I Relational Database Modeling– how to define

Relational Model

- data structure, operations, constraints
- Design theory for relational database

High-level Models

- E/R model, UML model, ODL

II Relational Database Programming – how to operate

From an abstract point of view to study the question of database queries and modifications. (chapter 5)

- Relational Algebra
- A Logic for relation

From a practical point to learn the operations on Database

- The Database Language SQL (chapter 6~10)

Chapter 5 Algebraic and Logic Query languages

- Relational operations (chapter 2)
- Extended operators
- Datalog: a logic for relations
- Relational algebra and Datalog

Review 1: what is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
  - The result is an algebra that can be used as a *query language* for relations.

Review 2:
"Core" of Relational Algebra

- **Set operations: Union, intersection and difference (the relation schemas must be the same)**
- **Selection: Picking certain rows from a relation.**
- **Projection: picking certain columns.**
- **Products and joins: composing relations in a useful ways.**
- **Renaming of relations and their attributes.**

Review 3: Bags Model

- SQL, the most important query language for relational databases is actually a bag language.
  - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like projection, are much more efficient on bags than sets.

Extended ("Nonclassical") Relational Algebra

Add features needed for SQL bags.

1. Duplicate-elimination operator δ
2. Extended projection.
3. Sorting operator τ
4. Grouping-and-aggregation operator γ
5. Outerjoin operator ∞°

Duplicate Elimination

- δ (R) = relation with one copy of each tuple that appears one or more times in R.

Example R =

A	B
1	2
3	4
1	2

δ (R) =

A	B
1	2
3	4

Sorting

$\tau_L(R)$ = list of tuples of R, ordered according to attributes on list L

Note that result type is outside the normal types (set or bag) for relational algebra. Consequence, τ cannot be followed by other relational operators.

R =

A	B
1	2
3	4
5	2

$\tau_B(R)$ = [(1,2), (5,2), (3,4)]

Extended Projection

Allow the columns in the projection to be functions of one or more columns in the argument relation.

Example:

R =

A	B
1	2
3	4

$\pi_{A+B,A,A}(R)$ =

A+B	A1	A2
3	1	1
7	3	3

- Arithmetic on attributes
- Duplicate occurrences of the same attribute

Aggregation Operators

- Aggregation operators are not operators of relational algebra.
- Rather, they apply to entire columns of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.

Example: Aggregation

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

Grouping Operator

$\Upsilon_L(R)$ where L is a list of elements that are either

1. Individual (grouping) attributes or
2. Of the form $\theta(A)$, where θ is an aggregation operator and A the attribute to which it is applied, computed by:

- Grouping R according to all the grouping attributes on list L.
- Within each group, compute $\theta(A)$, for each element $\theta(A)$ on list L
- Result is the relation whose column consist of one tuple for each group. The components of that tuple are the values associated with each element of L for that group.

Example: compute $\gamma_{beer,\ AVG(price)}(R)$

R=

Bar	Beer	price
Joe's	Bud	2.00
Joe's	Miller	2.75
Sue's	Bud	2.5
Sue's	Coors	3.00
Mel's	Miller	3.25

1. Group by the grouping attributes, beer in this case:

Bar	Beer	price
Joe's	Bud	2.00
Sue's	Bud	2.5
Joe's	Miller	2.75
Mel's	Miller	3.25
Sue's	Coors	3.00

Example (cont.)

2. Computer average of price with groups:

Beer	AVG (price)
Bud	2.25
Miller	3.00
Coors	3.00

$\gamma_{beer,\ AVG(price)}(R)$

Example: Grouping/Aggregation

R =

A	B	C
1	2	3
4	5	6
1	2	5

$\gamma_{A,B,AVG(C)}$ (R) = ??

First, group *R* :

A	B	C
1	2	3
1	2	5
4	5	6

Then, average *C* within groups:

A	B	AVG(C)
1	2	4
4	5	6

Outjoin

The normal join can “lose”information, because a tuple that doesn’t join with any from the other relation(dangles) has no vestige in the join result.

- The null value can be used to “pad”dangling tuples so they appear in the join.
- Gives us the outerjoin operator $\circ\!\bowtie$
- Variations: theta-outjoin, left- and right-outjoin (pad only dangling tuples from the left (resp., right).

Example: Outerjoin

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples are dangling.

R OUTERJOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

Example (cont.)

