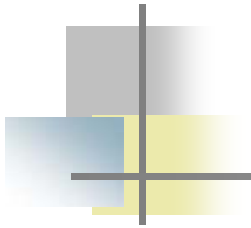
Answering Queries Using Views

Theory Of Database Systems

Roussos Ioannis



Definition

Answering Queries Using Views

(also known as **Rewriting Queries Using Views**)

Suppose we are given a query *Q* over a database schema, and a set of view definitions *V1, . . . , Vn* over the same schema.

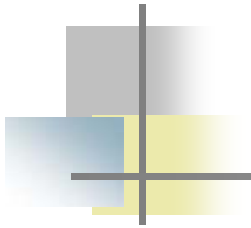
- Is it possible to answer the query *Q* using *only* the answers to the views *V1, . . . , Vn*?

Alternatively,

- What is the maximal set of tuples in the answer of *Q* that we can obtain from the views?
- If we can access both the views and the database relations, what is the cheapest query execution plan for answering *Q*?

Relevant applications

- Query optimization
  - using previously materialized views can speed up query processing
- Database design
  - view definitions provide a mechanism for supporting the independence of the physical view of the data and its logical view
- Data integration systems
  - contents of the sources are described as views over the mediated schema.
- Data warehouse and web-site design
  - choose a set of views to materialize/precompute in order to improve the performance
  - views precomputed offline (fast at runtime)
- Semantic data caching



Query Optimization

- Use previously materialized views in order to speed up query processing.
  1. detect when a view is logically usable for answering a query
  2. make a judicious *cost-based* decision on when to use the available views

- *Example Schema*:

  Prof(name, area)

  Teaches(prof, course, quarter)

  Registered(student, course, quarter)

  Course(title, number)

Query Optimization

- Query: *fetch students who attended Ph.D-level classes taught by professors in the Database area*

  select student

  from Teaches, Prof, Registered, Course

  where Prof.name=Teaches.prof and Teaches.course=Registered.course
  and Teaches.quarter=Registered.quarter and
  Registered.course=Course.title and Course.number ≥ 500 and
  Prof.area="DB".

- View: *attendance records of graduate level courses and above*

  create view Graduate as

  select student, course, number, quarter

  from Registered, Course

  where Registered.course=Course.title and Course.number ≥ 400.

Query Optimization

- *Query Rewriting*:

  select student

  from Teaches, Prof, **Graduate**

  where Prof.name=Teaches.prof and

  Teaches.course=Graduate.course and
  Teaches.quarter=Graduate.quarter and

  Graduate.number ≥ 500 and Prof.area="DB".

- Resulting evaluation is cheaper: the view Graduate has already
  - performed the join between Registered and Course
  - pruned the non-graduate courses
- BUT **not** always: depends on indexes on DB relations and on views…

Maintaining Physical Data Independence

- *Motivation:* separation between
  - the logical view of the data (e.g., as tables with their named attributes).
  - the physical view of the data (i.e., how it is layed out on disk).
- Easy in Relational Database Systems (1-1 map).

  But what about:
  - object-oriented systems?
  - semi-structured data?
- *Solution:* use views as a mechanism for describing the storage of the data.

Maintaining Physical Data Independence: *GMAPs*

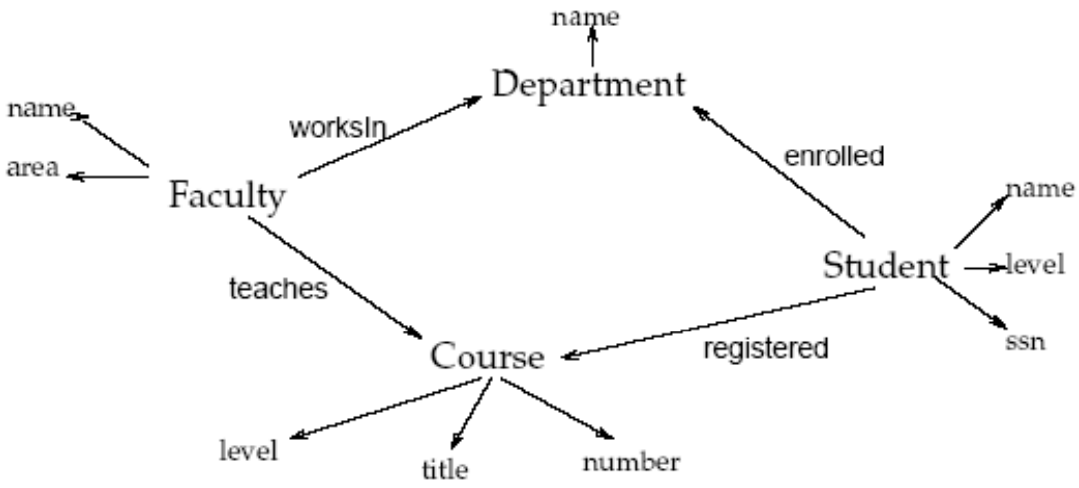
- e.g., [TSI96] described the storage of the data using **GMAP**s (generalized multi-level access paths), expressed over the conceptual model of the database.
- A GMAP describes the physical organization and indexes of the storage structure:
  - actual data structure used to store a set of tuples (e.g., a B-tree, hash file etc.).
  - describe the content of the structure, much like a view definition.

