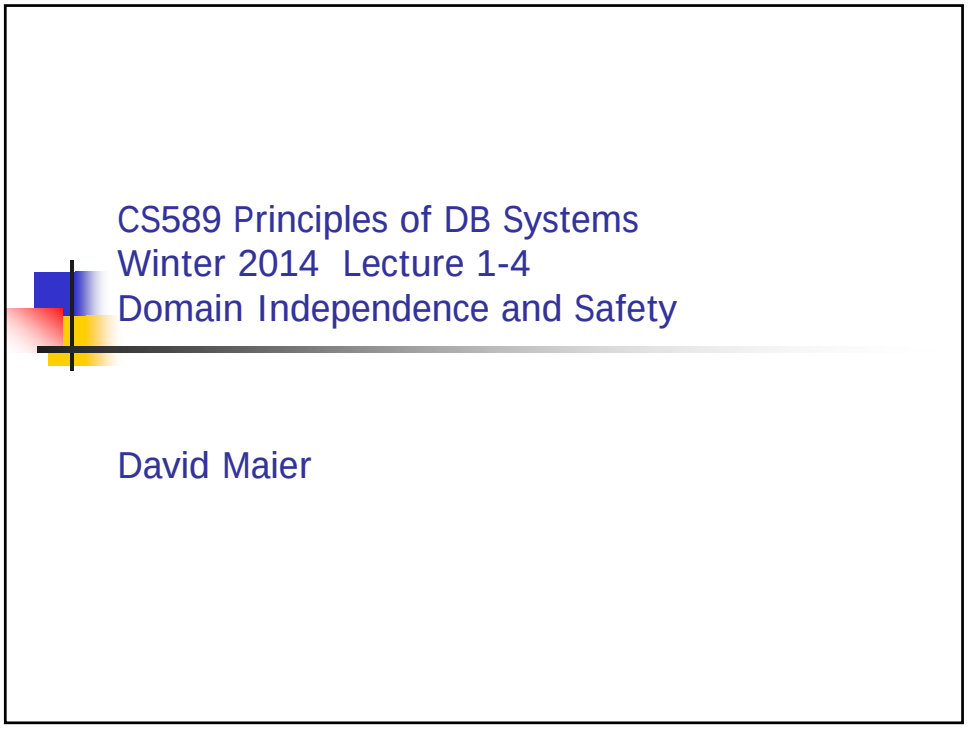

CS589 Principles of DB Systems
Winter 2014 Lecture 1-4
Domain Independence and Safety
David Maier

Goals for this lecture
Introduce the problems that can occur in calculus queries and Datalog .
Define syntactic restrictions to avoid the problems
Domain independence (for calculus)
Safety (for Datalog)
The restricted languages are equivalent to relational algebra
We could extend relational algebra instead, but would require infinite relations
CS 589 Principles of DB Systems, Spring 2016 © David Maier
2

What's the problem with calculus?

Student(name, age)
Teacher(name, age)

{n:name, a:age | ¬Student(n, a) }

{n:name, a:age | $\exists n_1(\exists a_1(\text{Student}(n,a_1) \vee \text{Teacher}(n_1,a)))$}

Are these valid well-formed formulas?
Do we want to consider these to be queries?
What would the query answer be?
Similar problems with tuple calculus

What's the query answer?

Student(name, age)
Teacher(name, age)

{n:name, a:age | ¬Student(n, a) }

Query answer depends on the domain for n and a.

DOM(name) is the set of possible names
DOM(age) is the set of possible ages

If domains are infinite, query answer is not a relation because relations must be finite.

If domains are finite, then changing the domain changes the query answer: char(20) → char(28)

What's the query answer?

{n:name, a:age | $\exists n_1(\exists a_1(\text{Student}(n,a_1) \vee \text{Teacher}(n_1,a)))$}

What's wrong here?

If Student(n, a_1) is true and Teacher(n_1,a) is false, the formula is true. But what is the value for a?

And similarly, if Teacher (n_1,a) is true and Student(n, a_1) is false, what is the value for n?

Domain dependence

In both cases, values will depend on the domains.

Domain Independence

(informal definition) A query is domain independent if for each database instance, the answer to the query does not change if the domain changes.

The two main problems that cause domain dependence:

- ¬F
  - where some variable in F is not restricted elsewhere in the query
- F1 ∨ F2
  - such that they have different free variables

Determining whether a query is domain independent is *undecidable*.

