Outline

0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance
4. Virtual Data Integration

• Virtual Data Integration

![Diagram of Virtual Data Integration]

- Query
- Global Schema
- Mappings
- Local Schema 1
- Local Schema 2
- Local Schema n
4. Virtual Data Integration

Problems:

• How to create mappings?
  – Discussed in previous part of the course

• How to compute query Q
  – This is the main focus of this part
4. Query Answering with Views

• How to compute query Q over global schema based on source schemas only?
  – What language is used to express mappings?
  – What language due we allow for Q?
  – What language(s) can we use to query local sources?
  – What language can we use to compute Q from query results returned by local sources?
  – How to deal with incompleteness?
4.1 Query Answering with Views

Example: Solutions

Local Schema
- **Person**
  - Name
  - Address

Global Schema
- **Person**
  - Name
  - Address
  - Office-phone
  - Office-address
  - Home-phone

\[ \forall x, y, z, a : Person(x, y) \land Address(y, z, a) \rightarrow \exists b, c : Person(x, z, a, b, c) \]

Query:

\[ Q(\text{Name}) := Person(\text{Name}, A, OP, OA, HP). \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
</tr>
<tr>
<td>Alice</td>
<td>2</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>City</th>
<th>Office-contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chicago</td>
<td>(312) 123 4343</td>
</tr>
<tr>
<td>2</td>
<td>Chicago</td>
<td>(312) 555 7777</td>
</tr>
<tr>
<td>3</td>
<td>New York</td>
<td>(465) 123 1234</td>
</tr>
</tbody>
</table>
4.1 Query Answering with Views

**Example: Solutions**

<table>
<thead>
<tr>
<th>Local Schema</th>
<th>Global Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Id</strong></td>
</tr>
<tr>
<td>Peter</td>
<td>1</td>
</tr>
<tr>
<td>Alice</td>
<td>2</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ \forall x, y, z, a : Person(x, y) \land Address(y, z, a) \rightarrow \exists b, c : Person(x, z, a, b, c) \]

Query: \[ Q(\text{Name}) :- Person(\text{Name}, A, OP, OA, HP). \]

Rewritten query over the source:

\[ Q(\text{Name}) :- Person(\text{Name}, AI), Address(AI, A, OP). \]
4.1 Query Answering with Views

Example: Solutions

Local Schema

```
Person
  Name
  Address
```

Global Schema

```
Person
  Name
  Address
  Office-phone
  Office-address
  Home-phone
```

Values of home-phone are not available in the source

\[ \forall x, y, z, a : Person(x, y) \land Address(y, z, a) \rightarrow \exists b, c : Person(x, z, a, b, c) \]

Query: \[ Q(\text{Home-ph}) :\text{-} Person(N, A, \text{OP}, OA, \text{Home-ph}). \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peter</td>
<td>1</td>
</tr>
<tr>
<td>Alice</td>
<td>2</td>
</tr>
<tr>
<td>Bob</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>City</th>
<th>Office-contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chicago</td>
<td>(312) 123 4343</td>
</tr>
<tr>
<td>2</td>
<td>Chicago</td>
<td>(312) 555 7777</td>
</tr>
<tr>
<td>3</td>
<td>New York</td>
<td>(465) 123 1234</td>
</tr>
</tbody>
</table>
4. Query Answering with Views

• Problems
  – How to determine whether query can be answered at all?
  – Given a rewriting of the query using views, how do we know it is correct?
  – What to do if views can only return some of the query results?
Motivating Example (Part 1)

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

\[ Q(T,Y,D) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = "\text{comedy}\"
\]

\[ \text{Director}(I,D), \text{Actor}(I,D) \]

\[ V_1(T,Y,D) : \neg \text{Movie}(I,T,Y,G), Y \geq 1940, G = "\text{comedy}\"
\]

\[ \text{Director}(I,D), \text{Actor}(I,D) \]

\[ V_1 \supseteq Q \quad \Rightarrow \quad Q'(T,Y,D) : \neg V_1(T,Y,D), Y \geq 1950 \]

Containment is enough to show that \( V_1 \) can be used to answer \( Q \).
Motivating Example (Part 2)

\[ Q(T,Y,D) : \neg Movie(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]

\[ Director(I,D), Actor(I,D) \]

\[ V_2(I,T,Y) : \neg Movie(I,T,Y,G), Y \geq 1950, G = \text{"comedy"} \]

\[ V_3(I,D) : \neg Director(I,D), Actor(ID,D) \]

**Containment** does not hold, but intuitively, \( V_2 \) and \( V_3 \) are useful for answering \( Q \).

\[ Q'(T,Y,D) : \neg V_2(I,T,Y), V_3(I,D) \]

How do we express that intuition?