Maintaining Physical Data Independence: *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.level
where Student Registered Course.

def gmap G3 as btree by
given Course.number
select Dept, Course
where Student Registered Course
and Student enrolled Dept.



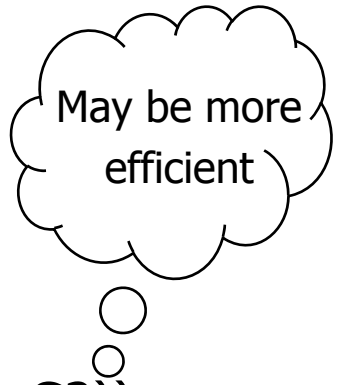
Maintaining Physical Data Independence: *GMAPs*

- answering a query amounts to finding a way of rewriting a query using the views described by the GMAPs.
- Difference from Query optimization:
  - since all the data is stored in the GMAPs, we ***must*** use the views to answer a given query.
- Query:

  select Student.name, Dept

  where Student Registered Course and

  Student enrolled Dept and Course.number=500.
- Possible Answers:
  - $\Pi_{\text{Student.name,Dept}}\,(\sigma_{\text{Course.number=500}}(\text{G1 Join G2}))$
  - $\Pi_{\text{Student.name,Dept}}\,(\sigma_{\text{Course.number=500}}(\text{G3 Join G1 Join G2}))$



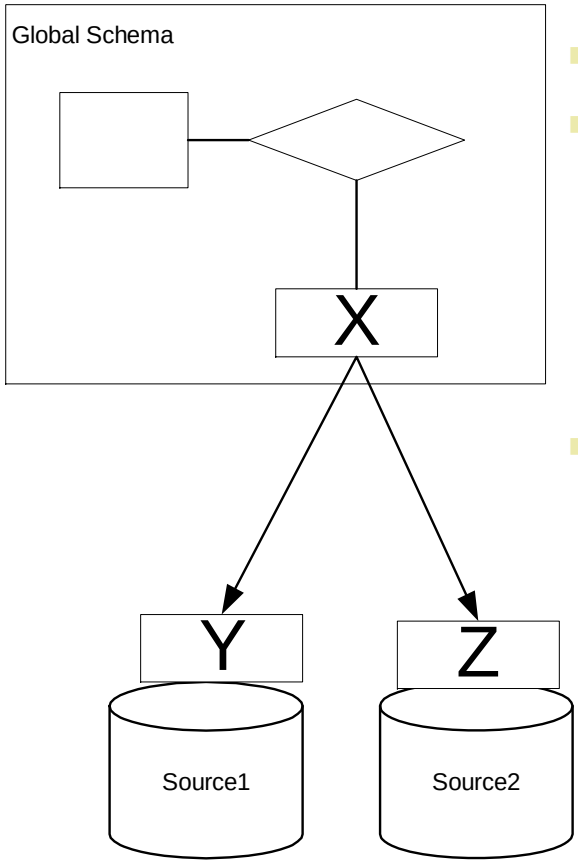
Data Integration

- A Data Integration System (a.k.a. a mediator system) provides a *uniform* query interface to a multitude of autonomous heterogeneous data sources.
  - traditional databases,
  - legacy systems
  - structured files
- The Data Integration System should:
  - *Free* the user from having to find the data sources relevant to a query, interact with each source in isolation, and manually combine data from the different sources.
  - Provide flexible, homogeneous and *transparent access* to several (possibly) distributed, autonomous, and heterogeneous sources.
  - *Automate* the integration process as much as possible.

Data Integration

- In order to provide a uniform interface:
  - exposes to the user only a *mediated schema*.
    - A mediated schema is a set of *virtual* relations.
  - Contain a set of source descriptions.
    - contents of each source.
    - the attributes that can be found in each source.
    - constraints on the contents of each source.
- Two approaches:
  - **G**lobal **A**s **V**iew (**GAV**)
    - The mediated schema is defined as a view over the source schemas
  - **L**ocal **A**s **V**iew (**LAV**)
    - Sources are described as views over the mediated schema

