

## Lecture 16: Relational Calculus and Logic

### Logic and Databases

*From relational algebra to relational calculus, and other logic-database connections.*

- relational calculus
- logic
- syntax and semantics of first order logic
- parallels between first order logic and relational databases

Read [E&N §6.6, §6.7, §8.5.4] for more on today's stuff.

## Procedural vs Declarative

Last week we saw relational algebra. Relational *calculus* is another formal database language, more like logic than algebra.

## Procedural vs Declarative

Last week we saw relational algebra. Relational *calculus* is another formal database language, more like logic than algebra.

That is, relational calculus expressions say *what* you want, rather than *how* to find out.

## Procedural vs Declarative

Last week we saw relational algebra. Relational *calculus* is another formal database language, more like logic than algebra.

That is, relational calculus expressions say *what* you want, rather than *how* to find out.

Actually, its not black and white. Consider the differences between the following 3 equivalent queries.

## Three different ways

### **SQL**

```
SELECT bdate, address  
FROM employee  
WHERE salary > 50000
```

# Three different ways

## SQL

```
SELECT bdate, address  
FROM employee  
WHERE salary > 50000
```

## relational algebra

$$\pi_{bdate, address}(\sigma_{salary > 50000}(employee))$$

# Three different ways

## SQL

```
SELECT bdate, address  
FROM employee  
WHERE salary > 50000
```

## relational algebra

$$\pi_{bdate, address}(\sigma_{(salary > 50000)}(employee))$$

## relational calculus (see [E&N, §6.6.1])

$$\{t.fname, t.address \mid employee(t) \wedge t.salary > 50000\}$$

## Relational Calculus Observations

$\{t.fname, t.address \mid employee(t) \wedge t.salary > 50000\}$

- it is based on “set-builder” notation

## Relational Calculus Observations

$$\{t.fname, t.address \mid employee(t) \wedge t.salary > 50000\}$$

- it is based on “set-builder” notation
- variable  $t$  stands for a tuple

## Relational Calculus Observations

$$\{t.fname, t.address \mid employee(t) \wedge t.salary > 50000\}$$

- it is based on “set-builder” notation
- variable  $t$  stands for a tuple
- attribute names are functions, writing  $t.fname$  for  $fname(t)$

## Relational Calculus Observations

$$\{t.fname, t.address \mid employee(t) \wedge t.salary > 50000\}$$

- it is based on “set-builder” notation
- variable  $t$  stands for a tuple
- attribute names are functions, writing  $t.fname$  for  $fname(t)$
- relation names are predicates

## Relational Calculus Observations

$$\{t.fname, t.address \mid employee(t) \wedge t.salary > 50000\}$$

- it is based on “set-builder” notation
- variable  $t$  stands for a tuple
- attribute names are functions, writing  $t.fname$  for  $fname(t)$
- relation names are predicates

# Relational Calculus Observations

$$\{t.fname, t.address \mid employee(t) \wedge t.salary > 50000\}$$

- it is based on “set-builder” notation
- variable  $t$  stands for a tuple
- attribute names are functions, writing  $t.fname$  for  $fname(t)$
- relation names are predicates

To make proper sense of this, we need to understand a bit about formal logic.

# Logic101 in Ten Minutes

## Logic

- What is Logic?
- History of logic and computation
- Main ideas of logic:
  - statement (syntax)
  - situation (semantics)
  - truth
  - argument
  - validity
  - proof
- Syntax and semantics of first order logic (but not deduction)

# What is Logic?

Logic is the study of good reasoning.

# What is Logic?

Logic is the study of good reasoning.

Is the reasoning in the following two examples *good*?

Why, or why not?

## Example 1

The drums are louder than the bass.

The guitar is louder than the drums.

Therefore, the guitar is louder than the bass.

## Example 2

I told the bass player to turn it down or I'd punch his head in.

Therefore, the guitar is louder than the bass.

# What is Logic?

Logic is the study of good reasoning.

Is the reasoning in the following two examples *good*?

Why, or why not?

## Example 1

The drums are louder than the bass.

The guitar is louder than the drums.

Therefore, the guitar is louder than the bass.

## Example 2

I told the bass player to turn it down or I'd punch his head in.

Therefore, the guitar is louder than the bass.

Logic seeks theories of good reasoning, to give answers and explanations to questions like these.

## Logic provides tools for precise work with ideas

- Logic came from philosophers trying to understand, explain and even improve the certainty of mathematical arguments.

## Logic provides tools for precise work with ideas

- Logic came from philosophers trying to understand, explain and even improve the certainty of mathematical arguments.
- Philosophers use formal logic to analyse concepts into more basic ideas, and solve problems caused by the confusion of English.

## Logic provides tools for precise work with ideas

- Logic came from philosophers trying to understand, explain and even improve the certainty of mathematical arguments.
- Philosophers use formal logic to analyse concepts into more basic ideas, and solve problems caused by the confusion of English.

### Example 3

Nothing is better than a holiday in St Tropez.

A ham sandwich is better than nothing.

Thus, a ham sandwich is better than a holiday in St Tropez.

## Logic provides tools for precise work with ideas

- Logic came from philosophers trying to understand, explain and even improve the certainty of mathematical arguments.
- Philosophers use formal logic to analyse concepts into more basic ideas, and solve problems caused by the confusion of English.

### Example 3

Nothing is better than a holiday in St Tropez.

A ham sandwich is better than nothing.

Thus, a ham sandwich is better than a holiday in St Tropez.

- Philosophical logic lead to the idea of reasoning by mechanical symbol manipulation, and thus to the computer, programming languages, specification languages, databases etc.

## Logic provides tools for precise work with ideas

- Proof theory, model theory and recursion theory are now important branches of mathematics, all dealing with aspects of formal logic.

## Logic provides tools for precise work with ideas

- Proof theory, model theory and recursion theory are now important branches of mathematics, all dealing with aspects of formal logic.
- Logic-like formal systems are widely used in theoretical study, such as the  $\lambda$ -calculus in theoretical computer science, and the Lambek-calculus in theoretical linguistics.

## Logic provides tools for precise work with ideas

- Proof theory, model theory and recursion theory are now important branches of mathematics, all dealing with aspects of formal logic.
- Logic-like formal systems are widely used in theoretical study, such as the  $\lambda$ -calculus in theoretical computer science, and the Lambek-calculus in theoretical linguistics.
- Formal logics are now often implemented in computer systems. Reasoning can be fully or partly automated and used to verify hardware designs, safety critical software etc.