So...we first define "positive" and "negative" occurrences of a variable

provides a binding — no fixed binding

Positive occurrences of a variable (iff only one of these hold)	Negative occurrences of a variable (iff only one of these hold)
$R(y_1, y_2, \ldots, x, \ldots, y_k)$	x does not appear in formula F at all
x=v where v is a constant	x=y where y is a variable (or y = x)
$\neg F$ if x occurs negatively in F	$\neg F$ if x occurs positively in F
$F_1 \wedge F_2$ if x occurs positively in F_1 or positively in F_2	$F_1 \wedge F_2$ if x occurs negatively in F_1 and negatively in F_2
$F_1 \vee F_2$ if x occurs positively in F_1 and positively in F_2	$F_1 \vee F_2$ if x occurs negatively in F_1 or negatively in F_2
$F_1 \rightarrow F_2$ if x occurs negatively in F_1 and positively in F_2	$F_1 \rightarrow F_2$ if x is positive in F_1 and negative in F_2
$\exists y{:}A$ (F) if x is not = y and x occurs positively in F	$\forall y{:}A$ (F) if x is not = y and x occurs negatively in F

not the same variable as

What are Positive and Negative Variables?

Student(name, age)

Teacher(name, age)

$\exists n_1(\exists a_1(\text{Student}(n,a_1) \wedge \text{Teacher}(n_1,a)))$

$\exists n_1(\exists a_1(\text{Student}(n,a_1) \vee \text{Teacher}(n_1,a)))$

$\forall a(\neg\text{Student}(n, a) \vee \exists n_1(\text{Teacher}(n_1,a)))$

$\neg\text{Student}(n, a)$

$\exists y(\text{Student}(n, a) \wedge a = y)$

Syntatic restriction: "Allowed" domain calculus queries

- A domain calculus query is allowed if its formula is allowed.
- A domain calculus formula F is allowed if all of these conditions hold:
  - Every free variable in F occurs positively in F (What are the free variables?)
  - For every subformula $\exists x{:}A\ (G)$ of F, x is positive in G
  - For every subformula $\forall x{:}A\ (G)$ of F, x is negative in G

Which are Allowed Queries?

Student(name, age)
Teacher(name, age)

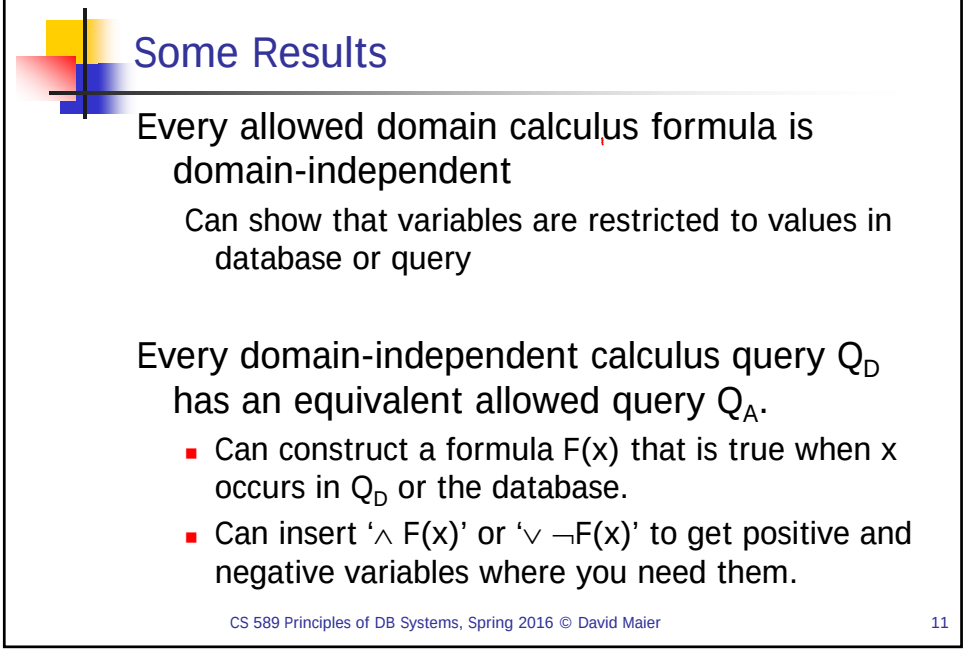
✓ $\{n{:}name, a{:}age \mid \exists n_1(\exists a_1(Student(n,a_1) \wedge Teacher(n_1,a)))\}$