Data Integration
GAV

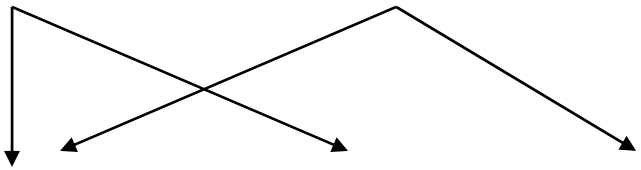


- X is defined as a view over Y and Z
- Mapping Example:

  Faculty(name,area) -> Person(id, name),
  Expert(id, area)

  Teaches(name, c-number, trim) -> Person(id,name),
  Gives(id, c-number, trimester)
- Query answering means ***unfolding*** the mappings/rules

Q(name):-
Faculty(name, "IS"), Teaches(name, c,2003/2")



Person(id, name), Expert(id,"IS"), Gives(id, c, "2003/2")

Data Integration
LAV

- The contents of a data source are described as a view over the mediated schema.
- Better than GAV:
  - Addition/deletion of new data sources.
  - Specification of constraints on contents of sources.
  - Greater expressive power.
- Answering a query over the mediated schema $\Rightarrow$ find a way to answer a query using a set of views.
- In data integration systems we attempt to find a query expression that provides the *maximal answers* from the views (instead of an equivalent rewriting).

Data Integration

- Example mediated schema:
  - Teaches(prof, course-number, univ)
  - Course(title, univ, number)
- Data sources (defined as views):

```
create view DB-courses as
select Course.title, Teaches.prof, Course.number, Course.univ
from Teaches, Course
where Teaches.course-number=Course.number and Teaches.univ=Course.univ
    and Course.title="Database Systems".

create view NTUA-phd-courses as
select Course.title, Teaches.prof, Course.number, Course.univ
from Teaches, Course
where Teaches.course-number=Course.number and
Course.univ="NTUA" and Teaches.univ="NTUA" and Course.number ≥ 500.
```

Data Integration

- Query: "who teaches Database courses at NTUA"

  select prof

  from DB-courses

  where univ="NTUA".

- Query: "ask for all the graduate-level courses (i.e., number 400 and above) being offered at NTUA".
  - Cannot find all the answers to this query
  - The system can attempt to find the *maximal set* of answers available from the sources:

  select title, number

  from DB-courses

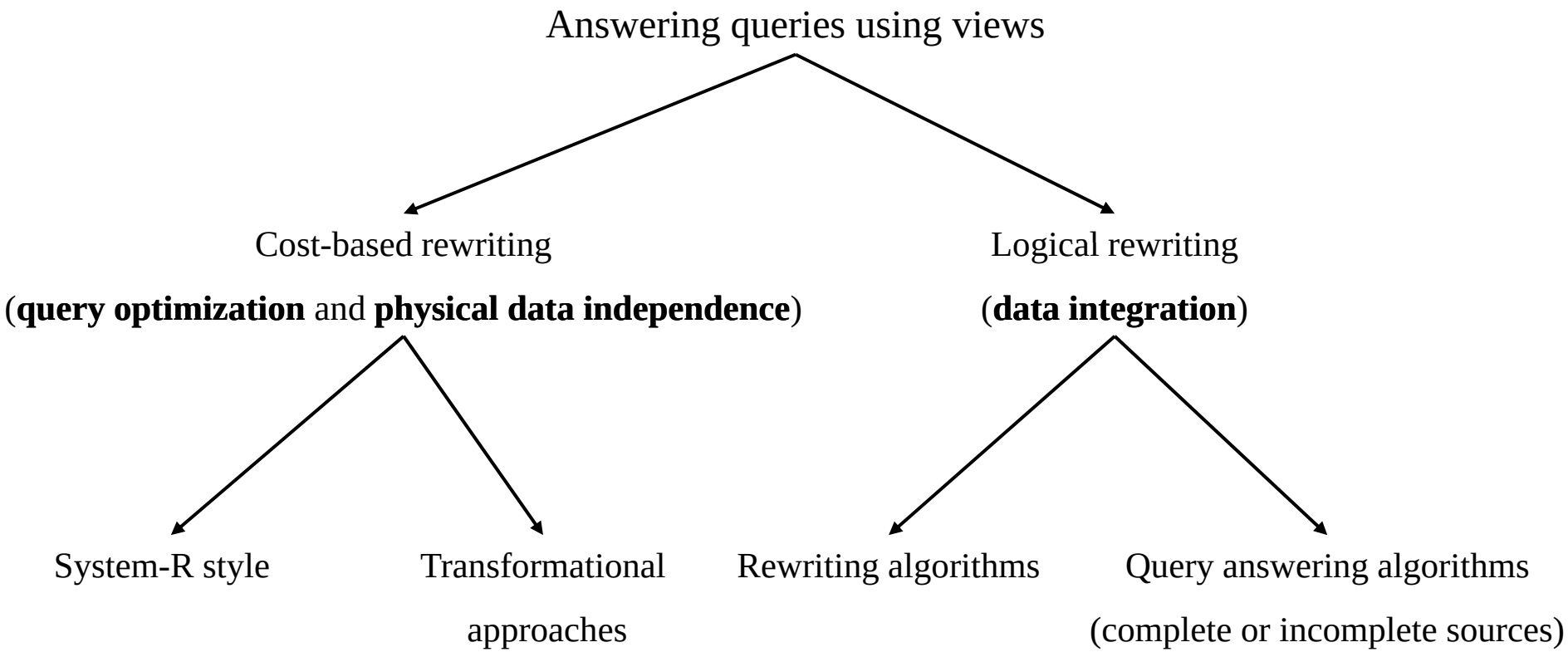
  where univ="NTUA" and number $\geq$ 400

