A Query Rewriting Algorithm: Exact Minimization of # of Joins

- Example (movie database)

select m1.director
from movie m1, movie m2, movie m3, schedule s1, schedule s2
where m1.director = m2.director and m2.actor = m3.actor
and m1.title = s1.title and m3.title = s2.title

Note: number of joins in corresponding algebra expression
is (number of tuples in FROM clause) – 1

Goal: minimize the number of tuples in the FROM clause
aka join minimization

Exact Minimization of # of Joins

- Example (movie database)

select m1.director
from movie m1, movie m2, movie m3, schedule s1, schedule s2
where m1.director = m2.director and m2.actor = m3.actor
and m1.title = s1.title and m3.title = s2.title

Can this be simplified?

Exact Minimization of # of Joins

- Example (movie database)

select m1.director
from movie m1, movie m2, movie m3, schedule s1, schedule s2
where m1.director = m2.director and m2.actor = m3.actor
and m1.title = s1.title and m3.title = s2.title

More intuitive representation:

movie	title	director	actor
m1	t	d	
m2		d	a
m3	y		a

schedule	theater	title
s1		t
s2		y

71

Exact Minimization of # of Joins

- Example (movie database)

Can this be simplified?

Claim: it is enough to keep m1 and s1 in the pattern

Reason: m1.actor can play the role of a

t can play the role of y

movie	title	director	actor
m1	t	d	
m2		d	a
m3	y		a

schedule	theater	title
s1		t
s2		y

72

Exact Minimization of # of Joins

- Example (movie database)

  Can this be simplified?

  Claim: it is enough to keep m1 and s1 in the pattern

  Reason: m1.actor can play the role of a

  t can play the role of y

movie	title	director	actor
m1	t	d	

schedule	theater	title
s1		t

73

Exact Minimization of # of Joins

- Example (movie database)

  Simplified SQL query:

```
select m1.director
from movie m1, schedule s1
where m1.title = s1.title
```

movie	title	director	actor
m1	t	d	

schedule	theater	title
s1		t

74

Exact Minimization of # of Joins

4 joins

select m1.director
from movie m1, movie m2, movie m3, schedule s1, schedule s2
where m1.director = m2.director and m2.actor = m3.actor
and m1.title = s1.title and m3.title = s2.title

1 join

select m1.director
from movie m1, schedule s1
where m1.title = s1.title

Exact Minimization of # of Joins

Another example: using constraints (aka *semantic optimization*)

"Find theaters showing a title by Berto and a title in which Winger acts"

select s1.theater
from schedule s1, schedule s2, movie m1, movie m2
where s1.theater = s2.theater and s1.title = m1.title and
m1.director = 'Berto' and s2.title = m2.title and
m2.actor = 'Winger'

movie	title	director	actor
m1	~~x~~	~~Berto~~	
m2	~~y~~		~~Winger~~

schedule	theater	title
s1	~~t~~	~~x~~
s2	~~t~~	~~y~~

Exact Minimization of # of Joins

Another example: using constraints

"Find theaters showing a title by Berto and a title in which Winger acts"

Suppose each title has only one director and each theater shows only one title

Then x = y and m2.director = 'Berto'

movie	title	director	actor
m1	x	Berto	
m2	y		Winger

schedule	theater	title
s1	t	x
s2	t	y

Exact Minimization of # of Joins

Another example: using constraints

"Find theaters showing a title by Berto and a title in which Winger acts"

Suppose each title has only one director and each theater shows only one title

Then x = y and m2.director = 'Berto'

movie	title	director	actor
m1	x	Berto	
m2	x	Berto	Winger

schedule	theater	title
s1	t	x
s2	t	x

Exact Minimization of # of Joins

Another example: using constraints

"Find theaters showing a title by Berto and a title in which Winger acts"

Suppose each title has only one director and each theater shows only one title

Then x = y and m2.director = 'Berto'

movie	title	director	actor
m2	~~x~~	~~Berto~~	~~Winger~~

schedule	theater	title
s1	~~t~~	~~x~~

Exact Minimization of # of Joins

Another example: using constraints

"Find theaters showing a title by Berto and a title in which Winger acts"

select s1.theater
from schedule s1, movie m2
where s1.title = m2.title and m2.director = 'Berto'
and m2.actor = 'Winger'

movie	title	director	actor
m2	~~x~~	~~Berto~~	~~Winger~~

schedule	theater	title
s1	~~t~~	~~x~~

Exact Minimization of # of Joins

3 joins

```
select s1.theater
from schedule s1, schedule s2, movie m1, movie m2
where s1.theater = s2.theater and s1.title = m1.title and
      m1.director = 'Berto' and s2.title = m2.title and
      m2.actor = 'Winger'
```

1 join

```
select s1.theater
from schedule s1, movie m2
where s1.title = m2.title and  m2.director = 'Berto'
      and m2.actor = 'Winger'
```


Exact Minimization of # of Joins

How do redundant joins arise?

