# Manipulating Query Expressions

Hala Skaf-Molli
Nantes University
Hala.Skaf@univ-nantes.fr

**This lecture is based on the book Principles of Data Integration (Chapter 2)**

# Introduction

- How does a data integration system decide which sources are relevant to a query? Which are redundant? How to combine multiple sources to answer a query?
- Answer: by reasoning about the contents of data sources.
  - Data sources are often described by queries / views.
- This chapter describes the fundamental tools for manipulating query expressions and reasoning about them.

# Outline

- Review of basic database concepts
- Query unfolding
- Query containment
- Answering queries using views

# Basic Database Concepts

- Relational data model
- Integrity constraints
- Queries and answers
- Conjunctive queries
- Datalog

# Relational Terminology

Relational schemas

- Tables, attributes

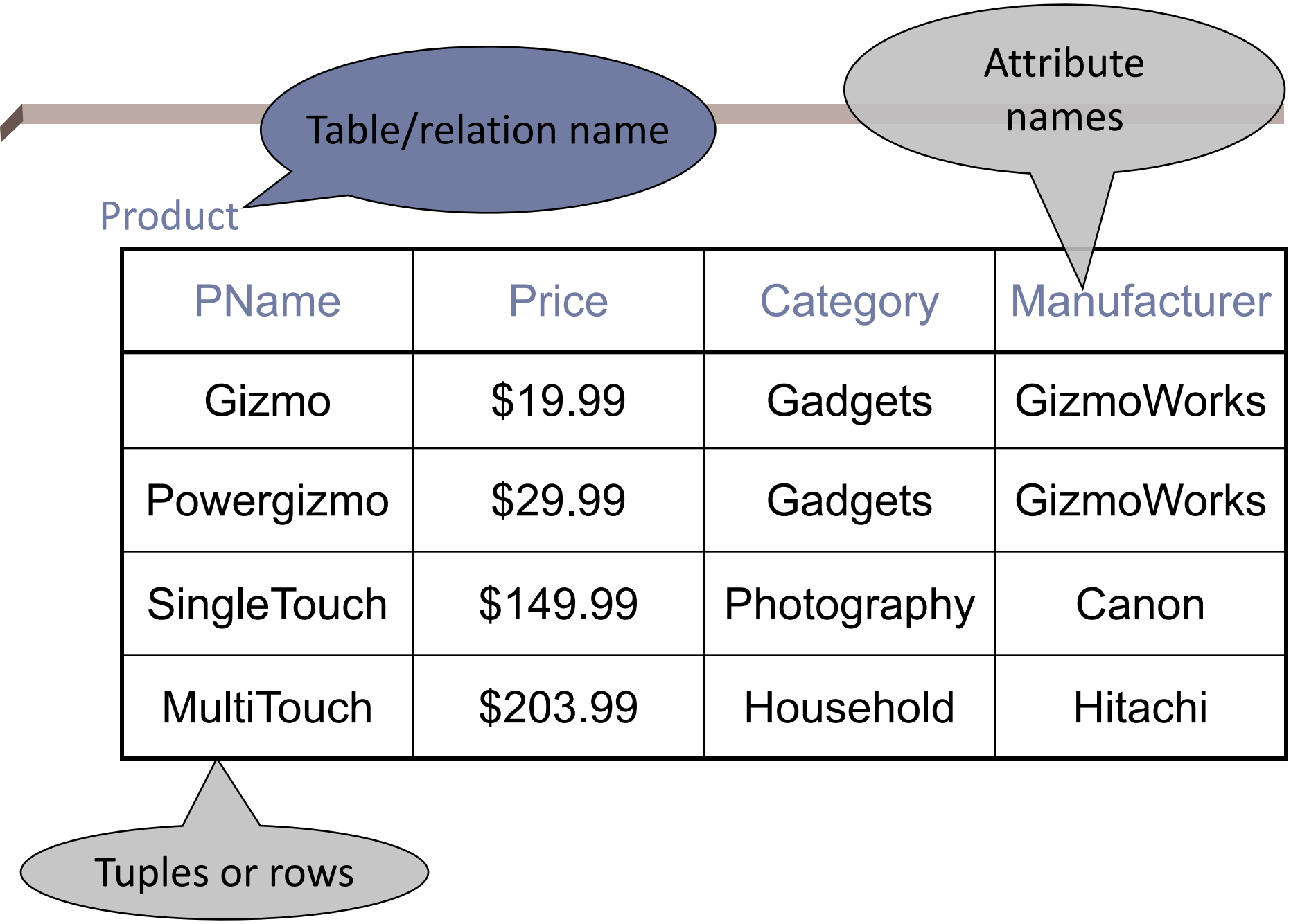
Relation instances

- Sets (or multi-sets) of tuples

Integrity constraints

- Keys, foreign keys, inclusion dependencies

- A state of the database (database instance) (D): is a snapshot of the contents of the database.


