Answering Queries Using Views

Paper by Alon Halevy
Presentation by Oana Sandu
Discussion by Doug Bateman

What is a view?

- A stored query that can be queried like a relation
  - Materialized view: result set is stored
  - Virtual view: not stored
- SQL view:

```
CREATE VIEW Animation_Heros AS
    SELECT Main.Name
    FROM Main, Genre
    WHERE Main.Hero = Genre.movie
          AND Genre.Type = "Animation"
```

- Datalog view:

```
movie-hero(hero) :- Main(hero, movie),
                    Genre(movie, "Animation")
```


Answering queries using views

- Answer a query in terms of views rather than using the underlying base table
- Query:

```
q(hero) :- Main(hero, movie),
           Genre(movie, "Animation")
```

- Materialized view:

```
hero-type(hero, type) :- Main(hero, movie),
    Genre(movie, type)
```

- Rewriting using view:

```
q(hero):-hero-type(hero, "Animation")
```


Data Management Problems

- We will discuss AQUV in 2 contexts:
- 1. Query Optimization
  - Speed up a query
- 2. Data Integration
  - Query over many local databases

Query Optimization

- Rewrite query in terms of views and base tables
  - Query:

```
q(hero, tophero):- Main(hero, movie),
                   Genre(movie, type),
                   TopCharacter(type, tophero)
```

  - View:

```
hero-type(hero, type) :- Main(hero, movie),
                         Genre(movie, type)
```

  - Rewriting:

```
q(hero, tophero):- hero-type(hero, type),
                   TopCharacter(type, tophero)
```


Query Optimization: Closed World

- Assumption: views are complete
- We want an equivalent query rewriting: retrieve *exactly* the same answers as the original query

Query: q(hero) :- Main(hero, movie), Genre(movie, "Action")

Views:

```
hero-type(hero, type) :- Main(hero, movie), Genre(movie, type)
dreamworks-hero(hero,type) :- Main(hero, movie),
                              Genre(movie,type), Producer(movie, "Dreamworks")
```

Equivalent rewriting:

```
q(hero):-hero-type(hero, "Action")
```

Nonequivalent rewriting:

```
q(hero):- dreamworks-hero(hero,"Action")
```

Query Optimization: Incorporating Views

- Fold views into System-R style optimizer
- Views used for alternate access paths
- Determine which views are usable in answering query q
- Compare evaluation plans
  - Some of q might be precomputed in views
  - Sometimes cheaper to use base tables

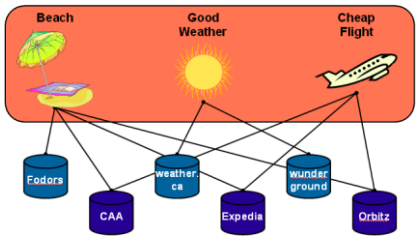
Discussion: Question 1

- What sorts of traditional database scenarios can you imagine using many materialized views? (Think about YOUR data.)

Discussion: Question 2

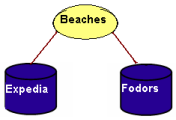
- Imagine that you're building a query optimizer. Would you consider it worth your while to use views when answering queries? Why or why not? Would you try it only for certain kinds of queries? Which ones? How does this tradeoff compare with using bushy trees?

Data Integration:



Data Integration: Open World

- Local sources:
  - maintained autonomously, can be incomplete
    - Expedia: info on Carribean beaches
    - Fodor's: info on US beaches
  - local sources together need not cover all tuples in the conceptual Beaches



Data Integration: Rewriting

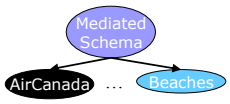
- Query written in terms of a mediated schema (not materialized!)
- So need an algorithm to rewrite the query in terms of the local schemas
- Assumption: Open World
- Success Criteria: answer includes as many correct tuples as can be determined from local sources

Data Integration: Rewriting

- How do we rewrite the query?
- Need some mapping between mediated schema and local sources
- Local As View approach:
  - Local sources expressed as views over mediated schema
  - Executing queries over mediated schema reduces to AQUV

Data Integration: Local As View

- Mediated Schema:
  Airport(code, city)
  Feature(city, attraction)



- Local Sources as Views:

  AirCanada(code, city) :- Airport(code, city)
  Beaches(code) :- Airport(code, city),
  Feature(city, "Beach")

- Can easily add new sources

AQUV in Data Integration: Maximally Contained Rewritings

- Query:
  Dest(code) :- Airport(code, city), Feature(city, "Beach")
- Sources/Views:
  CAA-Air(code, city) :- Airport(code, city)
  Fodors(city, POI) :- Feature(city, POI)
- Rewriting:
  Dest(code):-CAA-Air(code, city), Fodors(city, "Beach")
- Contained Rewriting: all answers obtained are valid answers to query
- Maximally Contained: get all possible answers given the local sources
- Finding rewriting is NP-complete

Discussion: question 3

- What cases can you imagine using data integration? (Think about YOUR data.)

Data Integration: Rewriting Algorithms

- 1. Bucket Algorithm
  - breaks down query answering into answering subgoals using views
- 2. MiniCon
  - considers subgoal interactions to reduce search space

Data Integration: Bucket Algorithm

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

- Step 1:
  - Create a bucket for each query subgoal
  - For each bucket, place all relevant views
- Step 2:
  - Checks all rewritings: cross-product of buckets (A,B,C)X(C,D) = (A,C),(A,D),(B,C),(B,D),(C,C),(C,D)
  - Containment checking is NP-hard!

Data Integration: Bucket Algorithm

- Ignores subgoal interactions in step 1
- Query: Dest(code) :- Airport(code, city), Feature(city, "Beach")
- Sources/Views:
  Orbitz(code) :- Airport(code, city)
  Beaches(code) :- Airport(code, city), Feature(city, "Beach")
  Frommers(city, POI):- Feature(city,POI)
- Only rejects Orbitz combos at very end:
  Dest'(code):-Orbitz(code),Frommers(city,"Beach")

The MiniCon Algorithm**

- Pruning earlier than bucket algorithm
- Creates MiniCon descriptions (MDCs) considering subgoal interactions when including views
- Combining MDCs: many fewer combos

**Rachel Pottinger and Alon Halevy

Discussion question 4

- Can you envision using web sites as "Local Sources" for data integration? What types of restrictions on queries do you think these sources might impose? How would your query rewriting algorithm need to be modified to account for these restrictions?

Data Integration: Example Revisited with MiniCon

- Query:
  Dest(code) :- Airport(code, city), Feature(city, "Beach")
- Sources/Views:
  Orbitz(code) :- Airport(code, city)
  Beaches(code) :- Airport(code, city),Feature(city, "Beach")
- Rewriting:
  Dest(code) :- Beaches(code)
- MCDs: View subgoals linked by existential variables must be mapped together