

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
 - Pottinger and Levy. “A Scalable Algorithm for Answering Queries Using Views.” *VLDB*, 2000
 - Mitra. “An Algorithm for Answering Queries Efficiently Using Views.” *ADBC*, 2001
- Inverse-rules algorithm
 - Duschka et al. “Recursive Query Plans for Data Integration.” *Journal of Logic Programming*, 2000

Brute-force algorithm

- Q : a CQ to be answered
 - V_1, V_2, \dots : views (also defined as CQ’s)
 - To find a rewriting of Q using V_i ’s
 - Try each possible join of V_i ’s as a rewriting for Q
 - Expand all V_i ’s in the join (that is, replace each V_i 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

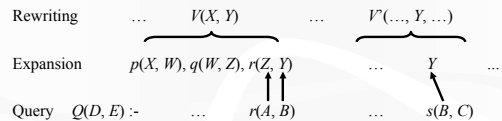
Expanding a rewriting

- Example
 - View: $\text{grandparent}(X, Z) :- \text{parent}(X, Y), \text{parent}(Y, Z)$
 - Rewriting of a query: $\text{g-g-g-grandparent}(X, Z) :- \text{grandparent}(X, Y), \text{grandparent}(Y, Z)$
 - Expansion: $\text{g-g-g-grandparent}(X, Z) :- \text{parent}(X, Y_1), \text{parent}(Y_1, Y), \text{parent}(Y, Y_2), \text{parent}(Y_2, Z)$
- 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

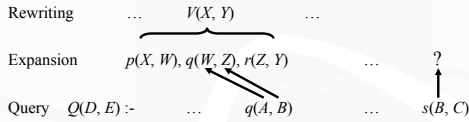
- Remember there should be a containing mapping from Q to a rewriting (with views expanded)
 - Each subgoal of Q must be covered by some view in the rewriting; that is, the query subgoal must map to some subgoal in some view
 - A distinguished variable (one that appears in the head of a rule) in Q must 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)



- 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')

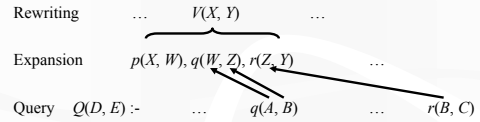
Examples (slide 2)



- 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

7

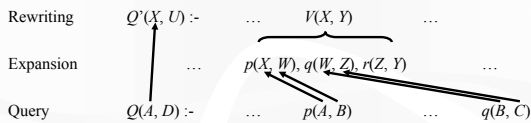
Examples (slide 3)



- 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

8

Examples (slide 4)



- 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

9

Buckets (slide 1)

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

- For each view V , check each subgoal of the form $p(X_1, \dots, X_n)$ in V
- Put this view subgoal into the bucket if
 - There is a mapping from A_1, \dots, A_n to X_1, \dots, X_n (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

10

Buckets (slide 2)

One bucket for each shared variable B in Q

- Let $\mathcal{E}_{G, B}$ be the set of subgoals in Q containing B
- For each view V , check each possible subset \mathcal{E}_V of the subgoals in V such that there is a containment mapping from $\mathcal{E}_{G, B}$ to \mathcal{E}_V
 - Intuition: V covers all query subgoals containing B
- Put \mathcal{E}_V into the bucket if
 - The containment mapping maps all distinguished variables in Q to distinguished variables in V

11

Example of filling buckets (slide 1)

- Views
 - $\text{grandparent}(X, Y) :- \text{parent}(X, Z), \text{parent}(Z, Y)$
 - $\text{great-grandparent}(U, V) :- \text{parent}(U, S), \text{parent}(S, T), \text{parent}(T, V)$
- Query
 - $\text{query}(A, B) :- \text{parent}(A, C), \text{parent}(C, D), \text{parent}(D, E), \text{parent}(E, F), \text{parent}(F, G), \text{parent}(G, B)$

- Buckets
 - 6 buckets for 6 query subgoals
 - 5 buckets for 5 shared variables (C, D, E, F, G)

12

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
 - No view subgoal has two distinguished variables
 - 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

13

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 $\{ \text{parent}(A, C), \text{parent}(C, D) \}$ to view subgoals
 - For grandparent view, we have
 - $\{ \text{parent}(X, Z), \text{parent}(Z, Y) \}$
 - For great-grandparent view, we have
 - $\{ \text{parent}(U, S), \text{parent}(S, T) \}$
 - What about $\{ \text{parent}(S, T), \text{parent}(T, V) \}$?

14

Example of filling buckets (slide 4)

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 D
 - Need to find a containment mapping from $\{ \text{parent}(C, D), \text{parent}(D, E) \}$ to view subgoals
 - For grandparent view, we have
 - $\{ \text{parent}(X, Z), \text{parent}(Z, Y) \}$
 - For great-grandparent view, we have
 - $\{ \text{parent}(U, S), \text{parent}(S, T) \}$
 - $\{ \text{parent}(S, T), \text{parent}(T, V) \}$

15

Intuition behind buckets

- Content of a bucket describes all possible ways of using a view to “cover” this particular subgoal or shared variable in the query
- Choose views to cover all subgoals and all shared variables in the query; join views to form a rewriting
 - Contents of the buckets help narrow down the choices considerably
 - Original bucket algorithm did consider how shared variables should be mapped
- The union of all rewritings formed this way gives a maximally-contained rewriting of the query (assuming no built-in predicates in the query)

16

Example of generating rewritings (slide 1)

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)

- Subgoal buckets are all empty
- Shared-variable buckets
 - C : $\{1, 2\} \rightarrow \{1, 2\}$ in gp, $\{1, 2\}$ in ggp
 - D : $\{2, 3\} \rightarrow \{1, 2\}$ in gp, $\{1, 2\}, \{2, 3\}$ in ggp
 - E : $\{3, 4\} \rightarrow \{1, 2\}$ in gp, $\{1, 2\}, \{2, 3\}$ in ggp
 - F : $\{4, 5\} \rightarrow \{1, 2\}$ in gp, $\{1, 2\}, \{2, 3\}$ in ggp
 - G : $\{5, 6\} \rightarrow \{1, 2\}$ in gp, $\{2, 3\}$ in ggp

17

Example of generating rewritings (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)

- Choose
 - C : $\{1, 2\} \rightarrow \{1, 2\}$ in gp
 - E : $\{3, 4\} \rightarrow \{1, 2\}$ in gp
 - G : $\{5, 6\} \rightarrow \{1, 2\}$ in gp
- All query subgoals are covered
- The other shared variables fortunately map to distinguished variables in gp
 - query(A, B) :- gp(A, D), gp(D, F), gp(F, B)

18

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

19

Inverse-rules algorithm

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

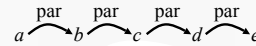
20

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(X, Y), 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!

21

Inverse-rules algorithm example (slide 2)



- Content of gp: gp(a, c), gp(b, d), gp(c, e)
 - Reconstruct par
 - par($X, f(X, Z)$) :- gp(X, Z)
 - par($f(X, Z), Z$) :- gp(X, Z)
- 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($f(c, e), e$)

22

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(X, Y), 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
- anc(a, e)

24

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