Answering Queries Using Views: Logic-Based Approach

CPS 296.1
Topics in Database Systems

Logic-based approach
• Often used in data integration
• Focuses on finding as many answers as possible
• Bucket-based algorithms
  – Levy et al. “Querying Heterogeneous Information Sources Using Source Descriptions.” VLDB, 1996
• Inverse-rules algorithm

Brute-force algorithm
• Q: a CQ to be answered
• V₁, V₂, …: views (also defined as CQ’s)
• To find a rewriting of Q using Vᵢ’s
  – Try each possible join of Vᵢ’s as a rewriting for Q
  – Expand all Vᵢ’s in the join (that is, replace each Vᵢ by its definition)
  – Test if the expansion (also a CQ) is contained in Q
  – Q is rewritten as a union of these rewritings
➢ Too many possibilities to explore

Expanding a rewriting
• Example
  – View: grandparent(X, Z) :- parent(X, Y), parent(Y, Z)
  – Rewriting of a query: g-g-g-grandparent(X, Z) :
    grandparent(X, Y), grandparent(Y, Z)
  – Expansion: g-g-g-grandparent(X, Z) :-
    parent(X, Y₁), parent(Y₁, Y),
    parent(Y, Y₂), parent(Y₂, Z)
• Watch the use of variables
  – Use the query variables for the head of the views
  – Make sure variables “local” to different views do not clash with each other

Bucket algorithm
• Remember there should be a containing mapping from Q to a rewriting (with views expanded)
  – Each subgoal of Q must be covered by some view in the rewriting; that is, the query subgoal must map to some subgoal in some view
  – A distinguished variable (one that appears in the head of a rule) in Q must map to a distinguished variable in some view
  – For a shared variable X (one that appears more than once in the body of a rule; i.e., needed for join) in Q, either
    • X maps to a distinguished variable in some view, or
    • All query subgoals involving X map to subgoals of a single view

Examples (slide 1)

Rewriting … f(X, Y) … V(..., Y, ...)
Expansion p(X, W), q(W, Z), r(Z, Y) … Y …
Query  Q(D, E) :- … r(x, B) … s(x, C)

• A is not distinguished, and not shared
  – A can map to Z in the expansion of V (not distinguished)
• B is not distinguished, but shared
  – Given the A mapping, B should map to Y in V (distinguished)
  – Other occurrences of B can map to distinguished variables in some other view (say Y in V)
Examples (slide 2)

Rewriting … \( f(X, Y) \) …
Expansion \( p(X, W), q(W, Z), r(Z, Y) \) … ?
Query \( Q(D, E) \) :- \( q(A, B) \) … \( r(B, C) \)

- A is not distinguished, and not shared
  - A can map to W in the expansion of V (not distinguished)
- B is not distinguished, but shared
  - Given the A mapping, B is forced to Z (not distinguished)
  - The other occurrence of B now has no place to go!
    - V has no s subgoal
    - Another view expansion would not have Z as a variable

Examples (slide 3)

Rewriting … \( f(X, Y) \) …
Expansion \( p(X, W), q(W, Z), r(Z, Y) \) …
Query \( Q(D, E) \) :- \( q(A, B) \) … \( r(B, C) \)

- A is not distinguished, and not shared
  - A can map to W in the expansion of V (not distinguished)
- B is not distinguished, but shared
  - Given the A mapping, B is forced to Z (not distinguished)
  - This mapping also happens to work out for the other occurrence of B
  - So B is completely “covered” by V

Examples (slide 4)

Rewriting \( Q(X, U) \) :- \( f(X, Y) \) …
Expansion \( q(X, W), q(W, Z), r(Z, Y) \) …
Query \( Q(A, D) \) :- \( p(A, B) \) … \( q(B, C) \)

- A is distinguished
  - Then A must map to a distinguished variable in a view expansion
  - Otherwise the target variable cannot appear in the head of the rewriting

Examples (slide 3)

Rewriting … \( f(X, Y) \) …
Expansion \( p(X, W), q(W, Z), r(Z, Y) \) …
Query \( Q(D, E) \) :- \( q(A, B) \) … \( r(B, C) \)

- A is not distinguished, and not shared
  - A can map to W in the expansion of V (not distinguished)
- B is not distinguished, but shared
  - Given the A mapping, B is forced to Z (not distinguished)
  - This mapping also happens to work out for the other occurrence of B
  - So B is completely “covered” by V

Example of filling buckets (slide 1)

- Views
  - grandparent(X, Y) :- parent(X, Z), parent(Z, Y)
  - great-grandparent(U, V) :- parent(U, S), parent(S, T), parent(T, V)
- Query
  - query(A, B) :-
    parent(A, C), parent(C, D), parent(D, E),
    parent(E, F), parent(F, G), parent(G, B)
- Buckets
  - 6 buckets for 6 query subgoals
  - 5 buckets for 5 shared variables (C, D, E, F, G)

Buckets (slide 1)