## Logic provides tools for precise work with ideas

- Proof theory, model theory and recursion theory are now important branches of mathematics, all dealing with aspects of formal logic.
- Logic-like formal systems are widely used in theoretical study, such as the  $\lambda$ -calculus in theoretical computer science, and the Lambek-calculus in theoretical linguistics.
- Formal logics are now often implemented in computer systems. Reasoning can be fully or partly automated and used to verify hardware designs, safety critical software etc.
- Important mathematical questions have been resolved with the help of computer reasoning: the four colour theorem, and the Kepler conjecture

# Statements, situations and truth

*A statement is a sentence which is true or false, depending on the situation.*

- Questions are not statements  
“Where are my keys?”

# Statements, situations and truth

*A statement is a sentence which is true or false, depending on the situation.*

- Questions are not statements  
“Where are my keys?”
- Exclamations are not statements  
“Oh damn and blast!”

# Statements, situations and truth

*A statement is a sentence which is true or false, depending on the situation.*

- Questions are not statements  
“Where are my keys?”
- Exclamations are not statements  
“Oh damn and blast!”
- Usually, we evaluate statements in the *real world*, but in logic, we must consider other *possible worlds*  
“If I was rich, I would take a holiday in St Tropez.”

# Statements, situations and truth

*A statement is a sentence which is true or false, depending on the situation.*

- Questions are not statements  
“Where are my keys?”
- Exclamations are not statements  
“Oh damn and blast!”
- Usually, we evaluate statements in the *real world*, but in logic, we must consider other *possible worlds*  
“If I was rich, I would take a holiday in St Tropez.”
- Hence, we should not simply say “statement  $P$  is true” but instead “statement  $P$  is true in situation  $w$ ”

# Arguments

*An argument is some statements called premises and a statement called the conclusion.*

- Note that sometimes people say “argument” for the *reasoning*, that is the steps that take you from the premises to the conclusion. In logic, we just mean the premises and the conclusion.

# Arguments

*An argument is some statements called premises and a statement called the conclusion.*

- Note that sometimes people say “argument” for the *reasoning*, that is the steps that take you from the premises to the conclusion. In logic, we just mean the premises and the conclusion.
- An argument might have no premises. That is, by “some” we mean “zero or more.”

# Arguments

*An argument is some statements called premises and a statement called the conclusion.*

- Note that sometimes people say “argument” for the *reasoning*, that is the steps that take you from the premises to the conclusion. In logic, we just mean the premises and the conclusion.
- An argument might have no premises. That is, by “some” we mean “zero or more.”
- Arguments are everywhere, but not usually carefully presented as premises and conclusion. We must determine what are the premises and what is the conclusion. Look for key phrases like “therefore” “it follows that” “hence” ...

# Validity

*An argument is valid if its conclusion is true in every situation where all its premises are true.*

- If a valid argument also has true premises, we say it is *sound*. (ie, premises true *in the real world*)

# Validity

*An argument is valid if its conclusion is true in every situation where all its premises are true.*

- If a valid argument also has true premises, we say it is *sound*. (ie, premises true *in the real world*)
- To show that an argument is *invalid*, we describe a situation that makes all the premises true, and the conclusion false.

# Validity

*An argument is valid if its conclusion is true in every situation where all its premises are true.*

- If a valid argument also has true premises, we say it is *sound*. (ie, premises true *in the real world*)
- To show that an argument is *invalid*, we describe a situation that makes all the premises true, and the conclusion false.
- This is called a *counterexample*.

# Example Counterexample

## Example 2

I told the bass player to turn it down or I'd punch his head in.  
Therefore, the guitar is louder than the bass.

# Example Counterexample

## Example 2

I told the bass player to turn it down or I'd punch his head in.  
Therefore, the guitar is louder than the bass.

The bass player turned it *up*, just to annoy you!

## Example Counterexample

### Example 2

I told the bass player to turn it down or I'd punch his head in.  
Therefore, the guitar is louder than the bass.

The bass player turned it *up*, just to annoy you!

This describes a possible situation where the premise is true, but the conclusion false. The argument is therefore invalid.

# Systems of Formal Logic

*A formal logic has 3 parts: syntax, semantics and deduction.*

- syntax means grammar, it defines what the sentences or *formulae* of the language are

# Systems of Formal Logic

*A formal logic has 3 parts: syntax, semantics and deduction.*

- syntax means grammar, it defines what the sentences or *formulae* of the language are
- semantics is meaning, in formal logic we give a mathematically precise meaning to each of our formulae

# Systems of Formal Logic

*A formal logic has 3 parts: syntax, semantics and deduction.*

- syntax means grammar, it defines what the sentences or *formulae* of the language are
- semantics is meaning, in formal logic we give a mathematically precise meaning to each of our formulae
- deduction is the method of proof, this is a system of strict rules by which we may obtain one formula from some others, these rules are the building blocks of all reasoning in that system

# Systems of Formal Logic

*A formal logic has 3 parts: syntax, semantics and deduction.*

- syntax means grammar, it defines what the sentences or *formulae* of the language are
- semantics is meaning, in formal logic we give a mathematically precise meaning to each of our formulae
- deduction is the method of proof, this is a system of strict rules by which we may obtain one formula from some others, these rules are the building blocks of all reasoning in that system

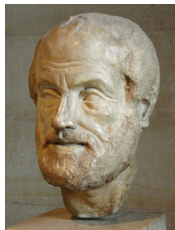
# Systems of Formal Logic

*A formal logic has 3 parts: syntax, semantics and deduction.*

- syntax means grammar, it defines what the sentences or *formulae* of the language are
- semantics is meaning, in formal logic we give a mathematically precise meaning to each of our formulae
- deduction is the method of proof, this is a system of strict rules by which we may obtain one formula from some others, these rules are the building blocks of all reasoning in that system

We want every provable argument to be valid (soundness), and every valid argument to be provable (completeness).

## Aristotle, about 350BC, Greece



- saw the need for a general account of correct reasoning
- codified valid argument forms called “syllogisms”
  - All men are mortal.
  - Aristotle is a man.
  - Therefore Aristotle is mortal.

