Answering Queries Using Views

Alon Y. Halevy, Anand Rajaraman, Joann J. Ordille

Presentation: Haley Li
Discussion: Daniel Long

Slides adapted from Jeffrey Niu, Nalin Munshi, Rachel Pottinger
Introduction

Main Papers:


Survey for background:

Background – Views

A view is a stored query. It can be saved and reused.

In SQL:

```
CREATE VIEW JapaneseProducts AS
SELECT Name, Price, Category, Manufacturer
FROM Product, Company
WHERE Product.Manufacturer = Company.Cname
AND Company.Country = 'Japan'
```

```
Product(Name, Price, Category, Manufacturer)
Company(Cname, StockPrice, Country)
```
Can rewrite queries using views (datalog):

Query:

\[
q(code) :- \text{Airport}(code, \text{city}), \text{Feature}(\text{city}, \text{POI})
\]

View definition:

\[
\text{feature-code}(code, \text{POI}) :- \text{Airport}(code, \text{city}), \text{Feature}(\text{city}, \text{POI})
\]

Rewriting using view:

\[
q(code) :- \text{feature-code}(code, \text{POI})
\]
Applications

1. Query Optimization
   • Build query plan using views

2. Data Integration
   • Query from multiple heterogeneous sources
Query Optimization

Goal:
• Use views alongside base relations to answer a query
  • Reuse materialized view
• Rewrite with views should yield same answer as original query
• Maintain physical data independence
  • Only need to work with views
  • Views do not change when modifying storage schema [1]

Taken from Jeffrey Niu’s, Nalin Munshi’s slides
Containment

Query $Q_1$ is contained in $Q_2$ ($Q_1 \subseteq Q_2$) if for any database $D$,

$$Q_1(D) \subseteq Q_2(D)$$

In other words, the tuples returned from running $Q_1$ on $D$ are a subset of running $Q_2$ on $D$. 
Containment

Query $Q_1$:

$q_1$(code) :- Airport(code, city)

Query $Q_2$:

$q_2$(code) :- Airport(code, “New York”)
## Containment

**Query $Q_1$:**

$q_1(\text{code}) :- \text{Airport(\text{code, city})}$

**Query $Q_2$:**

$q_2(\text{code}) :- \text{Airport(\text{code, "New York"})}$

| $Q_1(D)$: Code    | JFK | LGA | YVR | YYYZ | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_2(D)$: Code</td>
<td>JFK</td>
<td>LGA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Hence, we have $Q_2 \subseteq Q_1$
Equivalent Rewriting

Given query $Q$ and views $V = \{V_1, ..., V_m\}$, a query $Q'$ is an equivalent rewriting of $Q$ using $V$ if $Q$ and $Q'$ are contained within each other [1] (return the exact same answers).

   Taken from Jeffrey Niu's, Nalin Munshi's slides
Given query $Q$ and views $V = \{V_1, ..., V_m\}$, a query $Q'$ is an **equivalent rewriting** of $Q$ using $V$ if $Q$ and $Q'$ are contained within each other [1] (return the exact same answers).

**Query:**

$q(code) :- \text{Airport}(code, \text{city}), \text{Feature}(\text{city}, \text{POI})$

**View $V_1$:**

$\text{feature-code}(code, \text{POI}) :- \text{Airport}(code, \text{city}),
\text{Feature}(\text{city}, \text{POI})$

**Equivalent rewrite using $V_1$:**

$q(code) :- \text{feature-code}(code, \text{POI})$
Equivalent Rewriting

Given query $Q$ and views $V = \{V_1, \ldots, V_m\}$, a query $Q'$ is an equivalent rewriting of $Q$ using $V$ if $Q$ and $Q'$ are contained within each other [1] (return the exact same answers).

Query:

$q(\text{code}) :- \text{Airport(\text{code, city}), Feature(\text{city, POI})}$

View $V_2$:

$\text{Beach-code(\text{code}) :- \text{Airport(\text{code, city}), Feature(\text{city, "Beach"})}}$

Non-equivalent rewrite using $V_2$:

$q(\text{code}) :- \text{Beach-code(\text{code})}$

Only gets “Beach” results from Feature

---


Taken from Jeffrey Niu’s, Nalin Munshi’s slides
Data Integration

Goal:
• Provide a uniform query interface to many heterogeneous data sources over the internet
• Free the user from having to find the data sources relevant to a query
• Easy to add and delete sources
• Find the maximal set of answers available from the sources
  • Each source only has some of the tuples we want

Paper: Information Manifold (IM) provides uniform access to over 100 heterogeneous sources
Example – Data Integration

Beach 🌊 Good weather ☀️ Cheap flight 🧱

Lonely Planet
Weather.ca
AccuWeather
Booking.com
Expedia
Travelocity

Taken from Jeffrey Niu's, Nalin Munshi's slides
Example – Data Integration

Beach 🏖️ Good weather 🌞 Cheap flight ✈️

Lonely Planet → Booking.com
Weather.ca → Expedia
AccuWeather → Travelocity
Example – Data Integration

- Beach
- Good weather
- Cheap flight

- Lonely Planet
- Weather.ca
- AccuWeather
- Booking.com
- Expedia
- Travelocity
Discussion (2 people)

Given the example of Data Integration and its use of incomplete sources.