One bucket for each subgoal \( p(A_1, \ldots, A_n) \) of Q

- For each view \( V \), check each subgoal of the form \( p(X_1, \ldots, X_m) \) in \( V \)
- Put this view subgoal into the bucket if
  - There is a mapping from \( A_1, \ldots, A_n \) to \( X_1, \ldots, X_m \) (the only reason there might not be is if there were duplicate occurrences among the \( A_i \)'s)
  - If \( A_i \) is distinguished or shared in \( Q \), then \( X_i \) is distinguished in \( V \)
    - Intuition: \( V \) covers this query subgoal

Buckets (slide 2)

One bucket for each shared variable \( B \) in \( Q \)

- Let \( G_{G, b} \) be the set of subgoals in \( Q \) containing \( B \)
- For each view \( V \), check each possible subset \( G_V \) of the subgoals in \( V \) such that there is a containment mapping from \( G_{G, b} \) to \( G_V \)
  - Intuition: \( V \) covers all query subgoals containing \( B \)
- Put \( G_V \) into the bucket if
  - The containment mapping maps all distinguished variables in \( Q \) to distinguished variables in \( V \)
Example of filling buckets (slide 2)

grandparent(X, Y) :- parent(X, Z), parent(Z, Y)
great-grandparent(U, V) :- parent(U, S), parent(S, T), parent(T, V)
query(A, B) :- parent(A, C), parent(C, D), parent(D, E),
parent(E, F), parent(F, G), parent(G, B)

• Consider the bucket for parent(A, C)
  – A is distinguished and C is shared
  – So the bucket is empty
• Consider the bucket for parent(C, D)
  – Both C and D are shared
  – So the bucket is empty
• Similarly, buckets for other query subgoals are empty

Example of filling buckets (slide 3)

grandparent(X, Y) :- parent(X, Z), parent(Z, Y)
great-grandparent(U, V) :- parent(U, S), parent(S, T), parent(T, V)
query(A, B) :- parent(A, C), parent(C, D), parent(D, E),
parent(E, F), parent(F, G), parent(G, B)

• Consider the bucket for C
  – Need to find a containment mapping from
    \{parent(A, C), parent(C, D)\} to view subgoals
  – For grandparent view, we have
    \{parent(X, Z), parent(Z, Y)\}
  – For great-grandparent view, we have
    \{parent(U, S), parent(S, T)\}
  – What about \{parent(S, T), parent(T, V)\}??
Example of generating rewritings (slide 3)

grandparent(X, Y) :- parent(X, Z), parent(Z, Y)
great-grandparent(U, V) :- parent(U, S), parent(S, T), parent(T, V)
query(A, B) :- parent(A, C), parent(C, D), parent(D, E),
            parent(E, F), parent(F, G), parent(G, B)

Choose
  - ggp to cover C ((1, 2) → {1, 2}) and D ((2, 3) → {2, 3})
  - ggp to cover F ((4, 5) → {1, 2}) and G ((5, 6) → {2, 3})

All query subgoals are covered
- The other shared variable (E) fortunately maps to distinguished variables in ggp

query(A, B) :- ggp(A, E), ggp(E, B)

Inverse-rules algorithm example (slide 1)

- View
  - gp(X, Z) :- par(X, Y), par(Y, Z)
- Query
  - anc(X, Y) :- par(X, Y)
  - anc(X, Z) :- anc(Y, X), anc(Y, Z)
- Inverse rules for the view
  - par(X, f(X, Z)) :- gp(X, Z)
  - par(f(X, Z), Z) :- gp(X, Z)
- That is it; start evaluating the query!

Inverse-rules algorithm (slide 2)

Key ideas
- Invert view definitions: Turn view tuples into “facts” in the database that can be used to reconstruct base tables and answer queries
- Skolemization: Replace existential variables in the view definitions by Skolem functions applied to the variables in the heads

Inverse-rules algorithm example (slide 3)

- Reconstructed par
  - par(a, f(a, c)), par(b, f(b, d)), par(c, f(c, e)),
  par(f(a, c), c), par(f(b, d), d), par(f(c, e), e)
- Compute the query
  - anc(X, Y) :- par(X, Y)
  - anc(X, Z) :- anc(Y, X), anc(Y, Z)
  - anc(a, f(a, c)), anc(b, f(b, d)), anc(c, f(c, e)),
  anc(f(a, c), c), anc(f(b, d), d), anc(f(c, e), e)
  - anc(a, c), anc(b, d), anc(c, e), anc(f(a, c), f(c, e))
  - anc(a, f(c, e)), anc(f(a, c), e)

Sure answers: those without function symbols

Summary of inverse rules

- Conceptually simple
- Handles recursive queries
- Possible to remove uses of Skolem functions through more rewriting
- Requires reconstructing the base tables (the performance advantage of using materialized views is lost)
Many, many extensions…

- Object-oriented databases, semi-structured databases
- Using semantic information (e.g., constraints) in deriving rewritings
- Handling views with limited access patterns (e.g., search papers by author)
- Handling an infinite set of views (e.g., search papers by any number of keywords)
- …
- Still an active area of research