## Leibniz, about 1680, Germany

“Wonderful idea” - a symbolic language for concepts, allowing problems to be solved by mechanical calculation



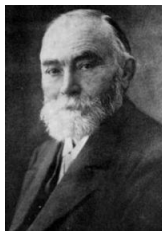
*The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right. (The Art of Discovery 1685, W 51)*

## Boole, about 1840, England



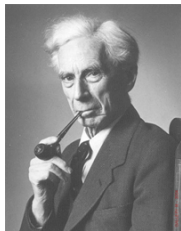
- developed a form of algebra (now called boolean algebra), where letters stand for concepts instead of numbers
- equivalent to our logic PL and elementary set theory
- the beginning of logic as mathematics

## Frege, 1879, Germany



- “Concept Script” (1879) the first formal system to allow multiple quantification (everybody loves somebody)
- strange graphical notation, no longer used
- sought logical foundations for mathematics, eg, definition of number - was Julius Caesar a number?
- almost unknown in his lifetime, and came to think his work useless (see next slide on Russell), now recognised as the father of modern logic

## Russell, 1910, England



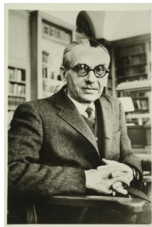
- discovered Russell's paradox "The set of all sets which are not members of themselves" which arises in Frege's system
- with Alfred North Whitehead, wrote the "Principia Mathematica" - in volume II they prove  $1+1=2$
- introduced the theory of types to avoid paradoxes, and theory of descriptions to avoid the problem of referential failure (The king of France is bald)
- arguably the most important philosopher of the 20th century

## Hilbert, 1920, Germany



- Hilbert wanted to save mathematics from the constructivists, who rejected important mathematical methods
- to do this, “Hilbert’s program” aimed to derive all maths from a single axiom system which can prove itself consistent

## Gödel, 1931, Austria



- Gödel proved that all systems capable of expressing arithmetic are either inconsistent or incomplete, and thus can not prove themselves consistent, destroying Hilbert's programme
- that is, any strong deductive system has statements which can neither be proved nor disproved

## Turing, 1936, England



- proposed a class of theoretical computing machine (now called a Turing machine), and showed that one machine of this type could be programmed to do the work of any of them - the *universal* machine
- extended Gödel's work to show that some problems were uncomputable (the Halting problem)
- helped win WWII by breaking German "Enigma" code at Bletchley Park

## Tarski, 1936, Poland



- defined truth
- introduced model theory
- proved that truth for a language can not be defined within that language
- integrated modern logic with modern abstract algebra

## Milner, 1972, England



- introduced a style of computer proof checkers, and theorem provers whose reliability depends only on a very small “core”
- the expressive “higher order logic” and combination of human and computer proof has made formal verification practical
- many important formal/theoretical contributions, including formal systems for concurrent processes ( $\pi$ -calculus)

## Truth Functions, Truth Tables

In classical *propositional logic*, English statements are represented by letters  $A, B, C, \dots$ , and we have *truth-functional* connectives for *and*, *or*, *not*, *if...then...*  $\wedge$ , *vee*,  $\neg$ ,  $\longrightarrow$ .

## Truth Functions, Truth Tables

In classical *propositional logic*, English statements are represented by letters  $A, B, C, \dots$ , and we have *truth-functional* connectives for *and*, *or*, *not*, *if...then...*  $\wedge$ , *vee*,  $\neg$ ,  $\longrightarrow$ .

For example, the meaning of  $\wedge$  is the function

$$\{T, F\} \times \{T, F\} \longrightarrow \{T, F\},$$

$$\{((T, T), T), ((T, F), F), ((F, T), F), ((F, F), F)\}$$

## Truth Functions, Truth Tables

In classical *propositional logic*, English statements are represented by letters  $A, B, C, \dots$ , and we have *truth-functional* connectives for *and*, *or*, *not*, *if...then...*  $\wedge$ , *vee*,  $\neg$ ,  $\longrightarrow$ .

For example, the meaning of  $\wedge$  is the function

$$\{T, F\} \times \{T, F\} \longrightarrow \{T, F\},$$

$$\{((T, T), T), ((T, F), F), ((F, T), F), ((F, F), F)\}$$

We can write this much more conveniently as a *truth-table*

A	B	A	$\wedge$	B
T	T	T	T	T
T	F	T	F	F
F	T	F	F	T
F	F	F	F	F

## Other Major Systems of Logic

Relevant logics Truth-functional *if...then...* is weird: If grass is pink, then I am the prime-minister. Relevant logics study better formal *if...then...*'s.

## Other Major Systems of Logic

**Relevant logics** Truth-functional *if...then...* is weird: If grass is pink, then I am the prime-minister. Relevant logics study better formal *if...then...*'s.

**Modal logics** Add non-truth-functional unary operators that can be read as *sometime in the future...* (temporal logic), *it is possible that...*, *it is permissible to...*, ...

## Other Major Systems of Logic

**Relevant logics** Truth-functional *if...then...* is weird: If grass is pink, then I am the prime-minister. Relevant logics study better formal *if...then...*'s.

**Modal logics** Add non-truth-functional unary operators that can be read as *sometime in the future...* (temporal logic), *it is possible that...*, *it is permissible to...*, ...

**First order logic** English sentences are analysed in terms of *individuals* and the properties or relationships being stated about them. We have *variables* which range over individuals. The variables can be universally (all) or existentially (some) *quantified*.

## Other Major Systems of Logic

- Relevant logics** Truth-functional *if...then...* is weird: If grass is pink, then I am the prime-minister. Relevant logics study better formal *if...then...*'s.
- Modal logics** Add non-truth-functional unary operators that can be read as *sometime in the future...* (temporal logic), *it is possible that...*, *it is permissible to...*, ...
- First order logic** English sentences are analysed in terms of *individuals* and the properties or relationships being stated about them. We have *variables* which range over individuals. The variables can be universally (all) or existentially (some) *quantified*.
- Higher order logic** Whereas the variables of first order logic can only range over individuals, higher order logic allows variables to range over properties and relationships. Its necessary to add a *type system* to the language, to avoid Russell's paradox.

# Syntax and Semantics of First Order Logic

## Logic

- Vocabulary for “real world” problems
- Syntax, more formally
- Free and bound variables
- Formal interpretation of vocabulary
- Valuation of variables
- Semantics of Quantifiers

## First Order Logic: Translation Key

To translate statements into logic, we need a *key*. This tells us

- The *domain*, which is the set of individuals we are talking about. For example, the set of all people.