Table/relation name
Attribute names
Product
PName
Price
Category
Manufacturer
Gizmo
$19.99
Gadgets
GizmoWorks
Powergizmo
$29.99
Gadgets
GizmoWorks
SingleTouch
$149.99
Photography
Canon
MultiTouch
$203.99
Household
Hitachi
Tuples or rows

# SQL (very basic)

**Employee**(empID, name,hireDate,manger)

**Interview**(candidate, date, recruiter, hireDecision, grade)

**EmployeePerf**(empID, name, reviewQuarter, grade, reviewer)

```
select recruiter, candidate
from Interview, EmployeePerf
where recruiter=name AND
        grade < 2.5
```

# Query Answers

- *Q(D):* the set (or multi-set) of rows resulting from applying the query *Q* on the database *D*.

- Unless otherwise stated, we will consider sets rather than multi-sets.

# SQL (w/aggregation)

**EmployeePerf**(empID, name, reviewQuarter, grade, reviewer)

```
select reviewer, Avg(grade)
from EmployeePerf
where reviewQuarter="1/2007"
```

# Integrity Constraints (Keys)

- A key is a set of columns that uniquely determine a row in the database:
  - There do not exist two tuples, $t_1$ and $t_2$ such that $t_1 \neq t_2$ and $t_1$ and $t_2$ have the same values for the key columns.
  - (EmpID, reviewQuarter) is a key for **EmployeePerf**

# Integrity Constraints (Functional Dependencies)

- A set of attribute **A** functionally determines a set of attributes **B** if: whenever , $t_1$ and $t_2$ agree on the values of **A** , they must also agree on the values of **B**.
- For example, (EmpID, reviewQuarter) functionally determine (grade).
- Note: a key dependency is a functional dependency where the key determines all the other columns.

# Integrity Constraints (Foreign Keys)

- Given table **T** with key *B* and table **S** with key *A*: *A* is a foreign key of *B* in **T** if whenever a **S** has a row where the value of *A* is **v**, then **T** must have a row where the value of *B* is **v**.
- Example: the **emplD** attribute of **EmployeePerf** is a foreign key for attribute **emp** of **Employee**.

# General Integrity Constraints

Tuple generating dependencies (TGD's)

$$(\forall \bar{X}) s_1(\bar{X}_1), ..., s_m(\bar{X}_m) \rightarrow (\exists \bar{Y})\, t_1(\bar{Y}_1), ..., t_1(\bar{Y}_1)$$

Equality generating dependencies (EGD's): right hand side contains only equalities.

$$(\forall \bar{X}) s_1(\bar{X}_1), ..., s_m(\bar{X}_m) \rightarrow X_1^1 = X_2^1, ..., X_1^k = X_2^k$$

Exercise: express the previous constraints using general integrity constraints.

# Conjunctive Queries

Most common form of query; equivalent to select-project-join queries

$$Q_1(\overline{X}) :- g_1(\overline{X_1}), \ldots, g_n(\overline{X_n})$$

Q: head, $g_1$.. $g_n$: body

Safe ..

Semantics (Evaluating CQ's)

if $\varphi$ maps the body subgoals to tuples in *D*

*then,* $\varphi(\overline{X})$ is an answer.

