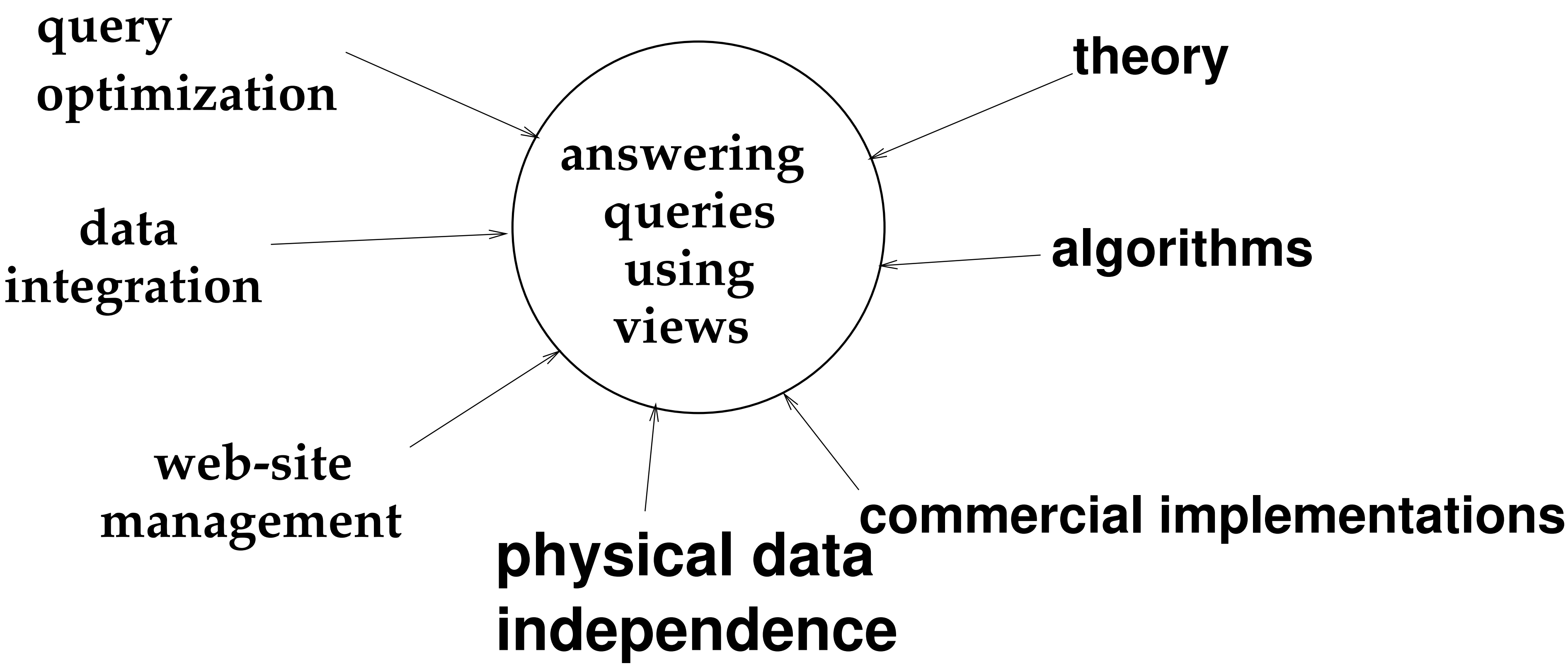
Answering Queries Using Views

Answering Queries Using Views



Introduction

The problem:

Given a query Q, and a set of view definitions,
$\mathcal{V} = V_1, \ldots, V_n$:

Is it possible to answer Q using only the views in $\mathcal{V}$?

Motivations:

- query optimization
- maintaining physical data independence
- data integration
- data warehouse design (run-time management of web-sites)

Example of Query Rewriting

Query Q, and views V_1, V_2:

$$q(X,Y) \ :\ -\ sameTopic(X,Y), cites(X,Y), cites(Y,X).$$

$$V_1(A,B) \ :\ -\ cites(A,B), cites(B,A)$$
$$V_2(C,D) \ :\ -\ sameTopic(C,D), cites(C,C_1), cites(D,D_1)$$