## First Order Logic: Translation Key

To translate statements into logic, we need a *key*. This tells us

- The *domain*, which is the set of individuals we are talking about. For example, the set of all people.
- *Constants* for any names we want to use. For example *g* for George, *b* for Bill.

## First Order Logic: Translation Key

To translate statements into logic, we need a *key*. This tells us

- The *domain*, which is the set of individuals we are talking about. For example, the set of all people.
- *Constants* for any names we want to use. For example  $g$  for George,  $b$  for Bill.
- *Predicate symbols* and their *arities* for any properties and relationships we want to talk about. For example  $Gx$  for  $x$  is a girl,  $Lxy$  for  $x$  loves  $y$ .

# First Order Logic: Translation Examples

Then we can translate

- “George loves Bill” as  $Lgb$

# First Order Logic: Translation Examples

Then we can translate

- “George loves Bill” as  $Lgb$
- “everybody loves Bill” as  $(\forall x)Lxb$   
 (“For each person  $x$ ,  $x$  loves Bill.”)

# First Order Logic: Translation Examples

Then we can translate

- “George loves Bill” as  $Lgb$
- “everybody loves Bill” as  $(\forall x)Lxb$   
 (“For each person  $x$ ,  $x$  loves Bill.”)
- “All the girls love Bill” as  $(\forall x)(Gx \longrightarrow Lxb)$   
 (“For each person, if they are a girl then they love Bill.”)

# First Order Logic: Translation Examples

Then we can translate

- “George loves Bill” as  $Lgb$
- “everybody loves Bill” as  $(\forall x)Lxb$   
 (“For each person  $x$ ,  $x$  loves Bill.”)
- “All the girls love Bill” as  $(\forall x)(Gx \longrightarrow Lxb)$   
 (“For each person, if they are a girl then they love Bill.”)
- “Somebody loves Bill, but no girls love Bill” as  $(\exists x)(Lxb) \wedge \neg(\exists x)(Gx \wedge Lxb)$

# First Order Logic: Translation Examples

Then we can translate

- “George loves Bill” as  $Lgb$
- “everybody loves Bill” as  $(\forall x)Lxb$   
 (“For each person  $x$ ,  $x$  loves Bill.”)
- “All the girls love Bill” as  $(\forall x)(Gx \longrightarrow Lxb)$   
 (“For each person, if they are a girl then they love Bill.”)
- “Somebody loves Bill, but no girls love Bill” as  $(\exists x)(Lxb) \wedge \neg(\exists x)(Gx \wedge Lxb)$
- “Everybody loves somebody” as  $(\forall x)(\exists y)Lxy$

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )
- An  $n$ -ary predicate followed by  $n$  constants or variables is a formula. (eg.  $Lxb$ )

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )
- An  $n$ -ary predicate followed by  $n$  constants or variables is a formula. (eg.  $Lxb$ )
- If  $A$  and  $B$  are formulae, then so are  $A \wedge B, A \vee B, A \longrightarrow B, \neg A$ .

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )
- An  $n$ -ary predicate followed by  $n$  constants or variables is a formula. (eg.  $Lxb$ )
- If  $A$  and  $B$  are formulae, then so are  $A \wedge B, A \vee B, A \longrightarrow B, \neg A$ .
- If  $A$  is a formula and  $\tau$  is a variable then  $(\forall \tau)A$  and  $(\exists \tau)A$  are formulae. The *scope* of these quantifiers is  $A$ .

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )
- An  $n$ -ary predicate followed by  $n$  constants or variables is a formula. (eg.  $Lxb$ )
- If  $A$  and  $B$  are formulae, then so are  $A \wedge B, A \vee B, A \longrightarrow B, \neg A$ .
- If  $A$  is a formula and  $\tau$  is a variable then  $(\forall \tau)A$  and  $(\exists \tau)A$  are formulae. The *scope* of these quantifiers is  $A$ .

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )
- An  $n$ -ary predicate followed by  $n$  constants or variables is a formula. (eg.  $Lxb$ )
- If  $A$  and  $B$  are formulae, then so are  $A \wedge B, A \vee B, A \longrightarrow B, \neg A$ .
- If  $A$  is a formula and  $\tau$  is a variable then  $(\forall \tau)A$  and  $(\exists \tau)A$  are formulae. The *scope* of these quantifiers is  $A$ .

### Free and Bound Variables

In  $Lxb$ , the variable  $x$  is *free*, but in  $(\forall x)Lxb$  it is *bound*.

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )
- An  $n$ -ary predicate followed by  $n$  constants or variables is a formula. (eg.  $Lxb$ )
- If  $A$  and  $B$  are formulae, then so are  $A \wedge B, A \vee B, A \longrightarrow B, \neg A$ .
- If  $A$  is a formula and  $\tau$  is a variable then  $(\forall \tau)A$  and  $(\exists \tau)A$  are formulae. The *scope* of these quantifiers is  $A$ .

### Free and Bound Variables

In  $Lxb$ , the variable  $x$  is *free*, but in  $(\forall x)Lxb$  it is *bound*.

If  $x$  is free in  $A$ , then it is *bound by* the  $(\forall x)$  in  $(\forall x)A$ .

## First Order Logic: Syntax

- We start with variables  $x, y, z, \dots$ , constants  $a, b, c, \dots$  and predicates  $P, Q, R, \dots$ . Each predicate has an arity  $0 \leq n$ .
- (Predicates can include eg  $=, \leq, \dots$ )
- An  $n$ -ary predicate followed by  $n$  constants or variables is a formula. (eg.  $Lxb$ )
- If  $A$  and  $B$  are formulae, then so are  $A \wedge B, A \vee B, A \longrightarrow B, \neg A$ .
- If  $A$  is a formula and  $\tau$  is a variable then  $(\forall \tau)A$  and  $(\exists \tau)A$  are formulae. The *scope* of these quantifiers is  $A$ .

### Free and Bound Variables

In  $Lxb$ , the variable  $x$  is *free*, but in  $(\forall x)Lxb$  it is *bound*.

If  $x$  is free in  $A$ , then it is *bound by* the  $(\forall x)$  in  $(\forall x)A$ .

