q^{\mathcal{D}} = q'^{\mathcal{D}'}$ for all databases \mathcal{D}
with at most $size(q) + size(q')$ constants?

Strategy for a given α :

- (1) Show decidability of Bounded Equivalence
- (2) Reduce Equivalence to Local Equivalence

Idea:

- Create finitely many **test database** that represent all databases with at most N constants
- Test databases contain **variables** with **complete order** constraints
- Evaluate q and q' over test databases
- Check whether q and q' return the same values by checking **validity of test formulas**

Example: Creating Test Databases for $N = 3$

34

$$q(\text{prod}(y)) \leftarrow p(y) \wedge y > 1 \wedge p(z) \wedge z > 1$$
$$q'(\text{prod}(y)) \leftarrow p(y) \wedge y \geq 1 \wedge p(z) \wedge z > 1$$

A test database contains

- tuples for p (the only predicate in the queries)
- the values of p are taken from either
 - constants in the queries (here, 1), or
 - from N (here, 3) new variables
- a complete ordering of the variables and constants

Evaluation over a Test Database (Example cont'd)

35

$$\begin{aligned}q(\text{prod}(y)) &\leftarrow p(y) \wedge y > 1 \wedge p(z) \wedge z > 1 \\q'(\text{prod}(y)) &\leftarrow p(y) \wedge y \geq 1 \wedge p(z) \wedge z > 1\end{aligned}$$

Example test database:

$$\mathcal{D} = \{p(1), p(u), p(w)\} \quad 1 < u < w$$

Evaluation yields

$$\begin{aligned}q^{\mathcal{D}} &= \text{prod}(\{u, u, w, w\}) = u^2 w^2 \\q'^{\mathcal{D}} &= \text{prod}(\{u, u, w, w, 1, 1\}) = 1^2 u^2 w^2\end{aligned}$$

Validity of Test Formulas

- Equivalence of q and q' over all instances of the test database amounts to checking the validity of a test formula, like,

$$1 < u < w \implies u^2 w^2 = 1^2 u^2 w^2.$$

Theorem:

Bounded equivalence of α -queries is decidable iff validity of test formulas for α is decidable

- This yields decidability of Bounded Equivalence for a number of α s, e.g.,

median, parity, cntd, count, max, sum, prod

Reducing Equivalence to Local Equivalence

37

- **Possible** if α can be computed over a large database \mathcal{D} by
 - splitting \mathcal{D} into many small \mathcal{D}_i s
 - computing α over the \mathcal{D}_i s
 - combining the results.
- **Necessary:** α stems from an associative commutative operation
- **Problem:** Possible overlaps between the \mathcal{D}_i s
- **Solution** exists if
 - multiplicities of values don't matter (e.g., *max*), or
 - double counts can be compensated by inverses
(e.g., *sum*, *count*, *prod*)

Decidability Results are not Enough

38

- The generic method is hopelessly *inefficient*
- To construct rewritings, we need syntactic characterisations of equivalence and containment

~> *look at special aggregation functions*

Reduction to Queries without Aggregation

39

Definition: The **core** of

$$q(\bar{x}, \alpha(\bar{y})) \leftarrow R \wedge C$$

is

$$\check{q}(\bar{x}, \bar{y}) \leftarrow R \wedge C.$$

Examples

- $q(\bar{x}, \text{sum}(y)) \leftarrow R \wedge C \quad \rightsquigarrow \quad \check{q}(\bar{x}, y) \leftarrow R \wedge C$
- $q(\bar{x}, \text{count}) \leftarrow R \wedge C \quad \rightsquigarrow \quad \check{q}(\bar{x}) \leftarrow R \wedge C$

Idea: Reduce equivalence and containment of aggregate queries to properties of their cores

Query Homomorphism: Definition

40

An **homomorphism** from

$$q'(\bar{x}) \leftarrow R' \wedge C'$$

to

$$q(\bar{x}) \leftarrow R \wedge C$$

is a *substitution* θ such that

- $\theta\bar{x} = \bar{x}$
- $\theta R' \subseteq R$
- $C \models \theta C'$.

