Manipulation of Query Expressions

Helena Galhardas, DEI/IST

Recap. Virtual Data Integration Architecture

Query

Mediated Schema

Semantic Mappings

S1

S2

S3
References

- Part of Chapter 2 (Sections 2.1, 2.2, 2.3.1, 2.4.1, 2.4.2), “Principles of Data Integration” by AnHai Doan, Alon Halevy, Zachary Ives

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
- Conjunctive queries
- Datalog
- Query unfolding
- Query containment and equivalence
- Answering queries using views

Query unfolding useful for…

```
Q() :- MR1(), MR2()
Q'() :- DS1(), DS2()
```
Query containment and equivalence useful for…

- Q' equivalent to Q''?
- Q' contains Q?

Answering queries using views useful for…

- V is defined over DS1 and DS2
- Can V be used to answer Q?
Basic Database Concepts

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

(Known) Terminology (1)

- **Relational database** composed of a set of relations (or tables)
- **Database schema** includes a relational schema for each table + set of integrity constraints
  - Key constraints, functional dependencies, foreign key constraints
  - Ex: Interview(candidate:string, date:date, recruiter:string, grade:float, hireDecision:boolean)
  - Candidate -> date, recruiter, grade, hireDecision
- The rows of a relational table are called **tuples** (or records)
- The **NULL** value means the value is not known or doesn't exist
  - Equality test envolving NULL always returns NULL
- A **database instance** is a particular snapshot of the contents of the DB
Foreign Keys

\[ S(A) \]
\[ T(B) \]
\[ B: \text{FK} \ S(A) \]
- The attribute B in T is a foreign key if whenever there is a row in T where the value of B is v, then there must exist a row in S where the value of A is v

(Known) Terminology (2)
- Queries used to formulate users’ needs
  - Structured (SQL or XQuery) or unstructured (for the Web, e.g., list of keywords)
  - Q(D): result of applying query Q to the database D
- Views are created when we want to reuse the same query expression in other queries
  - Materialized view: the answer is computed and maintained as the DB changes
  - Queries can also be used to specify relationships between schemata of data sources in a data integration context
- Two different notations for queries:
  - SQL
  - Conjunctive queries, for formal purposes
Conjunctive queries

- Based on Mathematical Logic
- Conjunctive query has the form:
  \[ Q(X) : - R_1(X_1), \ldots, R_n(X_n), c_1, \ldots, c_m \]
  where \( X, X_i \) are sets of attributes
  - \( R_1(X_1), \ldots, R_n(X_n) \) are the sub-goals (or conjuncts) and form the body of the query
  - \( R_i \)s are database relations, \( X_i \)s are tuples of variables and constants
  - The variables in \( X \) are called head variables or distinguished variables; the others are called existential variables
  - The predicate \( Q \) denotes the answer relation of the query
  - The \( c_i \)'s are interpreted atoms and have the form \( X \theta Y \), with \( X, Y \) either variables or constants (but at least one is a variable), and \( \theta \) is an interpreted predicate such as =, <=, >, !=, >=

Semantics of conjunctive queries

- Semantics of a conjunctive query \( Q \) over a DB instance \( D \):
  - \( \Psi \): any mapping that maps each of the variables in \( Q \) to constants in \( D \)
  - \( \Psi(R_i) \): result of applying \( \Psi \) to \( R_i(X_i) \), which consists of ground atoms (constants)
  - \( \Psi(c_j) \): result of applying \( \Psi \) to \( c_j \), which consists of ground atoms (constants)
  - \( \Psi(Q) \): result of applying \( \Psi \) to \( Q(X) \), which consists of ground atoms
  - If
    - Each of \( \Psi(R_1), \ldots, \Psi(R_n) \) is in \( D \), and
    - For each \( 1 \leq j \leq m \), \( \Psi(c_j) \) is satisfied,
  - Then, \( \Psi(Q) \) is in the answer to \( Q \) over \( D \)
Correspondence between SQL queries and conjunctive queries

Base relations:
- Interview(candidate, date, recruiter, hireDecision, grade)
- EmployeePerformance(empID, name, reviewQuarter, grade, reviewer)

SQL:
```sql
SELECT recruiter, candidate
FROM Interview i, EmployeePerformance e
WHERE i.recruiter = e.name
AND e.grade <= 2.5
```