If a formula has no free variables, it is called a *closed* formula.

# First Order Logic: Semantics

- Recall that a statement is true or false *in a given situation*.

# First Order Logic: Semantics

- Recall that a statement is true or false *in a given situation*.
- We have formalised the statements, now we need to formalise the situations and the definition of truth.

# First Order Logic: Semantics

- Recall that a statement is true or false *in a given situation*.
- We have formalised the statements, now we need to formalise the situations and the definition of truth.
- It starts with a mathematical version of the translation key we saw earlier.

# First Order Logic: Semantics

- Recall that a statement is true or false *in a given situation*.
- We have formalised the statements, now we need to formalise the situations and the definition of truth.
- It starts with a mathematical version of the translation key we saw earlier.

# First Order Logic: Semantics

- Recall that a statement is true or false *in a given situation*.
- We have formalised the statements, now we need to formalise the situations and the definition of truth.
- It starts with a mathematical version of the translation key we saw earlier.

An *interpretation* is

- a set called the *domain*
- an element of the domain for each constant
- an  $n$ -ary relation over the domain for each  $n$ -ary predicate symbol

# First Order Logic: Semantics

- Recall that a statement is true or false *in a given situation*.
- We have formalised the statements, now we need to formalise the situations and the definition of truth.
- It starts with a mathematical version of the translation key we saw earlier.

An *interpretation* is

- a set called the *domain*
- an element of the domain for each constant
- an  $n$ -ary relation over the domain for each  $n$ -ary predicate symbol

A *valuation* gives us an element of the domain for each variable.

## First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ .

## First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ . ie valuations  $v$  and  $u$  are  $x$ -variants iff  $v(\tau) = u(\tau)$  for all variables other than  $x$ .

## First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ . ie valuations  $v$  and  $u$  are  $x$ -variants iff  $v(\tau) = u(\tau)$  for all variables other than  $x$ .

Given an interpretation  $i$  and valuation  $v$

- $Rxb$  is true iff  $(v(x), i(b)) \in i(R)$   
(and similarly for other atomic formulae)

## First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ . ie valuations  $v$  and  $u$  are  $x$ -variants iff  $v(\tau) = u(\tau)$  for all variables other than  $x$ .

Given an interpretation  $i$  and valuation  $v$

- $Rxb$  is true iff  $(v(x), i(b)) \in i(R)$   
(and similarly for other atomic formulae)
- $A \wedge B$  is true iff  $A$  is true and  $B$  is true  
(and similarly for the other truth-functional connectives)

## First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ . ie valuations  $v$  and  $u$  are  $x$ -variants iff  $v(\tau) = u(\tau)$  for all variables other than  $x$ .

Given an interpretation  $i$  and valuation  $v$

- $Rxb$  is true iff  $(v(x), i(b)) \in i(R)$   
(and similarly for other atomic formulae)
- $A \wedge B$  is true iff  $A$  is true and  $B$  is true  
(and similarly for the other truth-functional connectives)
- $(\forall x)(A)$  is true iff  $A$  is true for all  $x$ -variants of  $v$ .

# First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ . ie valuations  $v$  and  $u$  are  $x$ -variants iff  $v(\tau) = u(\tau)$  for all variables other than  $x$ .

Given an interpretation  $i$  and valuation  $v$

- $Rxb$  is true iff  $(v(x), i(b)) \in i(R)$   
(and similarly for other atomic formulae)
- $A \wedge B$  is true iff  $A$  is true and  $B$  is true  
(and similarly for the other truth-functional connectives)
- $(\forall x)(A)$  is true iff  $A$  is true for all  $x$ -variants of  $v$ .
- $(\exists y)(A)$  is true iff  $A$  is true for some  $y$ -variant of  $v$ .

# First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ . ie valuations  $v$  and  $u$  are  $x$ -variants iff  $v(\tau) = u(\tau)$  for all variables other than  $x$ .

Given an interpretation  $i$  and valuation  $v$

- $Rxb$  is true iff  $(v(x), i(b)) \in i(R)$   
(and similarly for other atomic formulae)
- $A \wedge B$  is true iff  $A$  is true and  $B$  is true  
(and similarly for the other truth-functional connectives)
- $(\forall x)(A)$  is true iff  $A$  is true for all  $x$ -variants of  $v$ .
- $(\exists y)(A)$  is true iff  $A$  is true for some  $y$ -variant of  $v$ .

# First Order Logic: Semantics

Two valuations are called  $x$ -variants, if they agree about all variables other than  $x$ . ie valuations  $v$  and  $u$  are  $x$ -variants iff  $v(\tau) = u(\tau)$  for all variables other than  $x$ .

Given an interpretation  $i$  and valuation  $v$

- $Rxb$  is true iff  $(v(x), i(b)) \in i(R)$   
(and similarly for other atomic formulae)
- $A \wedge B$  is true iff  $A$  is true and  $B$  is true  
(and similarly for the other truth-functional connectives)
- $(\forall x)(A)$  is true iff  $A$  is true for all  $x$ -variants of  $v$ .
- $(\exists y)(A)$  is true iff  $A$  is true for some  $y$ -variant of  $v$ .

Note that for a closed formula, the truth value is the same no matter what valuation we use.

## We will not study deduction, but...

- a deductive system gives rules which allow you to derive a formula from some other formulae
- a proof starts with some *premises*, applies rules and reaches a *conclusion*
- if the deductive system is *sound* then the resulting argument is always *valid*
- Its kind of magical isn't it? New information without even looking, testing infinitely many possibilities in a finite number of steps!
- deduction is really the whole point of logic, even though there are by-product benefits for other fields like databases, AI, ...

# What has all this got to do with databases?

## Parallels between logic and relational databases

- Quantifiers in SQL
- Relational calculus
- Databases as logical interpretations
- Simplifying a relational calculus example
- (Domain vs Tuple) relational calculus
- Ontologies and metadata

## Logical Existence

The logical quantifiers are also available in SQL.  
Here is an example, first formalised using logic.

## Logical Existence

The logical quantifiers are also available in SQL.  
Here is an example, first formalised using logic.

*Which of our employees have dependents?*

## Logical Existence

The logical quantifiers are also available in SQL.  
Here is an example, first formalised using logic.