**Interview**(candidate, date, recruiter, hireDecision, grade)

**EmployeePerf**(empID, name, reviewQuarter, grade, reviewer)

# Conjunctive Queries

**Q(X,T) :-** Interview(X,D,Y,H,F), EmployeePerf(E,Y,T,W,Z)

Joins are expressed with multiple occurrences of the same variable

```
select candidate, recruiter
from Interview, EmployeePerf
where recruiter=name
```

# CQ Examples

- Database schema: parent(parent,child)
- CQ'S:
  - Parent-of-bart(X) :- parent(X, ``Bart'')
  - Equivalent to relational algebra query:
  - 
  - Grandparent(X,Y):- parent(X,Z),parent(Z,Y)
  - Joins are expressed with multiple occurrences of the same variable
- Unsafe query:
  - Unsafe-query(X,Y):-parent(X,Z)
    - Where would we get the value of Y in the query result ?

# Evaluating CO'S

- Substitute constants for variables in the body of Q such that all subgoals becomes true
  - Result contains the head under the same substitution
- Example:
  - grandparent(X,Y):- parent(X,Z), parent(Z,Y)
  - Database instance: parent(”Abe”,”Homer”),parent(“Homer”,”Bart”),parent(“Homer”,”Lisa”)
  - Only substitutions that make both subgoals true
    - X->
    - X->
  - These substitutions yields grandparent(“Abe”,”bart”) and grandparent(“Abe”,”Lisa”), which are the result tuples

# Conjunctive Queries (interpreted predicates)

**Q(X,T) :-**

Interview(X,D,Y,H,F), EmployeePerf(E,Y,T,W,Z),
**W < 2.5**.

Interpreted (or comparison) predicates.
Variables must also appear in regular atoms.

```
select recruiter, candidate
from Interview, EmployeePerf
where recruiter=name AND
        grade < 2.5
```

# Conjunctive Queries (negated subgoals)

**Q(X,T) :-**

Interview(X,D,Y,H,F), EmployeePerf(E,Y,T,W,Z),

¬OfferMade(X, date).

Safety: every head variable must appear in a positive subgoal.

# Unions of Conjunctive Queries

Multiple rules with the same head predicate express a union

**Q(X,T) :-**

Interview(X,D,Y,H,F), EmployeePerf(E,Y,T,W,Z),
W < 2.5.

**Q(X,T) :-**

Interview(X,D,Y,H,F), EmployeePerf(E,Y,T,W,Z),
Manager(y), W > 3.9.

# Datalog (recursion)

Database: edge(X,Y) – describing edges in a graph.
Recursive query finds all paths in the graph.

$r_1$ **path(X,Y) :-** edge(X,Y)

$r_2$ **path(X,Y) :-** edge(X,Z), path(Z,Y)

# Outline

- ✓ Review of basic database concepts
- ➢ Query unfolding
- ▪ Query containment
- ▪ Answering queries using views

# Query Unfolding

- Query **composition** is an important mechanism for writing complex queries.
  - Build query from views in a bottom up fashion.
- Query unfolding "unwinds" query composition.
- Important for:
  - Comparing between queries expressed with views
  - Query optimization (to examine all possible join orders)

# Query Unfolding Example

Relations: *Flight(source, destination)*
*Hub(city)*

$$Q_1(X,Y) :- Flight(X,Z), Hub(Z), Flight(Z,Y)$$
$$Q_2(X,Y) :- Hub(Z), Flight(Z,X), Flight(X,Y)$$
$$Q_3(X,Z) :- Q_1(X,Y), Q_2(Y,Z)$$

The unfolding of $Q_3$ is:

$$Q'_3(X,Z) :- Flight(X,U), Hub(U), Flight(U,Y),$$
$$Hub(W), Flight(W,Y), Flight(X,Z)$$

# Query Unfolding Algorithm

- Find a subgoal $p(X_1,\ldots,X_n)$ such that $p$ is defined by a rule $r$.
- Unify $p(X_1,\ldots,X_n)$ with the head of $r$.
- Replace $p(X_1,\ldots,X_n)$ with the result of applying the unifier to the subgoals of $r$ (use fresh variables for the existential variables of $r$).
- Iterate until no unifications can be found.
- If p is defined by a union of $r_1, \ldots, r_n$, create $n$ rules, for each of the $r$'s.

# Exercise

- Q1(X,Y):- S(X,Z),S(Y,Z)
- Q2(X):-S(X,Y)

- Q3(X):-Q1(X,Y),Q2(Y)

- Unfolding ??

# Query Unfolding Summary

- Unfolding does not necessarily create a more efficient query!
  - Just lets the optimizer explore more evaluation strategies.
  - Unfolding is the opposite of rewriting queries using views (see later).
- Allows query process to explore a wider collection of query plans:
  - A larger possibility to order join operation ..