Conjunctive query:
```
Q1(Y,X) :- Interview(X,D,Y,H,F),
          EmployeePerformance(E,Y,T,W,Z),
          W <= 2.5
```

Safety, disjunction

- Conjunctive queries must be safe:
  - Every variable appearing in the head also appears in the body
  - Otherwise, the set of possible answers to the query may be infinite
- Disjunction can be expressed writing two or more conjunctive queries with the same head predicate:
```
Q1(Y,X) :- Interview(X,D,Y,H,F),
          EmployeePerformance(E,Y,T,W,Z), W <= 2.5
Q1(Y,X) :- Interview(X,D,Y,H,F),
          EmployeePerformance(E,Y,T,W,Z), W >= 3.9
```
Negation

- Conjunctive queries with negated goals can also be considered:

\[ Q(X) : - R_1(X_1), ..., R_n(X_n), \neg S_1(Y_1), ..., \neg S_m(Y_m) \]

- The notion of safety is extended to:
  - Any variable appearing in the head of the query must also appear in a positive subgoal

Datalog program

- Set of Horn rules, each of which is a conjunctive query
- Instead of computing a single answer query, computes a set of intensional relations (IDB relations), one of them is designated the query predicate
  - Extensional Database (EDB) relations: are the database relations, given by a set of tuples; can occur only in the body of rules
  - Intensional Database (IDB) relations: are defined by a set of rules; can occur both in the head and in the body of rules
Datalog Terminology

- An example datalog rule:
  \[ \text{idb}(x,y) :- \text{r1}(x,z), \text{r2}(z,y), z < 10 \]

- Irrelevant variables can be replaced by \_ (anonymous var)
- Extensional relations or database schemas (edbs) are relations only occurring in rules’ bodies, or as base relations:
  - Ground facts only have constants, e.g., \text{r1}(‘abc’, 123)
- Intensional relations (idb) appear in the heads – these are basically views
  - Distinguished variables are the ones output in the head

Example

- A database that includes a relation representing the edges in a graph: \text{edge}(X, Y)

The Datalog program:

\[ \begin{align*}
\text{r1}: & \quad \text{path}(X,Y) :- \text{edge}(X,Y) \\
\text{r2}: & \quad \text{path}(X,Y) :- \text{edge}(X,Z), \text{path}(Z,Y)
\end{align*} \]

computes the paths in the graph

If \text{r2} was replaced by:

\[ \begin{align*}
\text{r3}: & \quad \text{path}(X,Y) :- \text{path}(X,Z), \text{path}(Z,Y)
\end{align*} \]

It would produce the same result
Semantics of a Datalog program

1. Start with empty extensions for the IDB predicates
2. Choose a rule in the program and apply it to the current extension of the EDB and IDB relations
3. Add the tuples computed for the head to its extension
4. Continue applying the rules of the program until no new tuples are computed for the IDB relations
5. The answer to the query is the extension of the query predicate

Example (cont)

\[
\text{r1: } \text{path}(X,Y) \leftarrow \text{edge}(X,Y) \\
\text{r2: } \text{path}(X,Y) \leftarrow \text{edge}(X,Z), \text{path}(Z,Y)
\]

Start with:
\[
\text{edge(1,2), edge(2,3), edge(3,4)}
\]
Apply r1 returns:
\[
\text{path(1,2), path(2,3), path(3,4)}
\]
Apply r2 once:
\[
\text{path(1,3), path(2,4)}
\]
Apply r2 twice:
\[
\text{path(1,4)}
\]
Outline

- Conjunctive queries
- Datalog
- **Query unfolding**
- Query containment and equivalence
- Answering queries using views

Query unfolding/expansion

- **Compositionality**: we can write queries that refer to views (named queries) in their body
  - Advantage of declarative languages
- **Query unfolding**: process of undoing the composition of queries
  - Given a query that refer to views, query unfolding rewrites the query so that it refers to database tables
For example

```
CREATE VIEW V(name, age) AS
SELECT name, age
FROM Employees
WHERE salary > 1000;

SELECT name
FROM V
WHERE age = 20;

Unfolding:
SELECT name
FROM (SELECT name, age
FROM Employees
WHERE salary > 1000) as A
WHERE A.age = 20;
```