- Complex queries written by humans
  especially on large schemas with constraints
- Queries resulting from view unfolding
  see next example
- Very complex SQL queries generated by tools

Example: Join redundancies from view unfolding

Database:

Patient	pid hospital docid

Doctor	docid docname

View (Scripps doctors):

```
create view ScrippsDoc as
select  d1.* from Doctor d1, Patient p1
where p1.hospital = 'Scripps' and p1.docid = d1.docid
```

View (Scripps patients):

```
create view ScrippsPatient as
select  p2.* from Patient p2
where p2.hospital = 'Scripps'
```

Scripps query (using views):

```
select p.pid, d.docname
from ScrippsPatient p, ScrippsDoc d
where p.docid = d.docid
```


Query on database obtained by view unfolding

query using view

```
select p.pid, d.docname
from ScrippsPatient p, ScrippsDoc d
where p.docid = d.docid
```

view1

```
create view ScrippsDoc as
select  d1.* from Doctor d1, Patient p1
where p1.hospital = 'Scripps' and p1.docid = d1.docid
```

view2

```
create view ScrippsPatient as
select  p2.* from Patient p2
where p2.hospital = 'Scripps'
```

result of view unfolding

```
select p.pid, d.docname
from Patient p, Doctor d, Patient p1
where p.docid = d.docid and p.hospital = 'Scripps'
and p1.hospital = 'Scripps' and p1.docid = d.docid
```

Resulting query has a redundant join

```
select p.pid, d.docname
from Patient p, Doctor d, Patient p1
where p.docid = d.docid and p.hospital = ‘Scripps’
and p1.hospital = ‘Scripps’ and p1.docid = d.docid
```

Patient	pid	hospital	docid
p	~~j~~	~~‘Scripps’~~	~~i~~
p1		~~‘Scripps’~~	~~i~~

redundant

Doctor	docid	docname
d	~~i~~	~~n~~

answer	pid	docname
	~~j~~	~~n~~

85

Patient	pid	hospital	docid
p	~~j~~	~~‘Scripps’~~	~~i~~

Doctor	docid	docname
d	~~i~~	~~n~~

answer	pid	docname
	~~j~~	~~n~~

Minimized SQL query has one join:

```
Select p.pid, d.docname
From Patient p, Doctor d
Where p.hospital = ‘Scripps’ and
      p.docid = d.docid
```

86

Exact Minimization of # of Joins

Minimization algorithm for conjunctive SQL queries:

SQL query whose **where** clause is
a conjunction of equalities

Basic idea:

$$\begin{tikzcd} \text{SQL query} \arrow[d] & & \text{simplified SQL query} \\ \text{QBE pattern} \arrow[rr, "\text{minimize}"'] & & \text{QBE pattern} \arrow[u] \end{tikzcd}$$

87

QBE patterns

Same as QBE, except (for better readability):

- no underscore to mark variables
- wildcards are explicitly denoted by "–" instead of blank
- no insert I. in the answer relation

movie	title	director	actor
	t	d	–
	–	d	a
	y	–	a

schedule	theater	title
	–	t
	–	y

answer	director
	d

88

From SQL conjunctive queries to QBE patterns

1. Rewrite the SQL query using tuple variables
2. For each tuple variable in the **from** clause aliasing relation R insert a corresponding QBE row in R
3. Use repeated variables and constants to express the equalities in the **where** clause

   if a contradiction arises (two different constants are made equal) then output ∅
4. The QBE answer relation contains a row whose variables occur in the coordinates specified in the **select** clause
5. Coordinates not involved in any equality and not in the answer are wildcards