# Outline

- ✓ Review of basic database concepts
- ✓ Query unfolding
- ➢ Query containment
- ▪ Answering queries using views

# Query Containment

- For two queries Q1 and Q2, if all of the answers to Q2 are a subset of those of Q1 for all databases, then Q1 contains Q2.
- Denoted : $\supseteq$

# Containment: Conjunctive Queries

$$Q_1(\overline{X}) :- g_1(\overline{X_1}),\ldots,g_n(\overline{X_n})$$

No interpreted predicates ($\geq,\neq$)
or negation for now.

Recall semantics (Evaluating CQ's)

if $\varphi$ maps the body subgoals to tuples in *D*
*then,* $\varphi(\overline{X})$ is an answer.

# Query Containment and Equivalence: Definitions

Let Q1 and Q2 be two queries of the same arity. We say that Q1 contains Q2 , denoted by, Q1 $\supseteq$ Q2, if for any database D, Q1(D) conatins Q2(D)

Query $Q_1$ is equivalent to query $Q_2$ if $Q_1(D) \supseteq Q_2(D)$ and $Q_2(D \supseteq) \; Q_1(D)$

Note: containment and equivalence are properties of the queries, not of the database!

# Example of CQ containment

- $Q_1$ : p(X,Y) :- r(X,W), b(W,Z), r(Z,Y)
- $Q_2$ : p(X,Y) :- r(X,W), b(W,W) r(W,Y)
- Claim: $Q_1$ contains $Q_2$ (Q1 $\supseteq$ Q2)

- Proof:
  - If p(x,y) is in $Q_2$, there is some w such that r(x,w), b(w,w) and r(w,y) true
  - For Q1, make the substitution X->x; Y->y; W->w; Z->w
  - All subgoals of $Q_1$ are true, and the head of Q1 becomes p(x,y)
  - Thus, p(x,y) is also in $Q_1$, proving that Q1 contains Q2.

# Containment Mappings

$$Q_1(\overline{X}) :- g_1(\overline{X_1}),\ldots,g_n(\overline{X_n})$$

$$Q_2(\overline{Y}) :- h_1(\overline{Y_1}),\ldots,h_m(\overline{Y_m})$$

$\varphi$: *Vars($Q_1$) $\rightarrow$ Vars($Q_2$) is a containment mapping if :*

$$\varphi(\overline{X}) = \overline{Y}$$

*and*

$$\varphi(g_i(\overline{X_i})) \in Body(Q_2)$$

# Example 1

$$Q_1(X,Z) :- p(X,Y,Z)$$

$$Q_2(X,Z) :- p(X,X,Z)$$

$$Q_1 \supseteq Q_2$$

# Example 2

$$Q_1(X,Y) :- p(X,Z), p(Z,Y)$$

$$Q_2(X,Y) :- p(X,Z), p(Z,Y), p(X,W)$$

$$Q_1 \supseteq Q_2$$

# Example 3

$Q_2$ contains $Q_1$ ?

- $Q_1(X,Y)$ :- $p(X,W,Z)$, $r(Y,W,Z)$
- $Q_2(Y_1,Y_2)$ :- $p(Y_1,Y_3,Y_4)$, $r(Y_2,Y_3,Y_5)$

# Theorem
## [Chandra and Merlin, 1977]

$Q_1$ contains $Q_2$ if and only if there is a containment mapping from $Q_1$ to $Q_2$

Deciding whether $Q_1$ contains $Q_2$
is NP-complete in the size of the two queries

# Outline

- ✓ Review of basic database concepts
- ✓ Query unfolding
- ✓ Query containment
- ➢ Answering queries using views

# Why Do We Need It?

- When sources are described as views, we use containment to compare among them.
- If we can remove sub-goals from a query, we can evaluate it more efficiently.
- Actually, containment arises everywhere…

# Example

Relations: *Flight(source, destination)*
*Hub(city)*

Views:

S1: $Q_1(X,Y)$ :- *Flight(X,Z), Hub(Z), Flight(Z,Y)*
*S2:* $Q_2(X,Y)$ :- *Hub(Z), Flight(Z,X), Flight(X,Y)*

Query:

$Q_3(X,Z)$ :- $Q_1(X,Y)$, $Q_2(Y,Z)$

Unfolding:

# Remove Redundant Subgoals

Redundant subgoals?