Query Homomorphism: Example

41

Example:

$$q'(x) \leftarrow p(x, y) \wedge r(y, z) \wedge \\ y \leq 3$$

$$q(x) \leftarrow p(x, w) \wedge p(x, x) \wedge r(x, u) \wedge \\ w \geq 5 \wedge x \leq 2$$

An homomorphism from q' to q is

$$\theta = \{x/x, y/x, z/u\}.$$

Finding an homomorphism is NP-complete!

Checking Containment

42

Theorem (Chandra/Merlin 77): *For relational conjunctive queries:*

$q \subseteq q' \iff$ *there is an homomorphism from q' to q*

What about queries with comparisons?

Containment and Comparisons: Classical Example

43

$$q' \leftarrow p(u, v) \wedge u \leq v$$

$$q \leftarrow p(y, z) \wedge p(z, y)$$

We see: q is *contained* in q' ,

but there is *no homomorphism* from q' to q .

Idea: replace q with its **linear expansion** $(q_L)_L$!

$$q_{\{y < z\}} \leftarrow p(y, z) \wedge p(z, y) \wedge y < z$$

$$q_{\{y = z\}} \leftarrow p(y, z) \wedge p(z, y) \wedge y = z$$

$$q_{\{y > z\}} \leftarrow p(y, z) \wedge p(z, y) \wedge y > z$$

(*case analysis*)

Containment and Comparisons: Klug's Theorem

44

Theorem (Klug 88): If

- q, q' are **conjunctive queries with comparisons**
- $(q_L)_L$ is the **linear expansion** of q ,

then:

$q \subseteq q' \iff$ for every q_L in $(q_L)_L$,
there is an homomorphism from q' to q_L

(analogous for disjunctive queries)

Containment with comparisons is Π_2^P -complete.

(van der Meyden)

Relational Max-Queries

45

$$q(\bar{x}, \max(y)) \leftarrow R \quad \equiv \quad q'(\bar{x}, \max(y)) \leftarrow R' ?$$

Theorem: *For relational max-queries:*

$$q \equiv q' \quad \Leftrightarrow \quad \text{the cores } \check{q} \text{ and } \check{q}' \text{ are equivalent}$$

Relational queries deliver the **same max**

only if they deliver the **same values!**

The Theorem Fails If There Are Comparisons

46

$$q(\max(y)) \leftarrow p(y) \wedge p(z) \wedge z < y$$

$$q'(\max(y)) \leftarrow p(y) \wedge p(z_1) \wedge p(z_2) \wedge z_1 < z_2$$

Observations:

- \check{q} returns *all* elements of p , *but the least*
- \check{q}' returns *all* elements
- \check{q} and \check{q}' return answers if p has at least two elements

\Rightarrow the max-queries are equivalent.

Which property of cores entails equivalence of max-queries?

Dominance

47

Definition: We say that \check{q} is **dominated** by \check{q}' iff for every DB, if \check{q} returns (\bar{d}, d) , then \check{q}' returns some (\bar{d}, d') such that $d \leq d'$.

Proposition: *For arbitrary max-queries:*

$q \sim q' \iff$ *the cores \check{q} and \check{q}' dominate each other*

\rightsquigarrow How can we check dominance?

Dominance Mappings

48

A **dominance mapping** θ from

$$\check{q}'(\bar{x}, y) \leftarrow R' \wedge C' \quad \text{to} \quad \check{q}(\bar{x}, y) \leftarrow R \wedge C$$

is like a homomorphism, except that

- $C \models y \leq \theta(y)$

instead of $y = \theta(y)$.