*Answering queries using views!*
Input: Query $Q$
View definitions: $V_1, \ldots, V_n$

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

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

- **Given Q and views**
  - Randomly combine views into a query Q’
  - Check equivalence of Q’ and Q
  - If Q’ is equivalent we are done
  - Else repeat

- **Why is this not good?**
  - There are infinitely many ways of combining views
    - E.g., V, V x V, V x V x V, …
  - We are not using any information in the query
Motivating Example (Part 3)

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

\[ Q(T,Y,D) : \neg \text{Movie}(I,T,Y,G), Y \geq 1950, G = "comedy" \]
\[ \text{Director}(I,D), \text{Actor}(I,D) \]

\[ V_4(I,T,Y) : \neg \text{Movie}(I,T,Y,G), Y \geq 1960, G = "comedy" \]

\[ V_3(I,D) : \neg \text{Director}(I,D), \text{Actor}(ID,D) \]

\[ Q''(T,Y,D) : \neg V_4(I,T,Y), V_3(I,D) \]

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 $Q$ given $V_1, \ldots, V_n$ and $L$ if:

1. $Q' \in L$,
2. $Q' \subseteq Q$, and
3. there is no $Q''$ in $L$ such that $Q'' \subseteq Q$ and $Q' \subset Q''$
Why again?

Global Schema

Local Schema 1

Local Schema 2

Local Schema n

Query

Mappings

LAV/GLAV!
Other use-cases

• Query optimization with materialized views
  – Need equivalent rewritings
  – Implemented in many commercial DBMS
  – Here interest is cost: how to speed-up query processing by using materialized views
Exercise: which of these views can be used to answer $Q$?

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

Director(I,D), Actor(I,D)

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

$$V_3(I,D) : \neg Director(I,D), Actor(I,D)$$

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

$$V_7(I,T,Y) : \neg Movie(I,T,Y,G), Y \geq 1950,$$

$$\neg Award(I,W)$$

$$V_8(I,T) : \neg Movie(I,T,Y,G), Y \geq 1940, G = "comedy"$$
Algorithms for answering queries using views

- **Step 1**: we’ll bound the space of possible query rewritings we need to consider (no comparisons)

- **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
**Theorem**: if there is an equivalent rewriting, there is one with at most $n$ subgoals.

**Query**: \[ Q(\overline{X}) : \neg p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}) \]

**Rewriting**: \[ Q'(\overline{X}) : \neg V_1(\overline{X_1}), \ldots, V_m(\overline{X_m}) \]

**Expansion**: \[ Q''(\overline{X}) : \neg g^1_1, \ldots, g^1_k, \ldots, g^m_1, \ldots, g^m_j \]

**Proof**: Only $n$ subgoals in $Q$ can contribute to the image of the containment mapping $\varphi$. 
• 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.
The Bucket Algorithm

Key idea:

– Create a bucket for each subgoal $g$ in the query.
– The bucket contains views that contribute to $g$.
– Create rewritings from the Cartesian product of the buckets (select one view for each goal)

• **Step 1**: assign views with renamed vars to buckets

• **Step 2**: create rewritings, refine them, until equivalent/all contained rewriting(s) are found
The Bucket Algorithm

Step 1:
- We want to construct buckets with views that have partially mapped variables
- For each goal \( g = R \) in query
- For each view \( V \)
- For each goal \( v = R \) in \( V \)
  - If the goal has head variables in the same places as \( g \) then
    - rename the view head variables to match the query goal vars
    - choose a new unique name for each other var
    - add the resulting view atom to the bucket
Step 1 Intuition

– A view can only be used to provide information about a goal R(X) if it has a goal R(Y)

  • Q(X) :- R(X,Y)
  • V(X) :- S(X,Y)

– If the query goal contains variables that are in the head of the query, then the view is only useful if it gives access to these values (they are in the head)