*Which of our employees have dependents?*

Domain = people,

$Ex = (x \text{ is our employee}),$

$Dxy = (x \text{ is a dependent of } y).$

$$Ex \wedge (\exists y)Dyx$$

## Logical Existence

The logical quantifiers are also available in SQL.  
Here is an example, first formalised using logic.

*Which of our employees have dependents?*

Domain = people,

$Ex = (x \text{ is our employee}),$

$Dxy = (x \text{ is a dependent of } y).$

$$Ex \wedge (\exists y)Dyx$$

This formula is not closed, it has one free variable  $x$ .

A valuation makes this true iff it takes  $x$  to one of our employees who has dependents.

## Relational Calculus Existence

$$Ex \wedge (\exists y)Dyx$$

*Which of our employees have dependents?*

## Relational Calculus Existence

$$Ex \wedge (\exists y)Dyx$$

*Which of our employees have dependents?*

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.ssn)\}$

- note similarity to logic version, but

## Relational Calculus Existence

$$Ex \wedge (\exists y)Dyx$$

*Which of our employees have dependents?*

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.ssn)\}$

- note similarity to logic version, but
- the relationship "...dependent of ..." is encoded differently in logic and relational model

## Relational Calculus Existence

$$Ex \wedge (\exists y)Dyx$$

*Which of our employees have dependents?*

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.ssn)\}$

- note similarity to logic version, but
- the relationship "... dependent of ..." is encoded differently in logic and relational model
- logic represents individuals as elements of the domain

## Relational Calculus Existence

$$Ex \wedge (\exists y)Dyx$$

*Which of our employees have dependents?*

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.ssn)\}$

- note similarity to logic version, but
- the relationship "... dependent of ..." is encoded differently in logic and relational model
- logic represents individuals as elements of the domain
- the relationship represented directly by a predicate (which is interpreted by a relation)

## Relational Calculus Existence

$$Ex \wedge (\exists y)Dyx$$

*Which of our employees have dependents?*

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.ssn)\}$

- note similarity to logic version, but
- the relationship "... dependent of ..." is encoded differently in logic and relational model
- logic represents individuals as elements of the domain
- the relationship represented directly by a predicate (which is interpreted by a relation)
- relational model represents individuals as tuples

## Relational Calculus Existence

$$Ex \wedge (\exists y) Dyx$$

*Which of our employees have dependents?*

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.ssn)\}$

- note similarity to logic version, but
- the relationship “...dependent of ...” is encoded differently in logic and relational model
- logic represents individuals as elements of the domain
- the relationship represented directly by a predicate (which is interpreted by a relation)
- relational model represents individuals as tuples
- this many-to-one relationship represented by a foreign key

## SQL Existence

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.essn)\}$

*Which of our employees have dependents?*

## SQL Existence

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.essn)\}$

*Which of our employees have dependents?*

```
SELECT *  
FROM employee e  
WHERE EXISTS (SELECT *  
              FROM dependent d  
              WHERE e.ssn=d.essn)
```

## SQL Existence

$\{x.fname, x.lname \mid$   
 $employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.essn)\}$

*Which of our employees have dependents?*

```
SELECT *  
FROM employee e  
WHERE EXISTS (SELECT *  
              FROM dependent d  
              WHERE e.ssn=d.essn)
```

Adapted from [E&N §8.5.4] on EXISTS etc.

## Database as Interpretation

To evaluate the logic formula in a relational calculus query, we need a *valuation* for the free variables, and an *interpretation* for the vocabulary such as “*employee*”.

- The (logical) domain is *tuples*

## Database as Interpretation

To evaluate the logic formula in a relational calculus query, we need a *valuation* for the free variables, and an *interpretation* for the vocabulary such as “*employee*”.

- The (logical) domain is *tuples*
- Literal values like 42 and “Smith” become constant symbols (denoting 1-tuples)

## Database as Interpretation

To evaluate the logic formula in a relational calculus query, we need a *valuation* for the free variables, and an *interpretation* for the vocabulary such as “*employee*”.

- The (logical) domain is *tuples*
- Literal values like 42 and “Smith” become constant symbols (denoting 1-tuples)
- Each relation schema name in the database is a logical predicate: *employee*(*e*) is true iff *e* is a tuple in the *employee* relation.

## Database as Interpretation

To evaluate the logic formula in a relational calculus query, we need a *valuation* for the free variables, and an *interpretation* for the vocabulary such as “*employee*”.

- The (logical) domain is *tuples*
- Literal values like 42 and “Smith” become constant symbols (denoting 1-tuples)
- Each relation schema name in the database is a logical predicate: *employee*(*e*) is true iff *e* is a tuple in the *employee* relation.
- Each attribute name is a *function symbol*.

## Database as Interpretation

To evaluate the logic formula in a relational calculus query, we need a *valuation* for the free variables, and an *interpretation* for the vocabulary such as “*employee*”.

- The (logical) domain is *tuples*
- Literal values like 42 and “Smith” become constant symbols (denoting 1-tuples)
- Each relation schema name in the database is a logical predicate: *employee*(*e*) is true iff *e* is a tuple in the *employee* relation.
- Each attribute name is a *function symbol*.
- Relationships/Associations are more complicated than in direct logic formalisations, because translated via the relational model

# Attributes as Functions

- Recall that relational *algebra* also sees attributes are functions.

# Attributes as Functions

- Recall that relational *algebra* also sees attributes are functions.
- The projection function  $\pi_{Iname}$  takes an *employee* tuple and returns its *Iname* value.

# Attributes as Functions

- Recall that relational *algebra* also sees attributes are functions.
- The projection function  $\pi_{Iname}$  takes an *employee* tuple and returns its *Iname* value.
- In relational calculus, we write  $e.Iname$  for  $\pi_{Iname}(e)$ .

## Functions in Logic (defining them away)

The definition of logic syntax did not include function symbols, but we can use them without cheating.

## Functions in Logic (defining them away)

The definition of logic syntax did not include function symbols, but we can use them without cheating.

This is a good example of using logic to express ideas precisely.

## Functions in Logic (defining them away)

The definition of logic syntax did not include function symbols, but we can use them without cheating.

This is a good example of using logic to express ideas precisely.

- Recall that a function  $f : A \longrightarrow B$  is a relation  $f \subseteq A \times B$  with exactly one pair for each  $a \in A$

## Functions in Logic (defining them away)

The definition of logic syntax did not include function symbols, but we can use them without cheating.