$Q'_3(X,Z)$ :- *Flight(X,U), Hub(U), Flight(U,Y), Hub(W), Flight(W,Y), Flight(Y,Z)*

$\Rightarrow$

$Q''_3(X,Z)$ :- *Flight(X,U), Hub(U), Flight(U,Y), Flight(Y,Z)*

Is $Q''_3$ equivalent to $Q'_3$?

# Problem Definition

Input: Query $Q$

View definitions: $V_1,\ldots,V_n$

A rewriting: a query $Q'$ that refers only to the views and interpreted predicates

An equivalent rewriting of Q using $V_1,\ldots,V_n$:
a rewriting Q', such that $Q' \Leftrightarrow Q$.

# Motivating Example (Part 1)

Movie(ID,title,year,genre)
Director(ID,director)
Actor(ID, actor)

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_1(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1940, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_1 \supseteq Q \quad \Rightarrow \quad Q'(T,Y,D) :- V_1(T,Y,D), Y \geq 1950$$

Containment is enough to show that $V_1$ can be used to answer Q.

# Motivating Example (Part 2)

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_2(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

$$V_3(I,D) :- Director(I,D), Actor(ID,D)$$

Containment does not hold, but intuitively, $V_2$ and $V_3$ are useful for answering $Q$.

$$Q''(T,Y,D) :- V_2(I,T,Y), V_3(I,D)$$

How do we express that intuition?

*Answering queries using views!*

# The problem

- **Query writing**: Given a query Q and a set of view definition V1, ….  Vn, a rewriting of the query using the views is a query expression Q' that refers only to view V1,… Vn.
- **Equivalence Query Rewriting**: Let Q a  query and V={V1… Vn} be a set of view definitions. The query Q' is equivalent rewriting of Q using V if:
  - Q' refers only to the views in V, and
  - Q' is equivalent to Q
- Check equivalence between a query Q and a rewriting, consider the unfolding of Q' w.r.t the views

# Motivating Example (Part 3)

Movie(ID,title,year,genre)
Director(ID,director)
Actor(ID, actor)

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_4(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1960, G = "comedy"$$

$$V_3(I,D) :- Director(I,D), Actor(ID,D)$$

$$Q'''(T,Y,D) :- V_4(I,T,Y), V_3(I,D)$$

*maximally-contained rewriting*

# Maximally-Contained Rewritings

Input: Query *Q*
*Rewriting query language* ***L***
View definitions: $V_1, \ldots, V_n$

Q' is a maximally-contained rewriting of Q using $V_1, \ldots, V_n$ *and* ***L*** if:

1. $Q' \in \boldsymbol{L}$,
2. *Q' is contained in Q* ($Q' \subseteq Q$), *and*
3. *there is no Q" in* ***L*** *such that* $Q'' \subseteq Q$ *and* $Q' \subset Q''$

# Exercise: which of these views can be used to answer Q?

$$Q(T,Y,D) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$
$$Director(I,D), Actor(I,D)$$

$$V_2(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

$$V_3(I,D) :- Director(I,D), Actor(I,D)$$

$$V_6(T,Y) :- Movie(I,T,Y,G), Y \geq 1950, G = "comedy"$$

$$V_7(I,T,Y) :- Movie(I,T,Y,G), Y \geq 1950,$$
$$G = "comedy", Award(I,W)$$

$$V_8(I,T) :- Movie(I,T,Y,G), Y \geq 1940, G = "comedy"$$

# Algorithms for answering queries using views

- Step 1: we'll bound the space of possible query rewritings we need to consider (no interpreted predicates)
- Step 2: we'll find efficient methods for searching the space of rewritings
  - Bucket Algorithm, MiniCon Algorithm
- Step 2b: we consider "logical approaches" to the problem:
  - The Inverse-Rules Algorithm
- We'll consider interpreted predicates, …

# Complexity Result [LMSS, 1995]

- Applies to queries with no interpreted predicates.
- Finding an equivalent rewriting of a query using views is NP-complete
  - Need only consider rewritings of query length or less.
- Maximally-contained rewriting:
  - Union of all conjunctive rewritings of length *n* or less.