• What are some other modern domains that require the use of incomplete sources?

• Can you think of any real-world scenarios (perhaps in your own research) where assuming complete data from a single source could be advantageous compared to the assumption of incomplete data?
Data Integration

Challenges:

Incomplete data
• View is sound but not complete
• Assume data is correct, but not all data relevant to view is in the view

Different local schemas all over the web
• Need to identify how to bind variables to create execution plans
Data Integration faces the challenges of different local schemas over the web.

• How important is standardization (data formats, APIs, etc.) in the evolution of data integration?

• Can you identify areas where lack of standardization is still a significant barrier?
Maximally-Contained Rewriting

Given query $Q$, views $V = \{V_1, \ldots, V_m\}$, a query language $\mathcal{L}$, a query $Q'$ is a maximally-contained rewriting of $Q$ using $V$ with respect to $\mathcal{L}$ if [1]:

- $Q'$ is a query in $\mathcal{L}$ that refers only to views in $V$
- $Q' \cup V$ is contained in $Q$
- $Q'$ is the query that obtains the most answers from $V$

Finding maximally-contained plan computes all certain answers [2]

Maximally-Contained Rewriting

Query:
Dest(code) :- Airport(code, city), Feature(city, "Beach")

Views:
Expedia-Air(code, city) :- Airport(code, city)
LonelyPlanet(city, POI) :- Feature(city, POI)

Rewriting:
Dest(code) :- Expedia-Air(code, city),
             LonelyPlanet(city, "Beach")

At best maximally contained. Rewriting gives as many answers as it can to query given the views.
Bucket Algorithm - CreateBucket

Given a query $Q(\bar{X}) \leftarrow R_1(\bar{X}_1), \ldots, R_m(\bar{X}_m), C_Q$:

CreateBucket:
create an empty bucket for every subgoal $R_i$
for every subgoal $R_i$:
  for each view $V(\bar{Y}) \subseteq S_1(\bar{Y}_1), \ldots, S_n(\bar{Y}_n), C_V$
    for every class $S_j$:
      if $R_i$ and $S_j$ are non-disjoint:
        map variables between $R_i$ and $S_j$
        if union of $Q$ and the mappings is satisfiable:
          add mapped $V$ to bucket $i$
do containment check over cartesian product of buckets
Bucket Algorithm - CreateBucket

Given a query $Q(\bar{X}) \leftarrow R_1(\bar{X}_1), \ldots, R_m(\bar{X}_m), C_Q$:

**CreateBucket:**
create an empty bucket for every subgoal $R_i$
for every subgoal $R_i$:
  for each view $V(\bar{Y}) \subseteq S_1(\bar{Y}_1), \ldots, S_n(\bar{Y}_n), C_V$
    for every class $S_j$:
      if $R_i$ and $S_j$ are non-disjoint:
        map variables between $R_i$ and $S_j$
        if union of $Q$ and the mappings is satisfiable:
          add mapped $V$ to bucket $i$
do containment check over cartesian product of buckets

Not in paper
**View definitions:**

V1(student, number, year) :- Registered(student, course, year),
Course(course, number), number ≥ 500, year ≥ 1992

V2(student, dept, course) :- Registered(student, course, year),
Enrolled(student, dept)

V3(student, course) :- Registered(student, course, year),
year ≤ 1990

**Query:**

q(S,D) :- Enrolled(S,D), Registered(S,C,Y), Course(C,N),
N ≥ 300, Y ≥ 1995.
## Bucket Algorithm - CreateBucket

<table>
<thead>
<tr>
<th>V1(student,number,year)</th>
<th>Registered(student,course,year),Course(course,number),number≥500,year≥1992</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2(student,dept,course)</td>
<td>Registered(student,course,year),\textit{Enrolled}(student,dept)</td>
</tr>
<tr>
<td>V3(student,course)</td>
<td>Registered(student,course,year),year ≤ 1990</td>
</tr>
<tr>
<td>q(S,D)</td>
<td>\textit{Enrolled}(S,D),Registered(S,C,Y),Course(C,N),N≥300,Y≥1995</td>
</tr>
</tbody>
</table>