$R \overset{\circ}{\bowtie}_L S =$

A	B	C
1	2	3
4	5	NULL

$R \overset{\circ}{\bowtie}_R S =$

A	B	C
1	2	3
null	6	7

Classroom Exercises

- R(A,B): {(0,1),(2,3),(0,1),(2,4),(3,4)}
- S(B,C):{(0,1),(2,4),(2,5),(3,4),(0,2),(3,4)

Computer:

1) $\pi_{B+1,C-1}$ (S) 2) $\tau_{b,a}$ (R) 3)δ (R)

4) $\Upsilon_{a,\ sum(b)}$ (R) 5) R outjoint S

Logic As a Query Language

- If-then logical rules have been used in many systems.
  - Important example: EII (Enterprise Information Integration).
- Nonrecursive rules are equivalent to the core relational algebra.
- Recursive rules extend relational algebra and appear in SQL-99.

Example: Enterprise Integration

- Goal: integrated view of the menus at many bars Sells(bar, beer, price).
- Joe has data JoeMenu(beer, price).
- Approach 1: Describe Sells in terms of JoeMenu and other local data sources.

Sells('Joe"s Bar', b, p) <- JoeMenu(b, p)

EII – (2)

- Approach 2: Describe how JoeMenu can be used as a view to help answer queries about Sells and other relations.

JoeMenu(b, p) <- Sells('Joe"s Bar', b, p)

- More about information integration later.

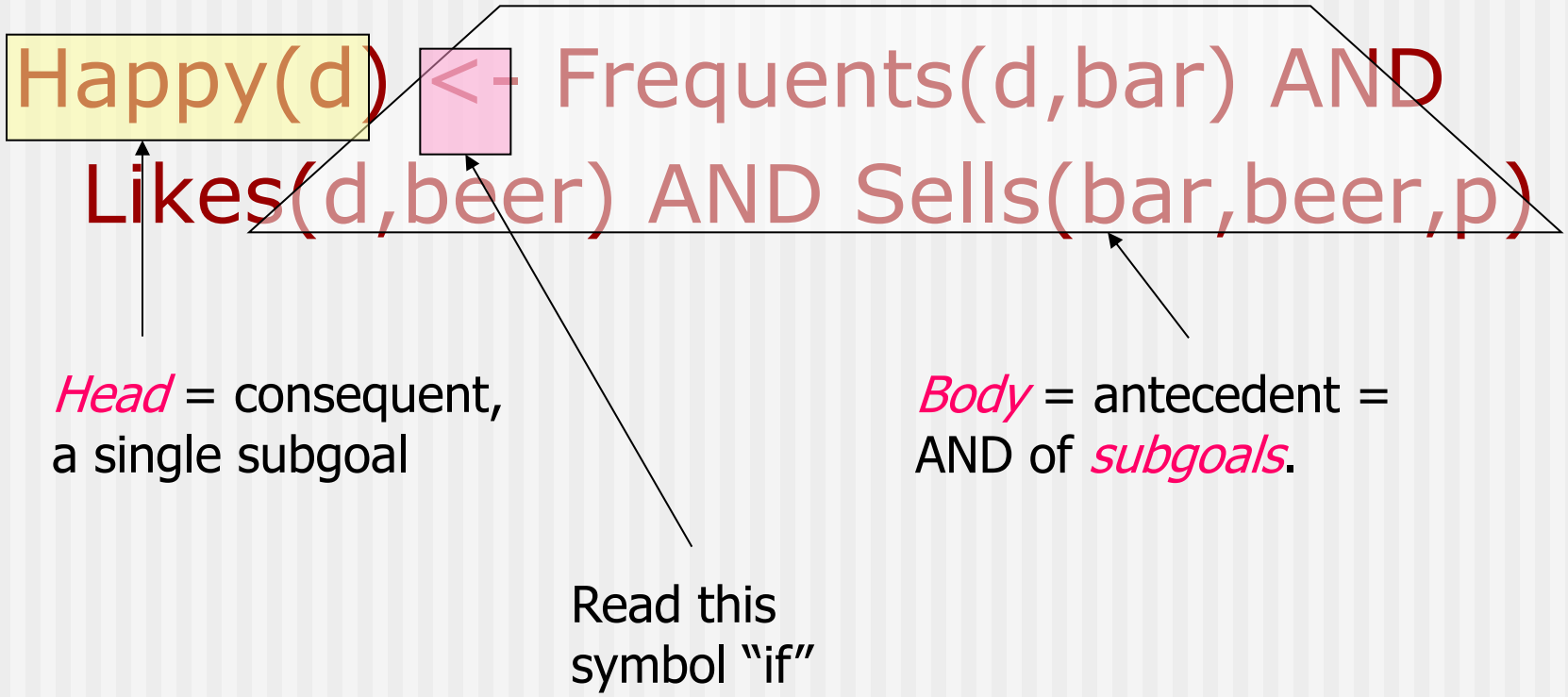
Predicates and atoms

- Relations are represented in Datalog by predicates.
- A predicate followed by its arguments is called an atom.
  - Atom = *predicate* and arguments.
  - Predicate = relation name or arithmetic predicate, *e.g.* <.
  - Arguments are variables or constants.
- R(a1,a2,…an) has value TRUE if (a1,a2,…an) is a tuple of R, otherwise, it is false.