  union

  select title, number

  from NTUA-phd-courses.

A taxonomy of the field



output of the algorithm for AQUV

Given a query Q, and a set of views V

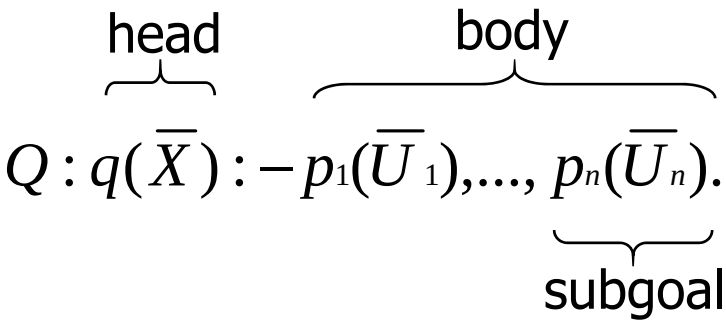
data integration

- produce an *expression Q′ (new query)* that references only the views
- the rewriting is either *equivalent* to *Q* or *contained* in *Q*
- logical correctness suffices
- large number of views

query optimization and *maintenance of physical data independence*

- produce a query *execution plan* for answering *Q* using the views (and possibly the database relations).
- the rewriting must be an *equivalent* to *Q* in order to ensure the correctness of the plan.
- also need to find the *cheapest* plan using the views.
- number of views is roughly comparable to the size of the schema.

Queries and Views



- We require that the query be safe.
- EDB/IDB predicates.
- Predicate dependencies.
- Recursive datalog programs.
- etc.

Query containment and equivalence

Definition

- A query Q1 is said to be *contained* in a query Q2, denoted by Q1 $\subseteq$ Q2, if for any database D, the set of tuples computed for Q1 is a subset of those computed for Q2, i.e. Q1(D) $\subseteq$ Q2(D).

  The two queries are said to be *equivalent* if Q1 $\subseteq$ Q2 and Q2 $\subseteq$ Q1.

Rewriting of a Query Using Views

- Given a query Q and a set of view definitions $V_1,\ldots,V_m$ a rewriting of the query using the views is a query expression Q' that refers *only* to the views $V_1,\ldots,V_m$.
- In practice, we may also be interested in rewritings that can also refer to the *database relations*, but this case does not introduce new difficulties.
- Two types of query rewritings:
  - *equivalent rewritings* (query optimization and maintaining physical data independence).
  - *maximally-contained rewritings.*

Rewriting of a Query Using Views

Definition (*Equivalent rewritings*)

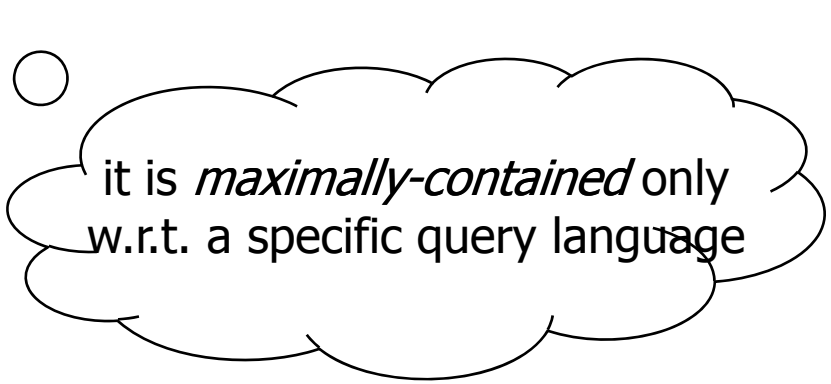
- Let Q be a query and V = $V_1,\ldots,V_m$ be a set of view definitions. The query Q' is an equivalent rewriting of Q using V if:
  - Q' refers only to the views in V, and
  - Q' is equivalent to Q.

Definition (*Maximally-contained rewritings*)

- Let Q be a query, V = $V_1,\ldots,V_m$ be a set of view definitions, and L be a query language. The query Q' is a maximally-contained rewriting of Q using V w.r.t. L if:
  - Q' is a query in L that refers only to the views in V,
  - Q' is contained in Q, and
  - there is no rewriting $Q_1 \in L$, such that $Q' \subseteq Q1 \subseteq Q$ and Q1 is not equivalent to Q'.