  • Q(X) :- R(X,Y)
  • V(X) :- S(X,Y), R(Y,Z)
Bucket Algorithm in Action

\[ Q(\text{ID}, \text{Dir}) : \neg \text{Movie(} \text{ID}, \text{title, year, genre}), \text{Revenues(} \text{ID}, \text{amount}), \text{Director(} \text{ID}, \text{dir}), \text{amount} \geq 100M \]

\[ V_1(\text{I,Y}) : \neg \text{Movie(} \text{I,T,Y,G}), \text{Revenues(} \text{I,A}), I \geq 5000, A \geq 200M \]
\[ V_2(\text{I,A}) : \neg \text{Movie(} \text{I,T,Y,G}), \text{Revenues(} \text{I,A}) \]
\[ V_3(\text{I,A}) : \neg \text{Revenues(} \text{I,A}), A \leq 50M \]
\[ V_4(\text{I,D,Y}) : \neg \text{Movie(} \text{I,T,Y,G}), \text{Director(} \text{I,D}), I \leq 3000 \]

View atoms that can contribute to \textit{Movie}:
\[ V_1(\text{ID}, \text{year’}), V_2(\text{ID}, \text{A’}), V_4(\text{ID}, \text{D’}, \text{year’}) \]
## Buckets and Cartesian product

<table>
<thead>
<tr>
<th>Movie(ID,title, year,genre)</th>
<th>Revenues(ID, amount)</th>
<th>Director(ID,dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1(ID,\text{year})$</td>
<td>$V_1(ID,Y')$</td>
<td>$V_4(ID,\text{Dir},Y')$</td>
</tr>
<tr>
<td>$V_2(ID,\text{A'})$</td>
<td>$V_2(ID,\text{amount})$</td>
<td></td>
</tr>
<tr>
<td>$V_4(ID,\text{D'},\text{year})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Consider first candidate rewriting: first $V_1$ subgoal is redundant, and $V_1$ and $V_4$ are mutually exclusive.

$q_1'(ID,\text{dir}) : \neg V_1(ID,\text{year}), V_1(ID,y'), V_4(ID,\text{dir},y')$
### Next Candidate Rewriting

<table>
<thead>
<tr>
<th>Movie(ID,title,year,genre)</th>
<th>Revenues(ID,amount)</th>
<th>Director(ID,dir)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1(ID,\text{year})$</td>
<td>$V_1(ID,Y')$</td>
<td>$V_4(ID,\text{Dir},Y')$</td>
</tr>
<tr>
<td>$V_2(ID,A')$</td>
<td>$V_2(ID,\text{amount})$</td>
<td></td>
</tr>
<tr>
<td>$V_4(ID,D',\text{year})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$q_2'(ID,\text{dir}) : -V_2(ID,A') , V_2(ID,\text{amount}) , V_4(ID,\text{dir},y')$

$q_2'(ID,\text{dir}) : -V_2(ID,\text{amount}) , V_4(ID,\text{dir},y') , \text{amount} \geq 100M$
The Bucket Algorithm

Step 2:

– For each combination of one element of each bucket:
  – Create query Q’ with query Q’s head and list all these view atoms in the body
  – If Q’ equivalent to Q (or contained in Q)
    • Done (equivalent)
    • Add to union of CQs for contained case
  – If not try to add comparisons
The Bucket Algorithm: Summary

• Cuts down the number of rewriting that need to be considered, especially if views apply many interpreted predicates.

• The search space can still be large because the algorithm does not consider the interactions between different subgoals.
  
  – See next example.
The MiniCon Algorithm

\[
Q(\text{title}, \text{year}, \text{dir}) : \neg \text{Movie}(\text{ID}, \text{title}, \text{year}, \text{genre}), \\
\text{Director}(\text{ID}, \text{dir}), \text{Actor}(\text{ID}, \text{dir})
\]

\[
V_5(\text{D}, \text{A}) : \neg \text{Director}(\text{I}, \text{D}), \text{Actor}(\text{I}, \text{A})
\]

**Intuition:** The variable \( I \) is not in the head of \( V_5 \), hence \( V_5 \) cannot be used in a rewriting. **MiniCon** discards this option early on, while the Bucket algorithm does not notice the interaction.
MinCon Algorithm Steps

• 1) Create MiniCon descriptions (MCDs):
  – Homomorphism on view heads
  – Each MCD covers a set of subgoals in the query with a set of subgoals in a view