- V2 has the Enrolled subgoal
- $S \rightarrow$ student
- $D \rightarrow$ dept

### Buckets

<table>
<thead>
<tr>
<th>Enrolled(S,D)</th>
<th>Registered(S,C,Y)</th>
<th>Course(C,N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2(S,D,C')</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Bucket Algorithm - CreateBucket

<table>
<thead>
<tr>
<th>V1(student,number,year)</th>
<th>Registered(student,course,year),Course(course,number),number ≥ 500,year ≥ 1992</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2(student,dept,course)</td>
<td>Registered(student,course,year),Enrolled(student,dept)</td>
</tr>
<tr>
<td>V3(student,course)</td>
<td>Registered(student,course,year),year ≤ 1990</td>
</tr>
<tr>
<td>q(S,D)</td>
<td>Enrolled(S,D),Registered(S,C,Y),Course(C,N),N ≥ 300,Y ≥ 1995</td>
</tr>
</tbody>
</table>

V1,V2,V3 have Registered
S → student
C → course
Y → year

V3 predicate does not match in year so it is not included

<table>
<thead>
<tr>
<th>Enrolled(S,D)</th>
<th>Registered(S,C,Y)</th>
<th>Course(C,N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2(S,D,C′)</td>
<td>V1(S,N′,Y)</td>
<td>V2(S,D′,C)</td>
</tr>
</tbody>
</table>
## Bucket Algorithm - CreateBucket

<table>
<thead>
<tr>
<th>V1(student,number,year)</th>
<th>Registered(student,course,year), Course(course,number), number≥500, year≥1992</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2(student,dept,course)</td>
<td>Registered(student,course,year), Enrolled(student,dept)</td>
</tr>
<tr>
<td>V3(student,course)</td>
<td>Registered(student,course,year), year ≤ 1990</td>
</tr>
<tr>
<td>q(S,D)</td>
<td>Enrolled(S,D), Registered(S,C,Y), Course(C,N), N≥300, Y≥1995</td>
</tr>
</tbody>
</table>

**V1 has Course**  
C → course  
N → number

<table>
<thead>
<tr>
<th>Enrolled(S,D)</th>
<th>Registered(S,C,Y)</th>
<th>Course(C,N)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2(S,D,C’)</td>
<td>V1(S,N’,Y)</td>
<td>V1(S’,N,Y’)</td>
</tr>
<tr>
<td>V2(S,D’,C)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bucket Algorithm - CreateExecutablePlan

- Checks if the plan is executable
- Sets bindings in execution plan
- Adds appropriate inputs to subsequent subgoals
- Removes unnecessary outputs
Time Complexity

Creating executable plan is polynomial in size of $Q'$
- NP-complete if more than 1 capability record possible per source

NP-complete overall
- Cartesian product over buckets for CreateExecutablePlan

Becomes much worse once we start allowing more predicates
The Information Manifold works with non-equivalent (contained) rewritings.

- What are other scenarios that you can imagine where you would want to use contained rewritings (maximally contained or otherwise)?

Example:

- Language translations $\rightarrow$ some words or combinations have different meanings.
- Approximate query $\rightarrow$ maybe no need to write all information
Impact of the Information Manifold

Information Manifold was highly influential:

• Became known as the Local-as-View approach
  • Easy to accommodate new sources
  • More precise source descriptions
• Sparked research into:
  • Describing sources – expressive power, tractability, binding patterns restrictions, etc.
  • Certain answers – model incomplete information
• Answering queries using views got more attention
Research Directions

• Generating schema mappings automatically
  • Using ML to make mappings
• Reference reconciliation
• Adaptive query processing
• Extension towards XML
• Model management
  • Algebra for manipulating schemas and mappings
• Peer-to-peer
Data Integration Industry

• Enterprise Information Integration (EII)
  • Provide tools for integrating from many sources without central warehouse

• High demand for data integration
  • Research matured
  • More data sharing (XML)
Current/Future Challenges

• Scaling/performance
• Getting people to share data
• Data uncertainty, inconsistency, lineage
• Leveraging human attention
The "Data Integration: The Teenage Years" was written in response to the original "Querying Heterogeneous Information Sources using Source Descriptions" paper winning the VLDB 10-years Best Paper Award.

- What are some important things to consider when reading a test-of-time award papers?
- What kinds of things would you want the authors to do?
- What kinds of things can you hope to get out of them?
Two applications/formalisms of answering queries using views:

1. **Query Optimization**
   - How to build a query plan using views
   - Find equivalent rewriting

2. **Data Integration**
   - How to integrate multiple data sources into one uniform interface
   - Find maximally-contained rewriting (incomplete sources)
   - Data integration saw significant growth in industry and research