Rewriting of a Query Using Views

- How can we find all the possible answers to the query, given a set of view definitions and their extensions?
  - If we find a rewriting that is *equivalent* to the query, then we are guaranteed to find *all* the possible answers.
  - However, a *maximally-contained* rewriting of a query using a set of views does **not** always provide all the possible answers that can be obtained from the views.



Rewriting of a Query Using Views: *Certain answers*

Definition

Let Q be a query and $V = V_1,\ldots,V_m$ be a set of view definitions over the database schema $R_1,\ldots,R_n$. Let the sets of tuples $u_1,\ldots,u_m$ be extensions of the views $V_1,\ldots,V_m$, respectively.

The tuple *a* is a certain answer to the query Q under the *closed-world* assumption given $u_1,\ldots,u_m$ if *a* is an answer to Q for any database D such that $V_i(D) = v_i$ for every i, $1 \leq i \leq m$.

The tuple *a* is a certain answer to the query Q under the *open-world* assumption given $u_1,\ldots,u_m$ if *a* is an answer to Q for any database D such that $V_i(D) \supseteq v_i$ for every i, $1 \leq i \leq m$.

- *closed-world* assumption: the view extensions are complete.
- *open-world* assumption: the views may be partial.

Rewriting of a Query Using Views: *Certain answers*

- Intuition behind the definition:
  - The extensions of a set of views do not define a unique database instance.
  - Given the extensions of the views we have only partial information about the real state of the database.
  - A tuple is a certain answer of the query *Q* if it is an answer for *any* of the possible database instances that are consistent with the given extensions of the views.

Rewriting of a Query Using Views: *Certain answers*

Example:

- Database schema: $R(A,B)$
- Views:
  - V1(X) :- R(X,Y). u_1 = {c1}.
  - V2(Y) :- R(X,Y). u_2 = {c2}.
- Query: Q(X,Y) :- R(X,Y).
- *closed-world* assumption: we can infer that the tuple ($c1$, $c2$) *must* be in the relation R, and hence it is a certain answer to Q.
- *open-world* assumption: since $V1$ and $V2$ are not necessarily complete, the tuple ($c1$, $c2$) need *not* be in R.
  - For example, R may contain the tuples ($c1$, d) and (e, $c2$) for some constants d and e. Hence, ($c1$, $c2$) is *not* a certain answer to Q.

When is a View Usable for a Query

- A view can be useful for a query if:
  - the set of relations it mentions overlaps with that of the query.
  - it selects some of the attributes selected by the query.
  - if the query applies predicates to attributes that it has in common with the view, then the view must apply either equivalent or logically weaker predicates.

When is a View Usable for a Query

```
select Advises.prof, Advises.student, Registered.quarter
from Registered, Teaches, Advises
where Registered.course=Teaches.course and
    Registered.quarter=Teaches.quarter and
    Advises.prof=Teaches.prof and
    Advises.student=Registered.student and
    Registered.quarter ≥ "winter98".

create view V1 as
select Registered.student, Teaches.prof, Registered.quarter
from Registered, Teaches
where Registered.course=Teaches.course and
    Registered.quarter=Teaches.quarter and
    Registered.quarter > "winter97".
```

When is a View Usable for a Query

```
create view V2 as
select Registered.student, Registered.quarter
from Registered, Teaches
where Registered.course=Teaches.course and
      Registered.quarter=Teaches.quarter and
      Registered.quarter ≥ "winter98".
```

does not select the attribute Teaches.prof

```
create view V3 as
select Registered.student, Teaches.prof,
     Registered.quarter
from Registered, Teaches
where Registered.course=Teaches.course and
     Registered.quarter ≥ "winter98".
```

does not apply the necessary join predicate Registered.quarter=Teaches.quarter

When is a View Usable for a Query

More formal…

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 in Q on the attributes of the tables in the domain of ψ, or must apply to them a logically weaker selection, and select the attributes on which predicates need to still be applied.
3. V must not project out any attributes of the tables in the domain of ψ that are needed in the selection of Q, unless these attributes can be recovered from another view.

Incorporating Materialized views into Query Optimization

- Extend a System-R style optimizer to accommodate usage of views:
  - Decide in a *cost-based* fashion when to use views to answer a query.
  - Output: execution plan for the query.
- Principles of System-R optimization:
  - Bottom-up approach to building execution plans.
  - Begin by constructing plans of size 1.
  - In phase n, consider plans of size n, by combining plans obtained in the previous phases.
  - Terminate when all relations are covered.
- Equivalent classes.