A Logical Rule

- Our first example of a rule uses the relations Frequents(drinker, bar), Likes(drinker, beer), and Sells(bar, beer, price).
- The rule is a query asking for "happy" drinkers --- those that frequent a bar that serves a beer that they like.

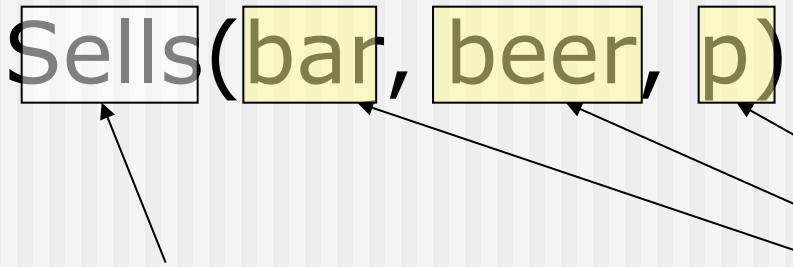
Anatomy of a Rule



Subgoals Are Atoms

- An *atom* is a *predicate*, or relation name with variables or constants as arguments.
- The head is an atom; the body is the AND of one or more atoms.
- Convention: Predicates begin with a capital, variables begin with lower-case.

Example: Atom



The predicate = name of a relation

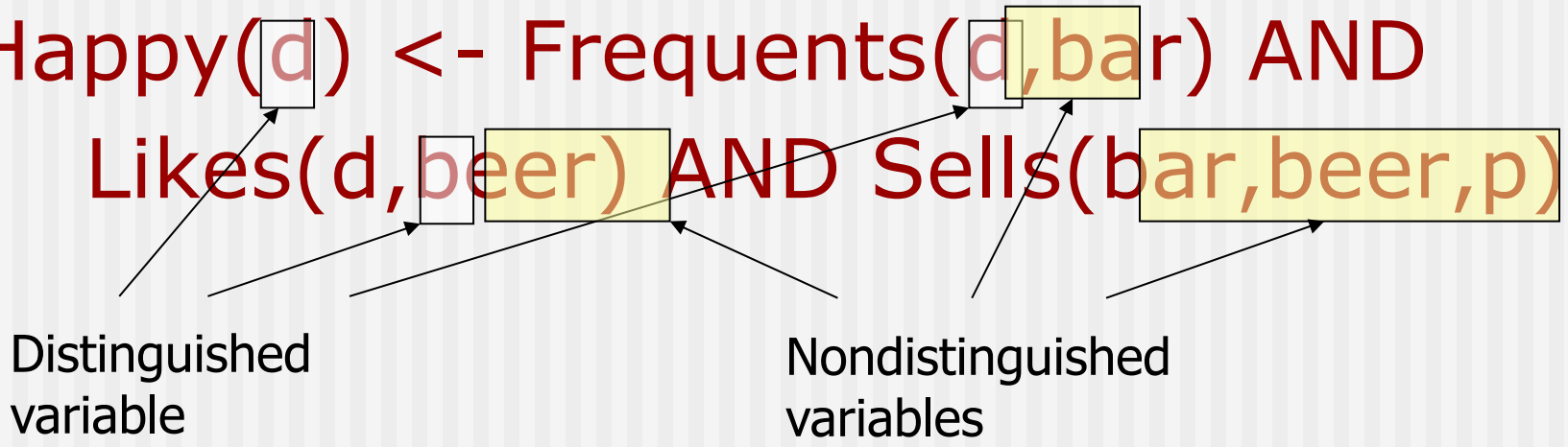
Arguments are variables (or constants).

Interpreting Rules

- A variable appearing in the head is *distinguished* ; otherwise it is *nondistinguished*.
- Rule meaning: The head is true for given values of the distinguished variables if there exist values of the nondistinguished variables that make all subgoals of the body true.