• 2) Combination step:
  – Any set of MCDs that covers the query subgoals (without overlap) is a rewriting
  – No need for an additional containment check!
MiniCon Descriptions (MCDs)
An atomic fragment of the ultimate containment mapping

\[ Q(title,act,dir) : \neg \text{Movie}(ID,title,year,genre), \]
\[ \quad \text{Director}(ID,dir), \text{Actor}(ID,act) \]

\[ V(I,D,A) : \neg \text{Director}(I,D), \text{Actor}(I,A) \]

MCD:

mapping:

\[ ID \rightarrow I \]
\[ dir \rightarrow D \]
\[ act \rightarrow A \]

covered subgoals of \( Q \): \{2,3\}
\( Q(\text{title}, \text{year}, \text{dir}) : \neg \text{Movie}(\text{ID}, \text{title}, \text{year}, \text{genre}), \text{Director}(\text{ID}, \text{dir}), \text{Actor}(\text{ID}, \text{dir}) \)

\( V(I,D,A) : \neg \text{Director}(I,D), \text{Actor}(I,A) \)

Need to specialize the view first:
\( V'(I,D,D) : \neg \text{Director}(I,D), \text{Actor}(I,D) \)

\[
\text{MCD:} \quad \begin{array}{l}
\text{mapping:} \\
\text{id} \rightarrow I \\
\text{dir} \rightarrow D
\end{array}
\]

covered subgoals of \( Q \): \{2,3\}
$Q(title, year, dir) : –\text{Movie}(ID, title, year, genre),$
$\quad Director(ID, dir), Actor(ID, dir)$
$V(I,D,D) : – Director(I,D), Actor(I,D),$
$\quad Movie(I,T,Y,G)$

Note: the third subgoal of the view is not included in the MCD.

MCD:
- mapping:
  - $ID \rightarrow I$
  - $dir \rightarrow D$

covered subgoals of $Q$ still: $\{2,3\}$
Inverse-Rules Algorithm

• A “logical” approach to AQUV
• Produces maximally-contained rewriting in polynomial time
  – To check whether the rewriting is equivalent to the query, you still need a containment check.
• Conceptually simple and elegant
  – Depending on your comfort with Skolem functions...
Given the following view:
\[ V_7(I,T,Y,G) : - \text{Movie}(I,T,Y,G), \text{Director}(I,D), \text{Actor}(I,D) \]

And the following tuple in \( V_7 \):
\[ V_7(79, \text{Manhattan}, 1979, \text{Comedy}) \]

Then we can infer the tuple:
\[ \text{Movie}(79, \text{Manhattan}, 1979, \text{Comedy}) \]

Hence, the following ‘rule’ is sound:
\[ \text{IN}_1: \text{Movie}(I,T,Y,G) : - V_7(I,T,Y,G) \]
Skolem Functions

Now suppose we have the tuple $V_7(79,\text{Manhattan},1979,\text{Comedy})$

Then we can infer that there exists some director. Hence, the following rules hold (note that they both use the same Skolem function):

\[ \text{IN}_2: \ Director(I,f_1(I,T,Y,G)) \leftarrow V_7(I,T,Y,G) \]
\[ \text{IN}_3: \ Actor(I,f_1(I,T,Y,G)) \leftarrow V_7(I,T,Y,G) \]
Inverse Rules in General
Rewriting = Inverse Rules + Query

\[ Q_2(title, year, genre) : \neg Movie(ID, title, year, genre) \]

Given \( Q_2 \), the rewriting would include:
IN\(_1\), IN\(_2\), IN\(_3\), Q\(_2\).

**Given input:** \( V_7(79, \text{Manhattan}, 1979, \text{Comedy}) \)
In reverse rules produce:

Movie(79, Manhattan, 1979, Comedy)
Director(79, \( f_1(79, \text{Manhattan}, 1979, \text{Comedy}) \))
Actor(79, \( f_1(79, \text{Manhattan}, 1979, \text{Comedy}) \))

\[ Movie(\text{Manhattan}, 1979, \text{Comedy}) \]
(the last tuple is produced by applying \( Q_2 \)).
Comparing Algorithms

• Bucket algorithm:
  – Good if there are many interpreted predicates
  – Requires containment check. Cartesian product can be big