Query Optimization: System-R style optimization

Conventional optimizer

Iteration 1

a) Find all possible access paths.

b) Compare their cost and keep the least expensive (of each equivalent class).

c) If the query has one relation, stop.

Optimizer using views

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.

Query Optimization: System-R style optimization

Conventional optimizer	**Optimizer using views**
For each query join:	
Iteration 2…	*Iteration 2…*
a) Consider joining the *relevant* access paths found in the previous iteration using all possible join methods.	a1) Consider joining *all* partial solutions found in the previous iteration using all possible equi-join methods and trying *all* possible subsets of join predicates.
	a2) Distinguish between complete and partial solutions.
b) Compare the cost of the resulting join plans and keep the least expensive.	b) If any newly generated solution is either not relevant to the query, or dominated by another, prune it.
c) If the query has only 2 relations, stop.	c) If there are no partial solutions, stop.

Query Optimization: System-R style optimization

Example

- Relations:
  - Enrolled(s-name, dept), Registered(s-name, c-name, quarter), Course(title, number).
- Views:

```
create view V2 as
select s-name, c-name, number
from Registered, Course
where Registered.c-name=course.c-name.
```

```
create view V1 as
select s-name, dept
from Enrolled
```

```
create view V3 as
select dept, c-name
from Registered, Enrolled
where Registered.s-name=Enrolled.s-name.
```

Query Optimization: System-R style optimization

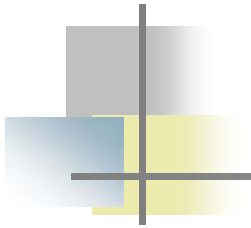
- Query: Fetch all the students attending Ph.D level classes, and the departments in which they're enrolled.

  select s-name, dept

  from Registered, Enrolled, Course

  where Registered.s-name=Enrolled.s-name and

  Registered.c-name=Course.title and number $\geq$ 500.
- First Iteration: all views are relevant to the query.
  - Choose best access path to each view.
- Second iteration: consider all possible methods to join pairs of plans produced in the first iteration.
  - Consider V1 Join V2 (+ selection on the number attribute).
- Third iteration:
  - Consider V3 Join V1 Join V2 (it may be cheaper than V1 Join V2, according to existing indexes).



Data Integration

- Algorithms for query optimization:
  - Small number of views.
  - Require only equivalent rewritings.
  - Not consider answers resulting from *unions* of views.
  - Output: Query execution plan.
- Data Integration.
  - Large number of views.
  - We are often *not* able to find an *equivalent* rewriting of the query using the source views.
    - best we can do is find the *maximally-contained* rewriting of the query.
  - Output: a query referring to the view relations.

Data Integration

- Bucket Algorithm.
  1. Create a bucket for each sub-goal in the query containing the views that have the same sub-goal and there is a mapping.
  2. Consider conjunctive rewriting for each element of the Cartesian product of the buckets, and check whether it is contained or can be made to be contained in the query.
- Inverse Rules Algorithm.
  - Construct set of rules that invert the view definition.
- MiniCon Algorithm.

Data Integration

- Bucket Algorithm.

Pros	Cons
Considers each sub-goal in isolation.	Considers each sub-goal in isolation.
To some degree takes into account context to prune search space.	Considers Cartesian product of buckets.
Extends naturally to cases where the query is a union of conjunctive queries.	It is hard to recover projected away attributes w/o additional knowledge.

Data Integration

- Inverse Rules Algorithm.

Pros	Cons
Simplicity and modularity (easy to add functional dependencies, binding patterns, recursive queries).	Needs additional optimizations as rule folding and removing irrelevant tuples.
Returns maximally contained rewriting even with arbitrary recursive Datalog programs.	Does not handle arithmetic comparison predicates

Data Integration

- *MiniCon Algorithm*

1a) Begin like the Bucket Algorithm

b) Form the MiniCon Descriptors

For sub-goal g in the query Q mapped to sub-goal g' in view V (bucket), look at the variables Q and consider the join predicates to find the minimal additional set of sub-goals in Q that must be mapped to sub-goal in V in order V be usable.