Example: Interpretation

Happy(d) <- Frequents(d,bar) AND
Likes(d,beer) AND Sells(bar,beer,p)



Interpretation: drinker *d* is happy if there exist a bar, a beer, and a price *p* such that *d* frequents the bar, likes the beer, and the bar sells the beer at price *p*.

Applying a Rule

- Approach 1: consider all combinations of values of the variables.
- If all subgoals are true, then evaluate the head.
- The resulting head is a tuple in the result.

Example: Rule Evaluation

Happy(d) <- Frequents(d,bar) AND
Likes(d,beer) AND
Sells(bar,beer,p)

FOR (each d, bar, beer, p)

IF (Frequents(d,bar), Likes(d,beer), and Sells(bar,beer,p) are all true)

add Happy(d) to the result

- Note: set semantics so add only once.

Drinker	Bar
David	Joe'sbar
Frank	Sue's bar
Susan	Joe's bar

Drinker	Beer
David	Bud
David	Miller
Frank	Bud
Susan	Coors

Bar	Beer	price
Joe's	Bud	2.00
Joe's	Miller	2.75
Sue's	Bud	2.5
Sue's	Coors	3.00

Only assignments that make all subgoals true:

$d \rightarrow$ David, bar $\rightarrow$ Joe'sbar, Beer→Bud

$d \rightarrow$ David, bar $\rightarrow$ Joe'sbar, Beer→Miller

$d \rightarrow$ Frank, bar $\rightarrow$ Sue'sbar, Beer→Bud

In the above cases it makes subgoals all true. Thus, add (*d*) = (david, Frank) to happy (d).

d→ Susan, bar→ Joe'sbar, beer→Coors, however the third subgoal is not true, because (Joe'sbar, Coors,p) is not in Sells.

A Glitch (Fixed Later)

- Relations are finite sets.
- We want rule evaluations to be finite and lead to finite results.
- "Unsafe" rules like P(x)<-Q(y) have infinite results, even if *Q* is finite.
- Even P(x)<-Q(x) requires examining an infinity of *x*-values.

Applying a Rule

- Approach 2: For each subgoal, consider all tuples that make the subgoal true.
- If a selection of tuples define a single value for each variable, then add the head to the result.
- Leads to finite search for P(x)<-Q(x), but P(x)<-Q(y) is problematic.

Example: Rule Evaluation – (2)

Happy(d) <- Frequents(d,bar) AND
Likes(d,beer) AND Sells(bar,beer,p)

FOR (each f in Frequents, i in Likes, and s in Sells)

IF (f[1]=i[1] and f[2]=s[1] and i[2]=s[2])

add Happy(f[1]) to the result

Drinker	Bar
David	Joe'sbar
Frank	Sue's bar
Susan	Joe's bar

Drinker	Beer
David	Bud
David	Miller
Frank	Bud
Susan	Coors

Bar	Beer	price
Joe's	Bud	2.00
Joe's	Miller	2.75
Sue's	Bud	2.5
Sue's	Coors	3.00

Three assignments of tuples to subgoals:

f(david Joe'sbar) i(David Bud) s(Joe's Bud 2.00)
f(david Joe'sbar) i(David Miller) s(Joe's Miller 2.75)
f(frank,Sue'sbar) i(Frank Bud) s(Sue's Bud 2.5)

makes
f[1]=i[1] and f[2]=s[1] and i[2]=s[2]) true

Thus, (david,frank) is the only tuples for the head.

Arithmetic Subgoals

- In addition to relations as predicates, a predicate for a subgoal of the body can be an arithmetic comparison.
- We write arithmetic subgoals in the usual way, e.g., $x < y$.

Example: Arithmetic

- A beer is "cheap" if there are at least two bars that sell it for under $2.

```
Cheap(beer) <- Sells(bar1,beer,p1)
    AND
        Sells(bar2,beer,p2) AND p1 < 2.00
        AND p2 < 2.00 AND bar1 <> bar2
```

Negated Subgoals

- NOT in front of a subgoal negates its meaning.
- Example: Think of Arc(a,b) as arcs in a graph.
  - S(x,y) says the graph is not transitive from *x* to *y* ; i.e., there is a path of length 2 from *x* to *y*, but no arc from *x* to *y*.

S(x,y) <- Arc(x,z) AND Arc(z,y)
AND NOT Arc(x,y)

Safe Rules

- A rule is *safe* if:
  1. Each distinguished variable,
  2. Each variable in an arithmetic subgoal, and
  3. Each variable in a negated subgoal,

  also appears in a nonnegated, relational subgoal.
- Safe rules prevent infinite results.