• MiniCon:
  – Good at detecting interactions between subgoals
• Inverse-rules algorithm:
  – Conceptually clean
  – Can be used in other contexts (see later)
  – But may produce inefficient rewritings because it “undoes” the joins in the views (see next slide)
• Experiments show MiniCon is most efficient.
• Even faster:

Inverse Rules Inefficiency

Example

Query and view:

\[ Q(X,Y) : -e_1(X,Z), e_2(Z,Y) \]

\[ V(A,B) : -e_1(A,C), e_2(C,B) \]

Inverse rules:

\[ e_1(A,f_1(A,B)) : -V(A,B) \]

\[ e_2(f_1(A,B),B) : -V(A,B) \]

Now we need to re-compute the join...
View-Based Query Answering

• Maximally-contained rewritings are parameterized by query language.

• More general question:
  – Given a set of view definitions, view instances and a query, what are all the answers we can find?

• We introduce certain answers as a mechanism for providing a formal answer.
View Instances = Possible DB’s

Consider the two views:

\[ V_8(dir) : -Movie(ID, dir, actor) \]
\[ V_9(actor) : -Movie(ID, dir, actor) \]

And suppose the extensions of the views are:

\[ V_8 : \{ \text{Allen, Copolla} \} \]
\[ V_9 : \{ \text{Keaton, Pacino} \} \]
There are multiple databases that satisfy the above view definitions: (we ignore the first argument of *Movie* below)

DB1. {\{(Allen, Keaton), (Coppola, Pacino)\}}
DB2. {\{(Allen, Pacino), (Coppola, Keaton)\}}

If we ask whether Allen directed a movie in which Keaton acted, we can’t be sure.

Certain answers are those true in *all* databases that are consistent with the views and their extensions.
Certain Answers: Formal Definition
[Open-world Assumption]

- Given:
  - Views: $V_1, ..., V_n$
  - View extensions $v_1, ..., v_n$
  - A query $Q$

- A tuple $t$ is a certain answer to $Q$ under the open-world assumption if $t \in Q(D)$ for all databases $D$ such that:
  - $V_i(D) \subseteq v_i$ for all $i$. 
Certain Answers
[Closed-world Assumption]

• Given:
  – Views: \( V_1, \ldots, V_n \)
  – View extensions \( v_1, \ldots, v_n \)
  – A query \( Q \)

• A tuple \( t \) is a certain answer to \( Q \) under the open-world assumption if \( t \in Q(D) \) for all databases \( D \) such that:
  – \( V_i(D) = v_i \) for all \( i \).
Certain Answers: Example

$V_8(dir) : \neg \text{Director}(ID,dir)$  \quad V_8: \{Allen\}

$V_9(actor) : \neg \text{Actor}(ID,actor)$  \quad V_9: \{Keaton\}

$Q(dir,actor) : \neg \text{Director}(ID,dir), \text{Actor}(ID,actor)$

Under closed-world assumption:
  single DB possible $\Rightarrow$ (Allen, Keaton)

Under open-world assumption:
  no certain answers.
The Good News

- The MiniCon and Inverse-rules algorithms produce all certain answers
  - Assuming no interpreted predicates in the query (ok to have them in the views)
  - Under open-world assumption
  - Corollary: they produce a maximally-contained rewriting
In Other News…

• Under closed-world assumption finding all certain answers is co-NP hard!

**Proof:** *encode a graph - G = (V,E)*

\[ v_1(X) : \neg \text{color}(X,Y) \quad I(V_1) = V \]
\[ v_2(Y) : \neg \text{color}(X,Y) \quad I(V_2) = \{\text{red, green, blue}\} \]
\[ v_3(X,Y) : \neg \text{edge}(X,Y) \quad I(V_3) = E \]

\[ q() : \neg \text{edge}(X,Y), \text{color}(X,Z), \text{color}(Y,Z) \]

q has a certain tuple iff G is not 3-colorable
Interpreted Predicates

• In the views: no problem (all results hold)
• In the query Q:
  – If the query contains interpreted predicates, finding all certain answers is co-NP-hard even under open-world assumption
  – Proof: reduction to CNF.
0) Course Info
1) Introduction
2) Data Preparation and Cleaning
3) Schema matching and mapping
4) Virtual Data Integration
5) Data Exchange
6) Data Warehousing
7) Big Data Analytics
8) Data Provenance