2) Combine MCD-s

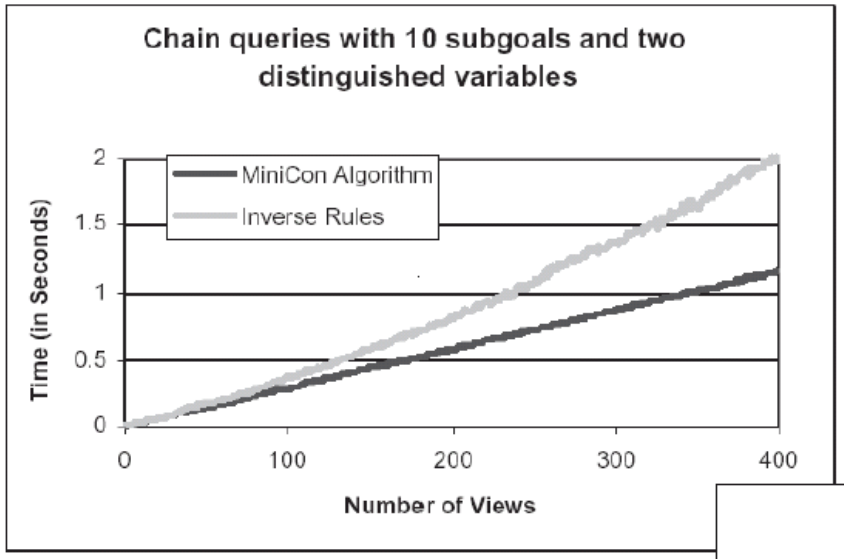
- proceed as in the bucket algorithm but consider rewritings involving only disjoin MCD-s.
- no need of containment check (additional speedup) for each rewriting.

Data Integration

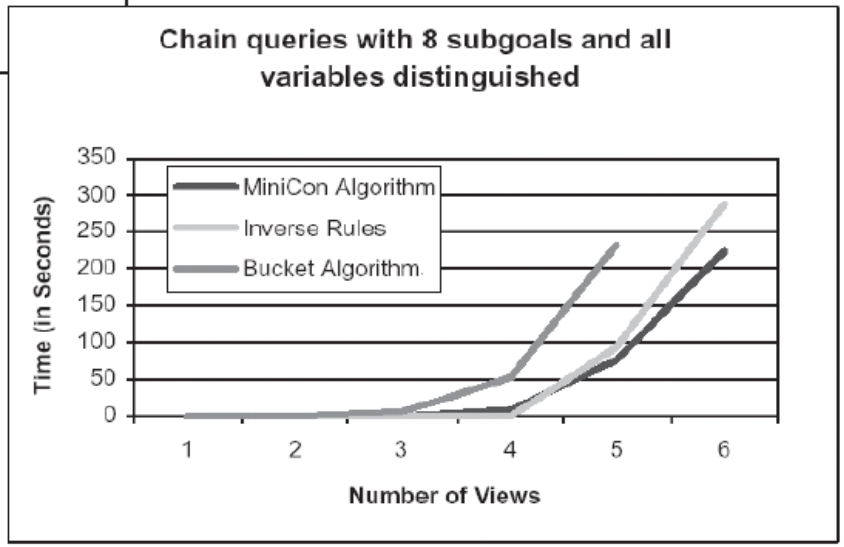
- q(D) :- Major(S, D), Registered(S, 444, Q), Advises(P, S)
- V1(dept) :-Major(student,dept), Registered(student,444,quarter)
- V2(prof,student,area) :-Advises(prof,student), Prof(prof,area)
- V3(dept,c-number) :-Major(student,dept),
  Registered(student,cnumber,quarter),Advises(prof,student)

Major(S,Q)	*Registered(S,C,Q)*	*Advises(P,S)*
~~V1(D')~~	~~V1(D')~~	V2(P,S,A')
V3(D',C')	V3(D',C')	V3(D',C')

Data Integration



- Chain queries with only few rewritings



- Chain queries with many rewritings

Completeness of query rewritings

- Important question: is the algorithm complete?
  - given a query Q and a set of views V, will the algorithm find a rewriting of Q using V when one exists?
- Class of queries and views expressed as conjunctive queries [LMSS95]
  - when the views do not contain comparison predicates, and the query Q has n subgoals, then there exists an equivalent-rewriting of Q using V *only* if there is a rewriting with at most *n* subgoals.
  - Immediate corollary: the problem of finding and equivalent rewriting of a query using a set of views is in NP, because it suffices to guess a rewriting and check its correctness.

Completeness of query rewritings

- The *bucket algorithm* is guaranteed to be complete when the queries and the view do not contain comparison predicates.

  Also, the bucket algorithm produces the maximally contained rewriting of the query if we consider rewritings that are unions of conjunctive queries.
- The problem of finding a rewriting is NP-hard for two independent reasons:
  1. the number of possible ways to map a single view into the query
  2. the number of ways to combine the mappings of dierent views into the query.
- *Very Important*: the rewriting of the query that produces the most *eficient* plan for answering the query may have more conjuncts that the original query.

The need for recursive rewritings

- When we cannot find an *equivalent* rewriting of the query using a set of views, we consider the problem of finding *maximally-contained* rewritings.
- Our hope is that when we apply the maximally-contained rewriting to the extensions of the views, we will obtain the set of *all certain answers* to the query.
- There are several contexts where in order to achieve this goal we need to consider *recursive datalog rewritings* of the query.
  - Recursive query (obvious) [DG97].
  - Binding patterns [DL97].
  - Functional dependencies [DL97].