Theorem:

q is dominated by q' \Leftrightarrow for every q_L in the lin. expansion of q ,
there is a dominance mapping from q' to q_L

\rightsquigarrow Equivalence of max-queries is Π_2^P -complete.

Equivalence of Relational *count*-Queries

49

$$q(\bar{x}, \text{count}) \leftarrow R \quad \sim \quad q'(\bar{x}, \text{count}) \leftarrow R' ?$$

Clearly: $\check{q}(\bar{x})$ and $\check{q}'(\bar{x})$ return for ever DB

identical results with the same multiplicities

Or: $\check{q}(\bar{x})$ and $\check{q}'(\bar{x})$ are *equivalent under bag-set-semantics*

(“bag-set-equivalent”)

Theorem (Chaudhuri/Vardi 93):

For relational conjunctive queries q, q' we have:

$$q \text{ and } q' \text{ are bag-set-equivalent} \iff q \text{ and } q' \text{ are isomorphic}$$

*i.e. q and q' are identical
up to renaming of variables*

Proof Technique: Counting Polynomials

50

Show: *If q and q' are bag-set-equivalent,
then there exists a surjective homomorphism $q' \rightarrow q$*

- From query $q(\bar{x}) \leftarrow R(\bar{x}, z_1, \dots, z_k)$ and test answer \bar{d}
construct a family $(\mathcal{D}_{\bar{N}})_{\bar{N} \in \mathbf{N}^k}$ of test DBs:

$$\left. \begin{array}{l} \Gamma(\bar{N}) \\ \Gamma'(\bar{N}) \end{array} \right\} := \text{multiplicity of the answer } \bar{d} \text{ over } \mathcal{D}_{\bar{N}} \text{ for } \begin{cases} q \\ q' \end{cases}$$

- $\Gamma(\bar{N})$ and $\Gamma'(\bar{N})$ are *polynomials* in $\bar{N} = (N_1, \dots, N_k)$
- $\Gamma(\bar{N})$ contains a *monomial* $cN_1^1 \cdots N_k^1$
- If $\Gamma'(\bar{N})$ contains a *monomial* $c'N_1^1 \cdots N_k^1$,
then there exists a *surjective homomorphism* $q' \rightarrow q$

The isomorphism theorem is wrong for queries with comparisons:

$$q(\text{count}) \leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < y \wedge x < z$$

$$q'(\text{count}) \leftarrow p(x) \wedge p(y) \wedge p(z) \wedge x < z \wedge y < z$$

Observation:

- q and q' are *not isomorphic*
- q and q' are *equivalent*

How can we test bag-set-equivalence?

Equivalence of General *count*-Queries

52

Theorem: For any disjunctive queries q, q' ,

q and q' are bag-set-equivalent \iff q and q' have isomorphic linear expansions

Note: Linear expansions lead to exponential blow-up!

\rightsquigarrow Equivalence of disjunctive *count*-queries is in PSPACE

When is

$$q(\bar{x}, \text{sum}(y)) \sim q'(\bar{x}, \text{sum}(y)) ?$$

Bag-set-equivalence of the cores is a **sufficient** condition.

But is it also necessary?

Queries with *sum*: Problem 1

54

The cores of q and q' are *not bag-set-equivalent*:

$$q(\text{sum}(y)) \leftarrow p(1) \wedge p(2) \wedge p(3) \wedge \\ p(y) \wedge 1 \leq y \leq 3$$

$$q'(\text{sum}(y)) \leftarrow p(1) \wedge p(2) \wedge p(3) \wedge \\ p(y) \wedge 1 \leq y \leq 2 \wedge \\ p(z) \wedge 1 \leq z \leq 2$$

Over the **integers**, q and q' are **equivalent**, since

$$1 + 2 + 3 = 1 + 1 + 2 + 2.$$

Over the **rational numbers**, they are **not**.