The query can be rewritten as:

$$Q' : q'(X,Y) : -V_1(X,Y), V_2(X,Y).$$

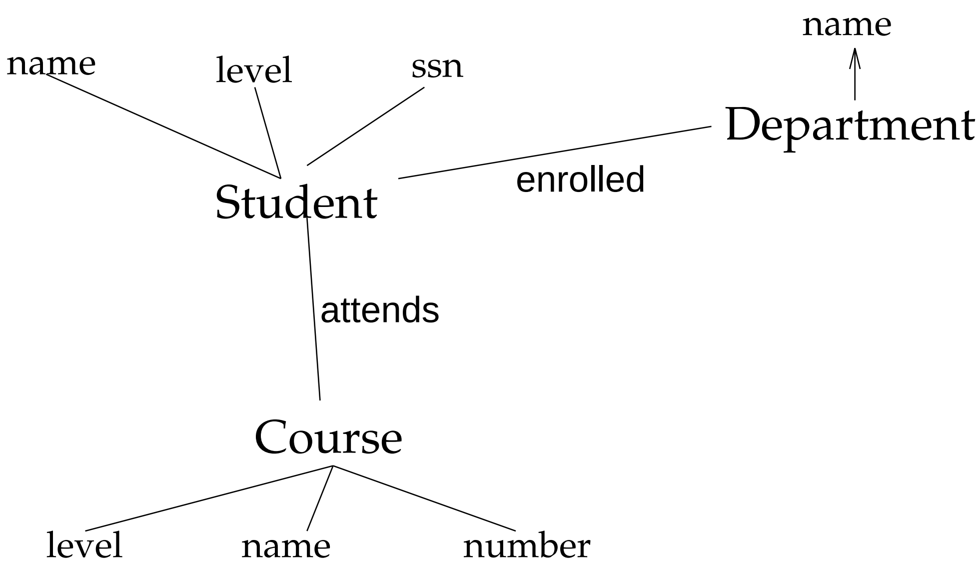
Check equivalence using the expansion:

$$q''(X,Y) : -cites(X,Y), cites(Y,X),$$
$$sameTopic(X,Y), cites(X,Y_1), cites(Y,Y_1)$$

Maintaining Physical Data Independence

def_gmap G1 as btree by
given Student.name
select Dept
where Student enrolled Dept.

def_gmap G2 as btree by
given student.name
select Course, Course.level
where Student attends Course.



The physical organization of the data should not necessarily match the logical view.

Answering Queries Using GMAPs

Given a query, how do we answer it using the GMAPs?

def_gmap G1 as btree by
given Student.name
select Dept
where Student enrolled Dept.

def_gmap G2 as btree by
given Student.name
select Course, Course.number
where Student attends Course.

Query:

select Student.name, Dept
where Student attends Course **and**
Student enrolled Dept **and** Course.number≥500.

Plan: $\sigma_{number \geq 500}(G1 \bowtie G2)$.

Outline

- Introduction & motivation
- (formal) Problem definition and intuitions
- Algorithms
- Finding certain answers.

Formal Definition

Given a query Q, and a set of views: $\mathcal{V} = V_1, \ldots, V_n$:

Q' is an equivalent rewriting of Q using $\mathcal{V}$ if:

- Q and Q' are equivalent queries, and
- Q' refers only occurrences of views in $\mathcal{V}$ or interpreted predicates.

Q' is a maximally contained rewriting of Q using $\mathcal{V}$ if:

- Q' is contained in Q, and
- there does not exist a rewriting Q_1, such that $Q' \subseteq Q_1 \subseteq Q$, and Q_1 is not equivalent to Q'.

When is a View Usable for a Query?

Query: $q(X, Z) : -r(X, Y), s(Y, Z), t(X, Z), Y > 5.$

What can go wrong?

$V_1(A, B) : -r(A, C), s(C_1, B)$

$V_2(A, B) : -r(A, C), s(C, B), C > 1$

$V_3(A, B) : -r(A, B), r_1(A, B), C > 1$

$V_4(A) : -r(A, B), s(B, C), t(A, C), B > 5$

Usability Conditions

1. There must be a mapping ψ from the tables mentioned in V to the tables mentioned in Q.
2. V must either apply the join and selection predicates on the common attributes, or select the ones on which it doesn't.
3. V must not project out any attributes needed by the query (unless they can be recovered otherwise).

A Basic Decidability Result

For conjunctive query and views (with no interpreted predicates) the following holds:

If there is an equivalent rewriting of Q using $\mathcal{V}$, then there is one with no more conjuncts than Q.

Corollary: rewriting queries using views is NP.

Rewriting queries using views is actually NP-complete (close relationship with query containment).

Bounds exist also when there are built-in comparison predicates.

Bounding the Size of the Possible Query Rewriting

Don't consider rewritings that are longer than the query.

query: $Q : q(\bar{X}) : -e_1(\bar{X}_1) \,\&\, \ldots \,\&\, e_n(\bar{X}_n)$

rewriting: $Q' : q(\bar{X}) : -v_1(\bar{Y}_1) \,\&\, \ldots \,\&\, v_m(\bar{Y}_m)$

The expansion of Q' is:

$$Q'' :$$
$$\underbrace{u_1^1(\bar{X}_1^1)\& \ldots \& u_1^{l_1}(\bar{X}_l^{l_1})}_{v_1(\bar{Y}_1)} \,\&\, \ldots \,\&\, \underbrace{u_m^1(\bar{X}_m^1)\& \ldots \& u_m^{l_m}(\bar{X}_m^{l_m})}_{v_m(\bar{Y}_m)}$$

If Q contains the rewriting Q', then there is a containment mapping from Q to the expansion of Q'. The image of Q can have at most n subgoals.