Need for recursive rewritings *functional dependencies*

- Database schema:

  schedule(Airline,Flight_No,Date,Pilot,Aircraft)

- Functional dependencies:

  Pilot $\rightarrow$ Airline and

  Aircraft $\rightarrow$ Airline

- View:

  v(D,P,C) :- schedule(A,N,D,P,C)

- Query asking for pilots that work for the same airline as *"Mike"*:

  q(P) :- schedule(A,N,D,`mike',C), schedule(A,N',D',P,C')

Need for recursive rewritings *functional dependencies*

- The view v *doesn't* record the airlines that pilots work for:
  - deriving answers to q requires using the *functional dependencies*.
- In general, for any value of n, the following conjunctive rewriting is a contained rewriting:

$q_n(P)$: $v(D_1, \text{mike}, C_1), v(D_2, P_2, C_1), v(D_3, P_2, C_2), v(D_4, P_3, C_2), \ldots,$
$v(D_{2n-2}, P_n, C_{n-1}), v(D_{2n-1}, P_n, C_n), v(D_{2n}, P, C_n)$

- Moreover, for each n, $q_n(P)$ may provide answers that were not given by q_i for $i < n$.

Need for recursive rewritings *functional dependencies*

- *Conclusion*:
  - we cannot find a maximally-contained rewriting of this query using the views if we *only* consider non-recursive rewritings.
- The maximally-contained rewriting is the following datalog program:

```
relevantPilot("mike").
relevantAirCraft(C) :- v(D, "mike", C).
relevantAirCraft(C) :- v(D,P,C), relevantPilot(P).
relevantPilot(P) :- relevantPilot(P1), relevantAirCraft(C), v(D1, P1, C),
    v(D2, P, C).
```

Need for recursive rewritings
Binding Patterns

- EDBs: DBpapers(X), Cites(X,Y), AwardPaper(X).
- Views:

  $DBSource^{f}$ (X) :- DBpapers(X)

  $CitationDB^{bf}$(X,Y) :- Cites(X,Y)

  $AwardDB^{b}$(X) :- AwardPaper(X)
- Query:

  Q(X) :- AwardPaper(X).
- We can follow any length of citation chains begininning from Dbpapers ➔ NO bound on the length of a rewriting.

  Q'(X) :- DBSource(X), AwardDB(X)

  Q'(X) :- DBSource(V), CitationDB(V,X_1), … , CitationDB(X_n,X), AwardDB(X).

Need for recursive rewritings
Binding Patterns

- A maximally-contained rewriting of the query using the views can always be obtained with a recursive rewriting [DL97]:

papers(X) :- DBsource(X)
papers(X) :- papers(Y), CitationDB(Y,X)
Q'(X) :- papers(X), AwardDB(X).

Finding the certain answers

- If Q' is an *equivalent-rewriting* of a query Q using the set of views V, then it will *always* produce the same result as Q, independent of the state of the database or of the views.

  Q' will always produce *all* the *certain answers* to Q for any possible database.

- When Q' is a *maximally-contained* rewriting of Q using the views V it may produce only a *subset* of the answers of Q for a given state of the database.
  - The maximality of Q' is defined only w.r.t. the other possible rewritings in a particular query language L that we consider for Q'

Finding the certain answers

- Under the *open-world assumption*, [AD98] show that in many practical cases, finding all the certain answers can be done in polynomial time.
  - The problem becomes NP-hard as soon as we allow ***union*** in the language for defining the views, or allow the predicate $\neq$ in the language defining the query

- Under the *closed-world* assumption the situation is even worse. Even when both the views and the query are defined by conjunctive queries without comparison predicates, the problem of finding all certain answers is already NP-hard.
  - Proof by reduction of the problem of graph 3-colorability to the problem of finding all the certain answers.

Finding the certain answers Proof for CWA

- Let G = (V, E) be an arbitrary graph
- Views:

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

- Query:

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

- c is a certain answer to q if and only if the graph encoded by edge is not three-colorable

References

- Duschka, Levy. "Recursive plans for information gathering", International Joint Conference on Articial Intelligence, 1997.
- Duschka, Genesereth, "Answering recursive queries using views", PODS 1997.
- Halevy, "Answering queries using views: A survey", VLDB Journal 2001
- Halevy, "Answering queries using views: A survey", Technical Report 1999
- Halevy, "Theory of Answering queries using views", SIGMOD 2000
- Levy, Mendelzon, Sagiv, Srivastava, "Answering queries using views", PODS 1995.
- Levy, Rajaraman, Ullman, "Answering queries using limited external processors", PODS 1996.
- Srivastava, Dar, Jagadish, Levy, "Answering SQL queries using materialized views", VLDB 1996
- Yang, Larson "Query transformation for PSJ-queries." VLDB 1987.