Queries with *sum*: Problem 2

55

$$\begin{aligned} q(\text{sum}(y)) \quad \leftarrow \quad & p(y) \wedge 0 \leq y \wedge \\ & p(z) \wedge 0 < z \wedge \\ & p(w) \wedge 0 \leq w \end{aligned}$$

$$\begin{aligned} q'(\text{sum}(y)) \quad \leftarrow \quad & p(y) \wedge 0 < y \wedge \\ & p(z) \wedge 0 < z \wedge \\ & p(w) \wedge 0 \leq w \end{aligned}$$

\check{q} and \check{q}' return *non-zero numbers* with the *same multiplicity* ...

... but \check{q} may return 0, while \check{q}' does not.

Hence: q and q' are equivalent, but the cores are not

Equivalence of Queries with *sum*

56

Problems arise because of comparisons **and** constants:

Theorem: *If q, q' are sum-queries without comparisons or without constants, then:*

q and q' are equivalent \iff the cores \check{q} and \check{q}' are bag-set-equivalent

In the general case characterisation by isomorphism of linear expansions:

Theorem: *For sum-queries with comparisons*

- *over the **integers** or*
 - *over the **rational numbers***
- equivalence is decidable and in PSPACE.*

Complexity of the Equivalence Problem

57

Test “ $q \equiv q'$ ” for conjunctive and disjunctive aggregate queries

Aggregation function	q arbitrary q' arbitrary	q w/o comp. q' arbitrary	q w/o comp. q' w/o comp.	q linear [†] q' arbitrary	q linear q' linear
<i>count,</i> <i>sum, prod</i>	GI-hard*/ in PSPACE	GI-compl.	GI-compl.	polyn.	polyn.
<i>max, min,</i> <i>top2</i>	Π_2^P -compl.	Π_2^P -compl.	NP-compl.	NP-hard/ in Π_2^P	polyn.

* GI = Graph Isomorphism

† linear = no repeated relation symbols

Containment of Aggregate Queries

58

Well studied for *non*-aggregate queries, but no work for aggregate queries
(except for *count*, as “containment under bag-semantics”)

Definition (Chaudhuri/Vardi and Ioannidis/Ramakrishnan)

$q(\bar{x}, \text{count})$ is contained in $q'(\bar{x}, \text{count})$

iff

for every DB \mathcal{D} and every tuple \bar{d} we have:

$(\bar{d}, n) \in q^{\mathcal{D}} \Rightarrow$ there is some $n \geq n'$ such that $(\bar{d}, n') \in q'^{\mathcal{D}}$

i.e. q' “exceeds” q .

For this definition containment is

- Π_2^P -hard for conjunctive queries
- *undecidable* for disjunctive queries

Alternative Definition (Cohen/Nutt/Sagiv 2003)

59

$q(\bar{x}, count)$ is contained in $q'(\bar{x}, count)$

iff

for every DB \mathcal{D} and every tuple \bar{d} :

$$(\bar{d}, n) \in q^{\mathcal{D}} \Rightarrow (\bar{d}, n) \in q'^{\mathcal{D}}$$

(i.e. $q^{\mathcal{D}} \subseteq q'^{\mathcal{D}}$)

Results:

- *Containment can be reduced to equivalence*
for a large class of aggregation functions
(e.g. *count, cntd, sum, max, min, avg, median, prod*)
- Holds also for queries with *negative atoms* and
for *containment w.r.t. integrity constraints*

[ICDT'03]

Products and Containment

60

Idea: Consider the relation “ \supseteq ” over \mathbf{N}_0 , defined by

$$m \supseteq n \quad \text{if} \quad m = n \quad \text{or} \quad n = 0.$$

Then:

$$\boxed{m \supseteq n \quad \text{iff} \quad m \cdot n = n \cdot n}$$

Reduction: Products of Aggregate Queries

61

Given two aggregate queries

$$q'(\bar{x}, \text{sum}(y')) \leftarrow B'(\bar{x}, y', \bar{z}')$$

$$q(\bar{x}, \text{sum}(y)) \leftarrow B(\bar{x}, y, \bar{z}),$$

we define the **product** $q' \otimes q$ as

$$(q' \otimes q)(\bar{x}, \text{sum}(y')) \leftarrow B'(\bar{x}, y', \bar{z}') \wedge B(\bar{x}, y, \bar{z}).$$