From SQL conjunctive queries to QBE patterns

- Example

```
select  m1.director
from    movie m1, movie m2, movie m3, schedule s1, schedule s2
where   m1.director = m2.director and m2.actor = m3.actor
           and m1.title = s1.title and m3.title = s2.title
```

movie	title	director	actor

schedule	theater	title

answer	director

From SQL conjunctive queries to QBE patterns

- Example

select m1.director
from movie m1, movie m2, movie m3, schedule s1, schedule s2
where m1.director = m2.director and m2.actor = m3.actor
and m1.title = s1.title and m3.title = s2.title

movie	title	director	actor
m1	t	d	–
m2	–	d	a
m3	y	–	a

schedule	theater	title
s1	–	t
s2	–	y

answer	director
	d

91

From SQL conjunctive queries to QBE patterns

- Example

select m1.director
from movie m1, movie m2, movie m3, schedule s1, schedule s2
where m1.director = m2.director and m2.actor = m3.actor
and m1.title = s1.title and m3.title = s2.title

movie	title	director	actor
	t	d	–
	–	d	a
	y	–	a

schedule	theater	title
	–	t
	–	y

answer	director
	d

92

From SQL conjunctive queries to QBE patterns

- Another example

```
select s1.theater
from schedule s1, schedule s2, movie m1, movie m2
where s1.theater = s2.theater and s1.title = m1.title and
        m1.director = 'Berto' and s2.title = m2.title and
        m2.actor = 'Winger'
```

movie	title	director	actor

schedule	theater	title

answer	theater

From SQL conjunctive queries to QBE patterns

- Another example

```
select s1.theater
from schedule s1, schedule s2, movie m1, movie m2
where s1.theater = s2.theater and s1.title = m1.title and
        m1.director = 'Berto' and s2.title = m2.title and
        m2.actor = 'Winger'
```

movie	title	director	actor
m1	x	Berto	–
m2	y	–	Winger

schedule	theater	title
s1	t	x
s2	t	y

answer	theater
	t

From SQL conjunctive queries to QBE patterns

- Another example

```
select s1.theater
from schedule s1, schedule s2, movie m1, movie m2
where s1.theater = s2.theater and s1.title = m1.title and
        m1.director = 'Berto' and s2.title = m2.title and
        m2.actor = 'Winger'
```

movie	title	director	actor
	x	Berto	—
	y	—	Winger

schedule	theater	title
	t	x
	t	y

answer	theater
	t

95

- Query defined by a pattern: all answers obtained by matching the pattern into the database (just like QBE)

schedule	theater	title
	Hillcrest	Tango
	Paloma	Godfather
	…	….

schedule	theater	title
	—	t
	—	y

movie	title	director	actor
	Tango	Berto	Brando
	Godfather	Coppola	Brando
	…	…	…

movie	title	director	actor
	t	d	—
	—	d	a
	y	—	a

answer	director
	Berto

answer	director
	d

96

- Query defined by a pattern: all answers obtained by matching the pattern into the database (just like QBE)

schedule	theater	title
	Hillcrest	Tango
	Paloma	Godfather
	…	….

schedule	theater	title
	—	t
	—	y

movie	title	director	actor
	Tango	Berto	Brando
	Godfather	Coppola	Brando
	…	…	…

movie	title	director	actor
	t	d	—
	—	d	a
	y	—	a

answer	director
	Coppola

answer	director
	d

Pattern Minimization

- Back to example:

movie	title	director	actor
m1	t	d	—
m2	—	d	a
m3	y	—	a

schedule	theater	title
s1	—	t
s2	—	y

answer	director
	d

Intuition: rows m2, m3, s2 are redundant

Why: if m1 and s1 are present, then the entire pattern is satisfied

Pattern Minimization

- Back to example:

movie	title	director	actor
m1	t	d	–
m2	–	d	a
m3	y	–	a

schedule	theater	title
s1	–	t
s2	–	y

answer	director
	d

Intuition: rows m2, m3, s2 are redundant

More precisely: m2, m3 can be "mapped" to m1, and s2 to s1
a → m1.actor y → t

Pattern Minimization

Formally: pattern folding (aka *homomorphism*)
mapping f on variables, constants and wildcards of pattern P such that

- f(x) = x for answer variables and constants x
- every row of P is mapped to an existing row of P in the same relation

movie	title	director	actor
m1	t	d	–
m2	–	d	a
m3	y	–	a

schedule	theater	title
s1	–	t
s2	–	y

answer	director
	d

f(a) = m1.actor = "–"
f(m2.title) = t; f(y) = t
f(m3.director) = d
f(s2.theater) = s1.theater = "–"
f(d) = d