Example: Unsafe Rules

- Each of the following is unsafe and not allowed:
  1. S(x) <- R(y)
  2. S(x) <- R(y) AND NOT R(x)
  3. S(x) <- R(y) AND x < y
- In each case, an infinity of *x* 's can satisfy the rule, even if *R* is a finite relation.

An Advantage of Safe Rules

- We can use "approach 2" to evaluation, where we select tuples from only the nonnegated, relational subgoals.
- The head, negated relational subgoals, and arithmetic subgoals thus have all their variables defined and can be evaluated.

Datalog Programs

- *Datalog program* = collection of rules.
- In a program, predicates can be either
  1. EDB = *Extensional Database* = stored table.
  2. IDB = *Intensional Database* = relation defined by rules.
- Never both! No EDB in heads.

Evaluating Datalog Programs

- As long as there is no recursion, we can pick an order to evaluate the IDB predicates, so that all the predicates in the body of its rules have already been evaluated.
- If an IDB predicate has more than one rule, each rule contributes tuples to its relation.

Example: Datalog Program

- Using EDB Sells(bar, beer, price) and Beers(name, manf), find the manufacturers of beers Joe doesn't sell.

JoeSells(b) <- Sells('Joe"s Bar', b, p)

Answer(m) <- Beers(b,m)

AND NOT JoeSells(b)

Example: Evaluation

- Step 1: Examine all Sells tuples with first component 'Joe"s Bar'.
  - Add the second component to JoeSells.
- Step 2: Examine all Beers tuples (b,m).
  - If *b* is not in JoeSells, add *m* to Answer.

Relational Algebra & Datalog

- Both are query languages for relational database (abstractly)
- Algebra: use algebra expression.
- Datalog: use logic expressions.

Core of algebra = Datalog rules (no recursive)

From Relational Algebra to Datalog

$R \cap S$	$I(x) \leftarrow R(x)$ AND $S(x)$
$R \cup S$	$I(x) \leftarrow R(x)$
	$I(x) \leftarrow S(x)$
$R - S$	$I(x) \leftarrow R(x)$ AND NOT $S(x)$
$\pi_A(R)$	$I(a) \leftarrow R(a,b)$
$\sigma_F(R)$	$I(x) \leftarrow R(x)$ AND F

From Relational Algebra to Datalog (cont.)

$\pi_A(R)$	$I(a) \leftarrow R(a,b)$
$\sigma_{C1 \text{ AND } C2}(R)$	$I(x) \leftarrow R(x)$ AND C1 AND C2
$\sigma_{C1 \text{ OR } C2}(R)$	$I(x) \leftarrow R(x)$ AND C1 $I(x) \leftarrow R(x)$ AND C2
$R \times S$	$I(x,y) \leftarrow R(x)$ AND $S(y)$
$R \bowtie S$	$I(x,y,z) \leftarrow R(x,y)$ AND $S(y,z)$

Examples

Movie(title,year,length,inColor,studioname, producerC#)

- Projection $\pi_{title,year,length}(Movie)$

P(t,y,l)←Movie(t,y,l,c,s,p)

- Selection

$\sigma_{length>=100\ OR\ studioname='fox'}(Movie)$

S(t,y,l,c,s,p)←Movie(t,y,l,c,s,p) AND l>=100

S(t,y,l,c,s,p)←Movie(t,y,l,c,s,p) AND s='fox'

- Natural Join

J(a,b,c,d)←R(a,b) AND S(b,c,d)

Expressive Power of Datalog

- Without recursion, Datalog can express all and only the queries of core relational algebra.
  - The same as SQL select-from-where, without aggregation and grouping.
- But with recursion, Datalog can express more than these languages.

Recursive Rule example

Path(X,Y) ← Edge (X,Y)

Path (X,Y) ← Edge (X,Z) AND Path(Z,Y)

Summary of Chapter 5

- Extensions to relational algebra
- Datalog: This form of logic allows us to write queries in the relational model.
- Rule: head ← subgoals, they are atoms, and an atom consists of an predicate applied to some number of arguments.
- IDB and EDB
- Relational algebra and datalog

HomeWork & Classroom Exercises

- Exercise 5.3.1 (2.4.1) a), f), h)
- Exercise 5.4.1 g)

Let R(a,b,c), S(a,b,c) and T(a,b,c) be three relations, write one or more Datalog rules that define the result of the following algebra.

$\Pi_{a,b}(R) \cap \rho_{U(a,b)} (\Pi_{b,c}(S))$