(sum as an example)

Reduction: Proof Idea for *count*

62

$$(q' \otimes q)(\bar{x}, \text{count}) \leftarrow B'(\bar{x}, z') \wedge B(\bar{x}, z)$$

Let \bar{d} be a fixed answer tuple (= binding for \bar{x}). Then we have:

• q' returns m times the answer \bar{d}

• q returns n times the answer \bar{d}

$\Rightarrow (q' \otimes q)$ returns $m \cdot n$ times the answer \bar{d}

$\Rightarrow (q \otimes q)$ returns $n \cdot n$ times the answer \bar{d} (*analogous argument*)

Hence

$$q' \supseteq q \quad \text{iff} \quad (q' \otimes q) \equiv (q \otimes q)$$

When Does This Reduction Work?

63

Question: Is it always true that $q' \supseteq q$ iff $(q' \otimes q) \equiv (q \otimes q)$?

Answer: No. However, it is true for queries
with “**expandable**” aggregation functions.

Theorem: *If q and q' contain expandable aggregation functions, then*

$$(q' \supseteq q) \quad \text{iff} \quad (q' \otimes q) \equiv (q \otimes q).$$

The theorem holds even if q, q' have disjunctions, negation and comparisons
and in the presence of integrity constraints

n -Expansion of a Bag

64

For a bag B , the n -**expansion** $B \otimes n$ is created
by duplicating every element in B exactly n times.

$$B = \{1, 2, 4\}$$

bag B

$$B \otimes 2 = \{1, 1, 2, 2, 4, 4\}$$

2-expansion of B

$$B \otimes 3 = \{1, 1, 1, 2, 2, 2, 4, 4, 4\}$$

3-expansion of B

Expandable Aggregation Functions

65

An aggregation function α is **expandable** if

$$\alpha(B) = \alpha(B') \iff \alpha(B \otimes n) = \alpha(B' \otimes n).$$

for all bags B, B' and all $n > 0$.

- **Expandable:**

count, cntd, max, top2, sum, avg, standard dev, median

- **Non-expandable:**

parity, prod

Example: Professors From Bozen

66

Students per professor:

$$q'(p, count) \leftarrow \text{advises}(p, s)$$

Students per professor from Bozen:

$$q(p, count) \leftarrow \text{advises}(p, s) \wedge \text{lives_in}(p, \text{bozen})$$

Containment Check:

$$(q' \otimes q)(p, count) \leftarrow \begin{aligned} &\text{advises}(p, s) \wedge \\ &\text{advises}(p, s') \wedge \text{lives_in}(p, \text{ta}) \end{aligned}$$

$$(q \otimes q)(p, count) \leftarrow \begin{aligned} &\text{advises}(p, s) \wedge \text{lives_in}(p, \text{ta}) \wedge \\ &\text{advises}(p, s') \wedge \text{lives_in}(p, \text{ta}) \end{aligned}$$

$$(q' \otimes q) \text{ and } (q \otimes q) \text{ are isomorphic } \rightsquigarrow q \subseteq q'$$

Example: Students from Trento

67

Students per professor:

$$q'(p, count) \leftarrow \text{advices}(p, s)$$

Students from Trento per professor:

$$q(p, count) \leftarrow \text{advices}(p, s) \wedge \text{lives_in}(s, \text{trento})$$