Pattern Minimization

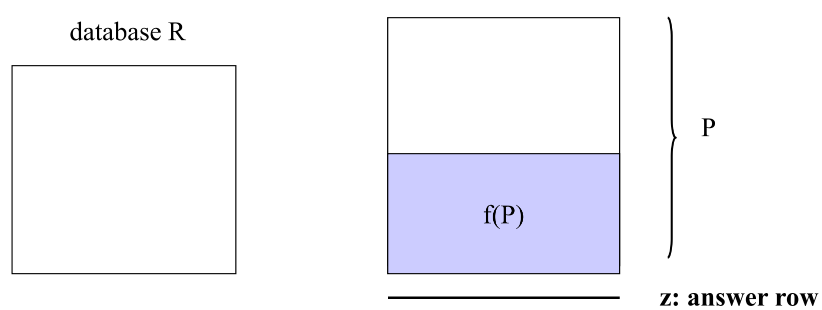
Formally: pattern folding (aka *homomorphism*)
mapping f on variables, constants and wildcards of pattern P such that

- f(x) = x for answer variables and constants x
- every row of P is mapped to an existing row of P in the same relation

Theorem: if P is a pattern and f is a folding of P
then f(P) defines the same query as P

101

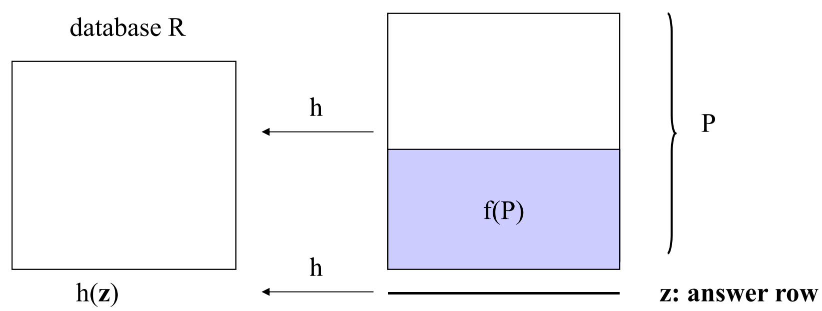
Theorem: P and f(P) define the same query.
Proof idea



Must show that answer of P on R = answer of f(P) on R

102

Theorem: P and f(P) define the same query.
Proof idea

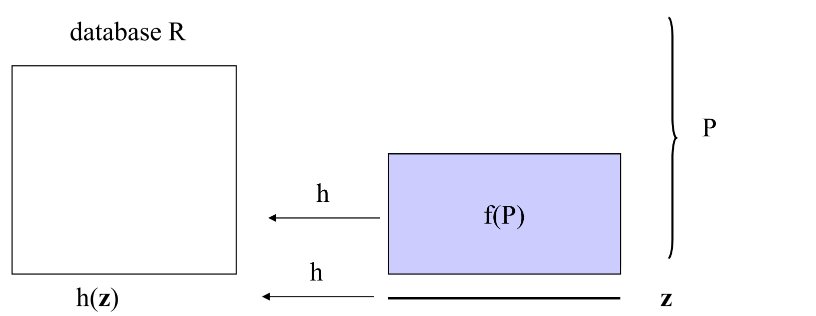


answer of P on R $\subseteq$ answer of f(P) on R

Since f(P) $\subseteq$ P, any matching h of P in database R is also a matching of f(P)

103

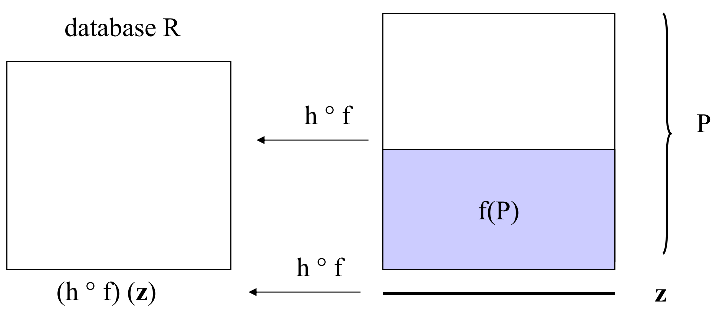
Theorem: P and f(P) define the same query.
Proof idea



answer of f(P) on R $\subseteq$ answer of P on R

104

Theorem: P and f(P) define the same query.
Proof idea



answer of f(P) on R $\subseteq$ answer of P on R

If h is a matching of f(P) in R then h ° f is a matching of P in R and (h ° f) (**z**) = h (f(**z**)) = h(**z**).