This is a good example of using logic to express ideas precisely.

- Recall that a function  $f : A \longrightarrow B$  is a relation  $f \subseteq A \times B$  with exactly one pair for each  $a \in A$
- Suppose we want to use logic to talk about a function

## Functions in Logic (defining them away)

The definition of logic syntax did not include function symbols, but we can use them without cheating.

This is a good example of using logic to express ideas precisely.

- Recall that a function  $f : A \longrightarrow B$  is a relation  $f \subseteq A \times B$  with exactly one pair for each  $a \in A$
- Suppose we want to use logic to talk about a function
- We can use a binary predicate symbol  $F$  in place of a function symbol  $f$ , as follows

## Functions in Logic (defining them away)

Now,  $A$ ,  $B$ ,  $F$ ,  $x$  are *syntax*.

## Functions in Logic (defining them away)

Now,  $A$ ,  $B$ ,  $F$ ,  $x$  are *syntax*.

To get what they denote, apply interpretation  $i$  or valuation  $v$ .

## Functions in Logic (defining them away)

Now,  $A$ ,  $B$ ,  $F$ ,  $x$  are *syntax*.

To get what they denote, apply interpretation  $i$  or valuation  $v$ .

Then  $i(A)$ ,  $i(B)$  are sets,  $i(F)$  a relation,  $v(x)$  a domain element.

## Functions in Logic (defining them away)

Now,  $A, B, F, x$  are *syntax*.

To get what they denote, apply interpretation  $i$  or valuation  $v$ .

Then  $i(A), i(B)$  are sets,  $i(F)$  a relation,  $v(x)$  a domain element.

- We can ensure that  $i(F)$  contains at least one pair  $(a, b) \in i(A) \times i(B)$  for each member  $a \in i(A)$

$$(\forall x)(Ax \longrightarrow (\exists y)(By \wedge Fxy))$$

## Functions in Logic (defining them away)

Now,  $A, B, F, x$  are *syntax*.

To get what they denote, apply interpretation  $i$  or valuation  $v$ .

Then  $i(A), i(B)$  are sets,  $i(F)$  a relation,  $v(x)$  a domain element.

- We can ensure that  $i(F)$  contains at least one pair  $(a, b) \in i(A) \times i(B)$  for each member  $a \in i(A)$

$$(\forall x)(Ax \longrightarrow (\exists y)(By \wedge Fxy))$$

- We can ensure that there is no more than one (its unique)

$$(\forall x)(\forall y)(\forall z)((Fxy \wedge Fxz) \longrightarrow y = z)$$

## Functions in Logic (defining them away)

Now,  $A, B, F, x$  are *syntax*.

To get what they denote, apply interpretation  $i$  or valuation  $v$ .

Then  $i(A), i(B)$  are sets,  $i(F)$  a relation,  $v(x)$  a domain element.

- We can ensure that  $i(F)$  contains at least one pair  $(a, b) \in i(A) \times i(B)$  for each member  $a \in i(A)$

$$(\forall x)(Ax \longrightarrow (\exists y)(By \wedge Fxy))$$

- We can ensure that there is no more than one (its unique)

$$(\forall x)(\forall y)(\forall z)((Fxy \wedge Fxz) \longrightarrow y = z)$$

- Then, when we want to write  $\dots f(t) \dots$  we can write  $(\forall y)(Fty \longrightarrow (\dots y \dots))$  instead

What were those guys thinking of?

Write for human beings

## What were those guys thinking of?

Write for human beings

[E&N §6.6.7] Shows a tuple relational calculus query to show

*the names of employees who work on all the projects controlled  
by department number 5*

## What were those guys thinking of?

Write for human beings

[E&N §6.6.7] Shows a tuple relational calculus query to show

*the names of employees who work on all the projects controlled  
by department number 5*

$$\{e.lname, e.fname \mid employee(e) \wedge ((\forall x)(\neg project(x)) \vee \neg(x.dnum = 5) \vee ((\exists w)(worksOn(w) \wedge w.essn = e.ssn \wedge x.pnumber = w.pno))))\}$$

## What were those guys thinking of?

Write for human beings

[E&N §6.6.7] Shows a tuple relational calculus query to show

*the names of employees who work on all the projects controlled  
by department number 5*

$$\{e.lname, e.fname \mid employee(e) \wedge ((\forall x)(\neg project(x)) \vee \neg(x.dnum = 5) \vee ((\exists w)(worksOn(w) \wedge w.essn = e.ssn \wedge x.pnumber = w.pno))))\}$$

Which is a lot more complicated than it needs to be.

## *If ... then ...* in (classical) Logic

- Greg: “Hey, I’m going to the shop (its a long way), if you give me \$2, I’ll buy you a sausage roll.”

## *If ... then ...* in (classical) Logic

- Greg: “Hey, I’m going to the shop (its a long way), if you give me \$2, I’ll buy you a sausage roll.”
- Raj: “No thanks Greg, I’m a vegetarian”

## *If ... then ...* in (classical) Logic

- Greg: “Hey, I’m going to the shop (its a long way), if you give me \$2, I’ll buy you a sausage roll.”
- Raj: “No thanks Greg, I’m a vegetarian”
- Did Greg tell the truth?

## *If ... then ...* in (classical) Logic

- Greg: “Hey, I’m going to the shop (its a long way), if you give me \$2, I’ll buy you a sausage roll.”
- Raj: “No thanks Greg, I’m a vegetarian”
- Did Greg tell the truth?
- If  $A$  is false and  $B$  is false, is  $A \longrightarrow B$  true, or false?

## *If ... then ...* in (classical) Logic

- Greg: “Hey, I’m going to the shop (its a long way), if you give me \$2, I’ll buy you a sausage roll.”
- Raj: “No thanks Greg, I’m a vegetarian”
- Did Greg tell the truth?
- If  $A$  is false and  $B$  is false, is  $A \longrightarrow B$  true, or false?
- In classical logic,  $A \longrightarrow B$  is equivalent to  $\neg A \vee B$

## If ... then ... in (classical) Logic

- Greg: “Hey, I’m going to the shop (its a long way), if you give me \$2, I’ll buy you a sausage roll.”
- Raj: “No thanks Greg, I’m a vegetarian”
- Did Greg tell the truth?
- If  $A$  is false and  $B$  is false, is  $A \longrightarrow B$  true, or false?
- In classical logic,  $A \longrightarrow B$  is equivalent to  $\neg A \vee B$
- Also note that  $A \longrightarrow (B \longrightarrow C)$  is equivalent to  $(A \wedge B) \longrightarrow C$   
(They are only false when  $A$  and  $B$  are true and  $C$  is false)

So, what they mean is...