Containment check:

$$(q' \otimes q)(p, count) \leftarrow \begin{aligned} &\text{advices}(p, s) \wedge \\ &\text{advices}(p, s') \wedge \text{lives_in}(s', \text{trento}) \end{aligned}$$

$$(q \otimes q)(p, count) \leftarrow \begin{aligned} &\text{advices}(p, s) \wedge \text{lives_in}(s, \text{trento}) \wedge \\ &\text{advices}(p, s') \wedge \text{lives_in}(s', \text{trento}) \end{aligned}$$

$$(q' \otimes q) \text{ and } (q \otimes q) \text{ are not isomorphic } \rightsquigarrow q \not\subseteq q'$$

Example: Functional Dependencies

68

Advisors per student:

$$q'(s, count) \leftarrow \text{advises}(p, s)$$

Visiting advisors per student:

$$q(s, count) \leftarrow \text{advises}(p, s) \wedge \text{visitor}(p)$$

Clearly, q is **not** contained in q' !

What if a student can have at most one advisor?

Example: Functional Dependencies (cntd.)

69

$$(q' \otimes q)(s, count) \leftarrow \text{advices}(p, s) \wedge \\ \text{advices}(p', s) \wedge \text{visitor}(p')$$

$$(q \otimes q)(s, count) \leftarrow \text{advices}(p, s) \wedge \text{visitor}(p) \wedge \\ \text{advices}(p', s) \wedge \text{visitor}(p')$$

We apply the “chase”: $p = p'$!

$$(q' \otimes q)(s, count) \leftarrow \text{advices}(p, s) \wedge \\ \text{advices}(p, s) \wedge \text{visitor}(p)$$

$$(q \otimes q)(s, count) \leftarrow \text{advices}(p, s) \wedge \text{visitor}(p) \wedge \\ \text{advices}(p, s) \wedge \text{visitor}(p)$$

Example: Functional Dependencies (cntd.)

70

After the chase:

$$(q' \otimes q)(s, count) \leftarrow \text{advises}(p, s) \wedge \text{advises}(p, s) \wedge \text{visitor}(p)$$

$$(q \otimes q)(s, count) \leftarrow \text{advises}(p, s) \wedge \text{visitor}(p) \wedge \text{advises}(p, s) \wedge \text{visitor}(p)$$

Now

- $(q' \otimes q)$ and $(q \otimes q)$ are *isomorphic*
- $\Rightarrow (q' \otimes q)$ and $(q \otimes q)$ are *equivalent*
- $\Rightarrow q$ is *contained* in q'

Strategy for the Design of Rewriting Algorithms

71

q, q' queries \mathcal{V} views r query with view predicates

Distinguish three problems:

- **Query Equivalence (and Containment)**
- **Rewriting Verification:** “Is r a rewriting of q using \mathcal{V} ?”
(i.e. “Is r equivalent to q w.r.t. the view definitions?”)
- **Rewriting Computation:** “Find a (all) rewriting(s) of q using \mathcal{V} !”

Strategy: *Reduce difficult problems to simple ones*

[TODS'06]

Reduktion: “Rewriting Verification \rightarrow Equivalence”

72

- **Rewriting Verification:** $\boxed{q \equiv_{\mathcal{V}} r}$? *equivalence modulo \mathcal{V}*

- **Problem:** How can we get rid of \mathcal{V} ?

- **Idea:** Unfold r ! *replace the views by their definitions!*

Then: $q \equiv r^{\text{unf}}$? *simple equivalence*

- **Problem due to aggregation:**

in general, unfolding does not preserve equivalence

\rightsquigarrow *When unfolding, pay attention to the aggregation function*

\rightsquigarrow *Allow only r 's that have a special form (**rewriting candidates**)*

Which Aggregation Function?