105

Pattern minimization algorithm

Repeatedly eliminate redundant rows
Row r is redundant if there is a folding f of P such that r $\notin$ f(P)

	movie	title	director	actor
		t	d	–
redundant		–	d	a
redundant		y	–	a

	schedule	theater	title
		–	t
redundant		–	y

answer	director
	d

106

Pattern minimization algorithm

Repeatedly eliminate redundant rows
Row r is redundant if there is a folding f of P such that r $\notin$ f(P)

movie	title	director	actor
	t	d	—

schedule	theater	title
	—	t

answer	director
	d

Minimal pattern

107

Another example

R	A	B	C
	a	b_1	c_1
	--	b	c_1
	a	b_2	--
	a_2	b_2	c
	a_2	b_1	c

answer	A	B	C
	a	b	c

Another example

R	A	B	C
	a	b_1	c_1
	--	b	c_1
	a	b_2	--
	a_2	b_2	c
	a_2	b_1	c

answer	A	B	C
	a	b	c

- By trial and error, we can eliminate rows 3 and 4 using the folding mapping b_2 to b_1 and leaving all other variables unchanged.

R	A	B	C
	a	b_1	c_1
	--	b	c_1
	a_2	b_1	c

answer	A	B	C
	a	b	c

minimized pattern

From minimized pattern back to SQL query

Example:

R	A	B	C
t1	a	b_1	c_1
t2	--	b	c_1
t3	a_2	b_1	c

answer	A	B	C
	a	b	c

select t1.A, t2.B, t3.C
from R t1, R t2, R t3
where t1.B = t3.B **and** t1.C = t2.C

110

A complete example

R: ABC

SQL conjunctive query:

select t1.A, t2.B, t3.C
from R t1, R t2, R t3
where t2.A = t3.A **and** t1.B = 5 **and**
t2.B = t3.B **and** t2.B = t1.B

Pattern:

R	A	B	C
t1	a	5	--
t2	a_1	5	--
t3	a_1	5	c

answer	A	B	C
	a	5	c

Minimized pattern:

R	A	B	C
t1	a	5	--
t3	a_1	5	c

answer	A	B	C
	a	5	c

Minimized SQL query:

select t1.A, 5 **as** B, t3.C
from R t1, R t3
where t1.B = 5 **and** t3.B = 5

Theorem: the minimization algorithm produces an SQL query with minimum number of joins among all conjunctive SQL queries equivalent to the original one on all databases.

But we can do even better: take into account constraints (semantic query optimization). To see how this works, we extend the algorithm with functional dependencies.

Data Dependencies (aka constraints)

- Statements about valid data
  - Keys
    "SSN uniquely determines all attributes of *employee*"
  - Referential integrity
    "Every student is a person"
  - Functional dependencies: extension of keys
    "Each employee works in no more than one department"
    NAME → DEPARTMENT
- Use of dependencies:
  - check data integrity
  - query optimization
  - schema design → "normal forms"

113

Functional Dependencies

- Generalization of key constraints

employee	*ssn*	*name*	*city*	*zip-code*	*state*

primary key: ssn ``ssn determines all other attributes``
ssn → name city zip-code state

more generally: some attributes may determine other attributes without being keys: zip-code → state

114

Functional Dependencies

- Functional dependency on R: expression $X \rightarrow Y$ where $X, Y \subseteq att(R)$
- An instance of R satisfies $X \rightarrow Y$ iff
  whenever two tuples agree on X, they also agree on Y

e.g.

SCHEDULE	THEATER	TITLE
	la jolla	killer tomatoes
	hillcrest	tango

Satisfies THEATER → TITLE

SCHEDULE	THEATER	TITLE
	la jolla	killer tomatoes
	hillcrest	tango
	hillcrest	splendor

Violates THEATER → TITLE
satisfies TITLE → THEATER

115

Using FDs in query optimization

Example revisited: suppose title → director, theater → title

"Find theaters showing a title by Berto and a title in which Winger acts"

select s1.theater
from schedule s1, schedule s2, movie m1, movie m2
where s1.theater = s2.theater and s1.title = m1.title and
m1.director = 'Berto' and s2.title = m2.title and
m2.actor = 'Winger'