✗ $\{n{:}name, a{:}age \mid \exists n_1(\exists a_1(Student(n,a_1) \vee Teacher(n_1,a)))\}$

✗ $\{n{:}name \mid \forall a(\neg Student(n, a) \vee \exists n_1(Teacher(n_1,a)))\}$

✗ $\{n{:}name, a{:}age \mid \neg Student(n, a)\ \}$

$\{n{:}name, a{:}age \mid \exists y(Student(n, a) \wedge a = y\ \}$ Domain Independent



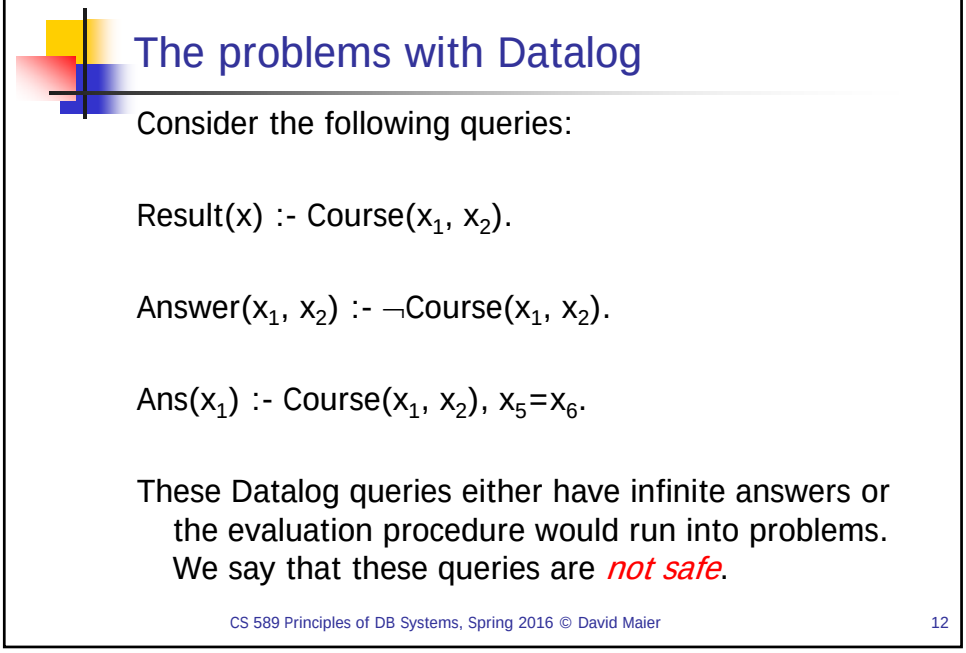
Some Results

Every allowed domain calculus formula is domain-independent

Can show that variables are restricted to values in database or query

Every domain-independent calculus query Q_D has an equivalent allowed query Q_A.

- Can construct a formula F(x) that is true when x occurs in Q_D or the database.
- Can insert '$\wedge$ F(x)' or '$\vee \neg$F(x)' to get positive and negative variables where you need them.



The problems with Datalog

Consider the following queries:

Result(x) :- Course(x_1, x_2).

Answer(x_1, x_2) :- $\neg$Course(x_1, x_2).

Ans(x_1) :- Course(x_1, x_2), x_5=x_6.

These Datalog queries either have infinite answers or the evaluation procedure would run into problems. We say that these queries are *not safe*.

So, we define positive and negative variables and safe Datalog rules

A Datalog rule C is safe if all the variables appearing in the literals of C (both head and body) occur positively in C.

A Datalog program is safe if all the rules are safe.

A variable x occurs positively if x appears in:

- $R(y_1, y_2, \ldots, x, \ldots, y_k)$ where this is a positive literal in the body of the rule
- x=v where v is a constant (in the body of the rule)
- x=y or y=x where y is a variable that occurs positively in C (in the body of the rule)

Comments

- In a safe Datalog rule, all the variables in the head of the rule must occur in one or more literals in the body.
- All the variables that occur in a negative literal in the body of a safe Datalog rule, must occur positively in one or more atomic formulae in its body.
- Safe Datalog query → query answer is always finite.
- Allowable domain calculus query → query answer is always finite.