73

$$q(t, \text{sum}(a)) \leftarrow \text{ta}(n, cn, t) \wedge \text{salary}(t, s, a)$$

$$\text{pos_per_type}(t, \text{count}) \leftarrow \text{ta}(n, cn, t)$$

$$\text{ta_job_salary}(t, \text{sum}(a)) \leftarrow \text{salary}(t, s, a)$$

Candidate: $r(t, \boxed{f(c, a)}^?) \leftarrow \text{pos_per_type}(t, c) \wedge \text{ta_job_salary}(t, a)$

Unfolding: $r^{\text{unf}}(t, \text{sum}(a)) \leftarrow \text{ta}(n', cn', t) \wedge \text{salary}(t, s', a)$

We want that $r \equiv_{\mathcal{V}} r^{\text{unf}}$ (“Unfolding Theorem”)

$$\Rightarrow \text{the only choice for } f \text{ is } \boxed{f(c, a)} := \boxed{\text{sum}(c * a)}$$

This is a theorem!

Then: $r \equiv_{\mathcal{V}} r^{\text{unf}} \equiv q \implies r$ is a rewriting of q

Summary: Rewriting Candidates

74

	Candidate Type	Rewriting Candidate
<i>count</i>	<i>count</i>	$r(\bar{s}, \text{sum}(\prod_{i=1}^n z_i)) \leftarrow v_1^c(\theta_1 \bar{s}_1, z_1) \wedge \dots \wedge v_n^c(\theta_n \bar{s}_n, z_n) \wedge C$
<i>sum</i>	<i>count</i>	$r(\bar{s}, \text{sum}(y * \prod_{i=1}^n z_i)) \leftarrow v_1^c(\theta_1 \bar{s}_1, z_1) \wedge \dots \wedge v_n^c(\theta_n \bar{s}_n, z_n) \wedge C$
	<i>sum/count</i>	$r(\bar{s}, \text{sum}(y * \prod_{i=1}^n z_i)) \leftarrow v^s(\theta \bar{s}, y) \wedge v_1^c(\theta_1 \bar{s}_1, z_1) \wedge \dots \wedge v_n^c(\theta_n \bar{s}_n, z_n) \wedge C$
<i>max</i>	<i>non-agg.</i>	$r(\bar{s}, \text{max}(y)) \leftarrow v_1(\theta_1 \bar{s}_1) \wedge \dots \wedge v_n(\theta_n \bar{s}_n) \wedge C$
	<i>max/ non-agg.</i>	$r(\bar{s}, \text{max}(y)) \leftarrow v^m(\theta \bar{s}, y) \wedge v_1(\theta_1 \bar{s}_1) \wedge \dots \wedge v_n(\theta_n \bar{s}_n) \wedge C$

All candidates satisfy the Unfolding Theorem

Existence of Rewritings

75

Reminder: we restrict ourselves to rewriting candidates

Enumeration

1. **Guess** a candidate
2. **Verify** whether it is a rewriting

... gives only semi-decidability

↪ How large can a (disjunctive) rewriting be? In general arbitrarily large!
(*due to new constants*)

For conjunctive queries and rewritings:

Theorem (Short Rewritings): *If there is some rewriting of q using \mathcal{V} , then there is also one that has no more atoms than q .*

Finding *count* and *sum*-rewritings

76

Theorem: *Existence of equivalent conjunctive rewritings is decidable for conjunctive queries and views.*

query type	complete rewriting	partial rewriting
arbitrary	NP-hard/in PSPACE	NP-hard/in PSPACE
q is relational	NP-complete	NP-complete
q is linear	NP-complete	polyn.

Combination of guessing and verification \rightsquigarrow *backtracking algorithms*

Finding *max*-rewritings

77

We consider *conjunctive* queries and views.

Theorem: *Suppose, query and views don't have comparisons.*

- *Existence of both, complete and partial rewritings, is NP-complete.*
- *Both problems are still NP-complete, even if queries and views don't have repeated predicates.*

Three reasoning problems for aggregate queries:

equivalence, containment, rewriting using views

- Characterisations of equivalence
- Reduction of containment to equivalence for expandable aggregation functions
- Algorithms for rewriting aggregate queries using views
 - based on classes of “candidate queries”
 - terminate for decidable subcases