movie	title	director	actor
m1	x	Berto	–
m2	y	–	Winger

schedule	theater	title
s1	t	x
s2	t	y

answer	theater
	t

116

movie	title	director	actor
m1	x	Berto	–
m2	y	–	Winger

schedule	theater	title
s1	t	x
s2	t	y

answer	theater
	t

This pattern is minimal. However, we know that title → director, theater → title. Since the database satisfies theater → title, x = y in every matching. Next, since title → director and x = y, m1.director = m2.director = 'Berto'. We obtain the following pattern:

movie	title	director	actor
m1	x	Berto	–
m2	x	Berto	Winger

schedule	theater	title
s1	t	x

answer	theater
	t

117

movie	title	director	actor
m1	x	Berto	–
m2	x	Berto	Winger

schedule	theater	title
s1	t	x

answer	theater
	t

Minimized pattern:

movie	title	director	actor
m2	x	Berto	Winger

schedule	theater	title
s1	t	x

answer	theater
	t

118

- In general: can simplify pattern P if the database satisfies a set F of FDs.
- Algorithm: The Chase
  - Input: pattern P, a set F of FDs
  - Output: pattern $\text{CHASE}_F(P)$
    equivalent to P on all relations satisfying F

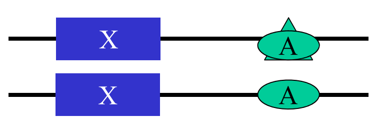
Intuition: the chase modifies P so that it satisfies all FDs in F

Note: assume without loss of generality
that FDs in F are of the form $X \rightarrow A$ where A is one attribute

119

Basic chase step with X → A

If pattern contains two rows that agree on X and disagree on A,
change them so that they also agree on A



120

The Chase in detail

- Repeat until no change
  - For each $X \to A$ in F do
    - For all rows t_1, t_2 in P such that $t_1(X) = t_2(X)$, $t_1(A) \neq t_2(A)$ do
      - if $t_1(A)$, $t_2(A)$ are non-answer variables then replace one by the other everywhere in P
      - if $t_1(A)$ is a non-answer variable and $t_2(A)$ is a wildcard, then replace $t_2(A)$ by $t_1(A)$ everywhere in P
      - If $t_1(A)$, $t_2(A)$ are wildcards, replace both with a new variable
      - if $t_1(A)$ is an answer variable and $t_2(A)$ is a variable or wildcard, then replace $t_2(A)$ by $t_1(A)$ everywhere in P
      - if $t_1(A)$ is constant, $t_2(A)$ is variable or wildcard, then replace $t_2(A)$ by $t_1(A)$ everywhere in P
      - if $t_1(A)$ is constant, $t_2(A)$ is constant then STOP and output $\varnothing$

121

Optimization of SQL conjunctive queries with FDs

- Input: SQL conjunctive query Q, set of FDs F
  - build the pattern P of Q
  - compute $CHASE_F(P)$
  - minimize $CHASE_F(P)$
  - construct the SQL query corresponding to the minimal pattern

Claim: the above produces an SQL query equivalent to Q on all databases satisfying F, that has the minimum possible number of joins

122

Example

select t1.A, t1.B, t2.C **from** R t1, R t2
where t1.A = 5 and t1.B = t2.B

R:ABC satisfies B → A

1. Pattern:

R	A	B	C
t1	5	b	--
t2	--	b	c

answer	A	B	C
	5	b	c

2. Chase with B → A:

R	A	B	C
t1	5	b	--
t2	5	b	c

answer	A	B	C
	5	b	c

3. Minimize:

R	A	B	C
	5	b	c

answer	A	B	C
	5	b	c

select * **from** R
where A = 5

Example

select t2.A, t2.B, t1.C
from R t1, R t2
where t1.A = 5 and t2.A = 6 and t1.B = t2.B

R: ABC satisfies
B → A

1. Pattern:

R	A	B	C
t1	5	b	c
t2	6	b	--

answer	A	B	C
	6	b	c

2. Chase with B → A: $5 \neq 6$ so the result is empty

3. Minimized query: $\varnothing$

Example

select t1.A, t3.B
from R t1, R t2, R t3, R t4
where t1.A = t2.A and t2.A = t4.A and t1.B = t3.B and t4.B = 5 and t2.C = t3.C

R: ABC satisfies A → B

1. Pattern:

R	A	B	C
t1	a	b	--
t2	a	--	c
t3	--	b	c
t4	a	5	--

answer	A	B
	a	b

2. Chase with A → B:

R	A	B	C
	a	5	c
	--	5	c

answer	A	B
	a	5

3. Minimize:

R	A	B	C
	a	5	c

select A, 5 **as** B **from** R
where B = 5

125