So,  $\neg A \vee \neg B \vee C$  is equivalent to  $(A \wedge B) \longrightarrow C$

## So, what they mean is...

So,  $\neg A \vee \neg B \vee C$  is equivalent to  $(A \wedge B) \longrightarrow C$

We can therefore rewrite

$$\begin{aligned} & (\forall x)(\neg \text{project}(x) \vee \neg(x.\text{dnum} = 5) \vee \\ & ((\exists w)(\text{worksOn}(w) \wedge w.\text{essn} = e.\text{ssn} \wedge x.\text{pnumber} = w.\text{pno}))) \end{aligned}$$

as

$$\begin{aligned} & (\forall x)(\text{project}(x) \wedge x.\text{dnum} = 5 \longrightarrow \\ & ((\exists w)(\text{worksOn}(w) \wedge w.\text{essn} = e.\text{ssn} \wedge x.\text{pnumber} = w.\text{pno}))) \end{aligned}$$

## So, what they mean is...

So,  $\neg A \vee \neg B \vee C$  is equivalent to  $(A \wedge B) \longrightarrow C$

We can therefore rewrite

$$\begin{aligned} & (\forall x)(\neg \text{project}(x) \vee \neg(x.\text{dnum} = 5) \vee \\ & ((\exists w)(\text{worksOn}(w) \wedge w.\text{essn} = e.\text{ssn} \wedge x.\text{pnumber} = w.\text{pno}))) \end{aligned}$$

as

$$\begin{aligned} & (\forall x)(\text{project}(x) \wedge x.\text{dnum} = 5 \longrightarrow \\ & ((\exists w)(\text{worksOn}(w) \wedge w.\text{essn} = e.\text{ssn} \wedge x.\text{pnumber} = w.\text{pno}))) \end{aligned}$$

The second line is just saying there is a link-tuple between the project and the employee.

# Domain Relational Calculus

Is exactly the same as tuple relational calculus, except

- The (logical) domain is not tuples, but the union of all the database domains.

# Domain Relational Calculus

Is exactly the same as tuple relational calculus, except

- The (logical) domain is not tuples, but the union of all the database domains.
- So, the variables range over atomic values, not tuples.

# Domain Relational Calculus

Is exactly the same as tuple relational calculus, except

- The (logical) domain is not tuples, but the union of all the database domains.
- So, the variables range over atomic values, not tuples.
- You have to make up tuples from these variables.

# Domain Relational Calculus

Is exactly the same as tuple relational calculus, except

- The (logical) domain is not tuples, but the union of all the database domains.
- So, the variables range over atomic values, not tuples.
- You have to make up tuples from these variables.
- To make it a bit easier, we write  $xyz$  for  $(x, y, z)$ .

# Domain Relational Calculus Example

## tuple relational calculus

$$\{x.fname, x.lname \mid$$
$$employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.essn)\}$$

## domain relational calculus

$$\{f \ell \mid employee(fx_2 \ell sx_5 x_6 x_7 x_8 x_9 x_{10}) \wedge$$
$$(\exists d)(\exists y_2)(\exists y_3)(\exists y_4)(dependent(dy_2 y_3 y_4) \wedge s = d)\}$$

# Domain Relational Calculus Example

## tuple relational calculus

$$\{x.fname, x.lname \mid employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.essn)\}$$

## domain relational calculus

$$\{f \ell \mid employee(fx_2 \ell sx_5 x_6 x_7 x_8 x_9 x_{10}) \wedge (\exists d)(\exists y_2)(\exists y_3)(\exists y_4)(dependent(dy_2 y_3 y_4) \wedge s = d)\}$$

- domain .. would be more concise than tuple .. for complex condition expressions

# Domain Relational Calculus Example

## tuple relational calculus

$$\{x.fname, x.lname \mid employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.essn)\}$$

## domain relational calculus

$$\{f \ell \mid employee(fx_2 \ell sx_5 x_6 x_7 x_8 x_9 x_{10}) \wedge (\exists d)(\exists y_2)(\exists y_3)(\exists y_4)(dependent(dy_2 y_3 y_4) \wedge s = d)\}$$

- domain .. would be more concise than tuple .. for complex condition expressions
- but, more brittle: add attribute to relation and queries break

# Domain Relational Calculus Example

## tuple relational calculus

$$\{x.fname, x.lname \mid employee(x) \wedge (\exists y)(dependent(y) \wedge x.ssn = y.essn)\}$$

## domain relational calculus

$$\{f \ell \mid employee(fx_2 \ell sx_5 x_6 x_7 x_8 x_9 x_{10}) \wedge (\exists d)(\exists y_2)(\exists y_3)(\exists y_4)(dependent(dy_2 y_3 y_4) \wedge s = d)\}$$

- domain .. would be more concise than tuple .. for complex condition expressions
- but, more brittle: add attribute to relation and queries break
- Do you think it will catch on?

# Metadata and Ontologies

- DBMS has tables *about* its tables

# Metadata and Ontologies

- DBMS has tables *about* its tables
- eg. *table(tableName)*, *attribute(attribName, type, table)*,  
where *attribute(table)* refers to *table(tableName)*

# Metadata and Ontologies

- DBMS has tables *about* its tables
- eg. *table(tableName)*, *attribute(attribName, type, table)*, where *attribute(table)* refers to *table(tableName)*
- this can be extended with human-readable description/definition text, called a *data dictionary*

# Metadata and Ontologies

- DBMS has tables *about* its tables
- eg. *table(tableName)*, *attribute(attribName, type, table)*, where *attribute(table)* refers to *table(tableName)*
- this can be extended with human-readable description/definition text, called a *data dictionary*
- these definitions can also be formalised using logic, allowing “smarter” queries. eg. patients with finger amputated includes patients with arm amputated. This kind of machine-readable knowledge dictionary is sometimes called an “ontology”

# The Logic Database Nexus

To summarise

## Logic and Databases

- both based on relations
- logic formulae to express conditions for queries
- metadata  $\longrightarrow$  ontologies