Algorithms for Rewriting

- Modifications to join enumeration algorithms:
  - Chaudhuri et al., 95, Tsatalos et al., 94.
  - The Oracle solution
- Algorithms suited for data integration:
  - Buckets, inverse rules.
- Extensions:
  - OQL [Florescu et al., 96]
  - Grouping and Aggregation [Srivastava et al.]
  - Many others

System-R Optimization Reminder

Iteration 1

a) find all possible access paths.

b) Compare their cost and keep the least expensive.

c) If the query involves one relation, stop.

Iteration 2

For each query join:

a) Consider all possible join methods on pairs from iteration 1.

b) Compare the cost of the resulting join plans, and keep the least expensive.

c) If the query involves only 2 relations, stop.

Iteration 3

...

Modifying a System-R Optimizer

Iteration 1

a1) Find all views that are *relevant* to the query.

a2) Distinguish between partial and complete solutions to the query.

b) Compare all pairs of views. If one has neither greater contribution or a lower cost than the other, prune it.

c) If there are no partial solutions, stop.

Iteration 2: for each query join:

a1) Consider joining all partial solutions found in the previous iteration using all possible join methods and trying all possible subsets of join predicates.

a2) Distinguish between complete and partial solutions.

b) If any newly generated solution is either not relevant to the query, or dominated by another, prune it.

c) If there are no partial solutions, stop.

Iteration 3

. . .

The Bucket Algorithm [Information Manifold]

$q(\bar{X}) :-$	g_1	g_2	$\cdots$	g_n
relevant sources for each subgoal	$S1$ $S2$	$S3$ $S4$	$S5$ $S6$	$S7$ $S8$

1. Consider relevance of sources to each subgoal in isolation
2. Consider only logical plans in the cartesian product of the buckets.

Inverse-rule Algorithm

Inverse rules reconstruct the original relations from the views

Given the view:

V(dept, c-name) :-
    Enrolled(s-name,dept), Registered(s-name,c-name).

we construct the inverse rule:

Enrolled(f_1(dept,X), dept) :- V(dept,X)
Registered(f_1(Y, c-name), c-name) :- V(Y,c-name)

THe rewriting includes the inverse rules + the original query.

Comparison of the Algorithms

Dynamic programming:

- Only one that makes a cost-based choice.
- Don't deal with contained rewritings, union rewritings, large number of views.

Bucket algorithm:

- Deals well with large number of views differing on their predicates.
- Requires containment checks, and significant modifications for binding patterns, functional dependencies.

Inverse rules:

- Modular, can be extended to binding patterns, functional dependencies, recursive queries.
- Doesn't deal well with predicates, and requires significant further optimizations.

Finding All the Answers

When a rewriting is equivalent to the query, we'll definitely get all the answers.

Maximally containment of a rewriting is only w.r.t. a specific query language.

Complexity of finding all the answers?

[Abiteboul & Duschka, PODS98]: surprisingly hard!

Given extensions $v_1, \ldots, v_n$ for the views in V, let $\mathcal{D}$ be the set of all databases that agree on $v_1, \ldots, v_n$.

$\bar{t}$ is a certain answer for Q given $v_1, \ldots, v_n$, if $\bar{t} \in Q(D)$ for every $D \in \mathcal{D}$.

Why & When is This Hard?

Distinguish: open-world assumption vs. closed-world assumption about the views.

The following encodes graph colorability:

Views:

V1(X) :- color(X,Y) (set of nodes)
V2(Y) :- color(X,Y) (the set {red, green, blue})
V3(X,Y) :- edge(X,Y) (set of edges)

Query:

q(c) :- edge(X,Y), color(X,Z), color(Y,Z)

Potpourri

- Original works: Larson and Yang, VLDB 85,87.
- Grouping and aggregation: [Srivastava et al., VLDB-96]
- Description Logics: [Beeri et al., PODS-97]
- Infinite sets of views [Levy et al., PODS-96, Vassalos and Papakonstantinou, VLDB-97]
- Polynomial-time cases [Chekuri and Rajaraman, ICDT-97]
- Inclusion dependencies [Gryz, ICDE-98]
- Views with unions [Afrati et al., ICDT-99, Duschka, Ph.D thesis]