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

- Chapter 5： *From an abstract point of view to study the question of database queries and modifications.*
  - Relational Algebra
  - A Logic for Relation

- Chapter 6~10： *From a practical point to learn the operations on Database*
  - The Database Language SQL

Chapter 5 Algebraic and Logic Query languages

- Relational operations (chapter 2)
- Extended operators
- Datalog: a logic for relations
- Relational algebra vs. 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 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 $\gamma_{\text{beer, AVG(price)}}(R\ )$

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

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

Example: Grouping/Aggregation

R =

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

$\gamma_{A,B,\mathrm{AVG}(C)}$ (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

A	B
1	2
4	5

B	C
2	3
4	6

=

A	B	C
1	2	3

- The normal join can "lose" information, the (4,5) and (4,6) (dangles) has no vestige in the join result.
- Outerjoin operator $\circ\!\bowtie$: the null value can be used to "pad" dangling tuples.
- 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.)

$R \;⟕_{L}\; S =$

A	B	C
1	2	3
4	5	NULL

$R \;⟖\; {}_{R}S =$

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 $_{outjoin}$ S

Logic As a Query Language

- If-then logical rules have been used in many systems.
- Nonrecursive rules are equivalent to the core relational algebra.
- Recursive rules extend relational algebra and appear in SQL-99.

Logic As a Query Language (cont.)

- A Query: to find a cheap beer whose price is less than 2 dollars

- A Rule:

if sells (bar,beer,price) and the price < 2 then the beer is cheap.

Predicates and atoms

- 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.
- Relations are represented in Datalog by predicates.
- R(a1,a2,…an) has value TRUE if (a1,a2,…an) is a tuple of R, otherwise, it is false.

A Logical Rule

- Frequents(drinker, bar)
  Likes(drinker, beer)

  Sells(bar, beer, price)
- Define a rule called "happy drinkers" --- those that *frequent* a bar that *serves* a beer that they *like*.

Anatomy of a Rule

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

Head = consequent,
a single subgoal

Read this
symbol "if"

Body = antecedent =
AND of *subgoals*.

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

Sells(bar, beer, p)

The predicate
= name of a
relation

Arguments are
variables (or constants).

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's bar
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 → David, bar → Joe'sbar, Beer→Bud

d → David, bar → Joe'sbar, Beer→Miller

d → Frank, bar → 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.

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.

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's bar
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 variable in head,
  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

R
7
9
0
1

S
?
?
?
?

- In each case, an **infinity** of *x* ’s can satisfy the rule, even if *R* is a finite relation.

An Advantage of Safe Rules

R
7
9
0
1

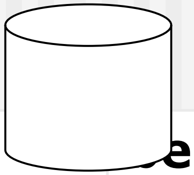
- Safe rule: S(x) <- R(x) AND x > 1
- Where tuples(x) is from only the nonnegated, relational subgoals R.
- The head, negated relational subgoals, and arithmetic subgoals thus have all their variables defined and can be evaluated.

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.

For example

Create table sells(bar string, beer string, Price float);



bar	beer	price
Joe’s	Bud	3
Joes’s	Miller	1
Mary’s	Bud	1
Mary’s	Miller	1.5
David	Bud	1.5

- EDB:

Sells(bar,beer,price)

Beer(name,manf)

- IDB:

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

cheapBeer
Miller
Bud

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

Evaluating Datalog Programs

- Pick an order to evaluate the IDB predicates, all the predicates in the body of its rules needs to be evaluated.
- If an IDB predicate has more than one rule, each rule contributes tuples to its relation.

Example: Datalog Program

- EDB Sells(bar, beer, price) and Beers(name, manf)
- Query: to 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 $S(y,z)$

Example:

- U (a,b,c) and V (b,c,d) have theta join

Relational algebra:

$$U \underset{a<d \text{ or } U.b<>V.b}{\bowtie} V$$

Relational datalog:

X(a,ub,uc,vb,vc,d)<- U(a,ub,uc) and V(vb,vc,d) and a<d

X(a,ub,uc,vb,vc,d) <- U(a,ub,uc) and V(vb,vc,d) and ub <>vb

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)

More will be on chapter 6

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 vs. datalog

HomeWork

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

- Upload your homework until next Thursday