Example

- Relation **Flight** stores pairs of cities between which there is a direct flight
- Relation **Hub** stores the set of hub cities of the airline
- Query **Q1** asks for pairs of cities between which there is a flight that goes through a hub
- Query **Q2** asks for pairs of cities that are on the same outgoing path from a hub

```
Q1(X,Y) :- Flight(X,Z), Hub(Z), Flight(Z,Y)
Q2(X,Y) :- Hub(Z), Flight(Z,X), Flight(X,Y)
Q3(X,Z) :- Q1(X,Y), Q2(Y,Z)
```

Unfolding of Q3 is:

```
Q3'(X,Z) :- Flight(X,U), Hub(U), Flight(U,Y), Hub(W), Flight(W,Y), Flight(Y,Z)
```
Observations

- Query unfolding may lead to a resulting query that has subgoals that seem redundant
  - There are algorithms to eliminate those
- The query and the view may have interpreted predicates that are satisfied in isolation, but after the unfolding, the query is unsatisfiable
- Through repeated applications of the unfolding step, the number of subgoals may increase exponentially
- Unfolding does not necessarily yield more efficient ways to execute the query
- Unfolding allows the query processor to explore a larger set of query plans due to the existence of a larger set of possible orderings of join operations in the query

Query containment and equivalence

$Q_3'(X,Z) :- \text{Flight}(X,U), \text{Hub}(U), \text{Flight}(U,Y), \text{Hub}(W), \text{Flight}(W,Y), \text{Flight}(Y,Z)$

Should produce the same result as:

$Q_4(X,Z) :- \text{Flight}(X,U), \text{Hub}(U), \text{Flight}(U,Y), \text{Flight}(Y,Z)$

The result of $Q_4$ should be a superset of the result of $Q_5$:

$Q_5(X,Z) :- \text{Flight}(X,U), \text{Hub}(U), \text{Flight}(U,Y), \text{Hub}(Y), \text{Flight}(Y,Z)$

That considers both intermediate stops to be hubs
Formal definition

Containment and equivalence

- Let Q1 and Q2 be two queries of the same arity.
  - Q1 is contained in Q2 ($Q1 \subseteq Q2$), if for any database $D$, $Q1(D) \subseteq Q2(D)$

- Q1 is equivalent to Q2 ($Q1 \equiv Q2$) if $Q1 \subseteq Q2$ and $Q2 \subseteq Q1$

Another Example of Containment

Suppose we have two queries:

$q1(S,C) :- \text{Student}(S,N), \text{Takes}(S,C), \text{Course}(C,'DB & Info Systems'), \text{inCIS}(C)$

$q2(S,C) :- \text{Student}(S,N), \text{Takes}(S,C), \text{Course}(C,X)$

Intuitively, q1 must contain the same or fewer answers vs. q2:
- It has the same conditions, except one extra condition and one extra conjunction (i.e., it's more restricted)
- There's no union or any other way it can add more data

We can say that $q2 \text{ contains } q1$ because this holds for any instance of our DB {Student, Takes, Course}
Outline

- Conjunctive queries
- Datalog
- Query unfolding
- Query containment and equivalence
- Answering queries using views

Answering queries using views

- Important use of query containment: when we detect that query Q1 is contained in query Q2, then the answer to Q1 can be often obtained from the answer to Q2

- Next problem to solve: When can we answer a query Q from a collection of views?
Example

Given the following database of films:

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

and the query asking for comedies produced after 1950 where the director was also an actor in the movie

\[ Q(T, Y, D) :- \text{Movie}(I, T, Y, G), Y \geq 1950, G = 'Comedy', \text{Director}(I, D), \text{Actor}(I, D) \]

Suppose there is the following pre-computed view available:

\[ V1(T, Y, D) :- \text{Movie}(I, T, Y, G), Y \geq 1940, G = 'Comedy', \text{Director}(I, D), \text{Actor}(I, D) \]

Then:

\[ Q \subseteq V1 \]

Example (cont.)

- We can use V1 to answer Q, because the Year attribute is in the head of V1

\[ Q'(T, Y, D) :- V1(T, Y, D), Y \geq 1950 \]

- This can be more efficient, because the join operations in Q do not need to be executed, since V1 is pre-computed
Another example

- If, instead of V1, we only have access to the views:

  \[ \text{V2}(I, T, Y) : \text{Movie}(I, T, Y, G), \ Y \geq 1950, \\
      G = 'Comedy' \]

  \[ \text{V3}(I, D) : \text{Director}(I, D), \text{Actor}(I, D) \]

- Neither V2 nor V3 contain Q, but we can still answer Q using V2 and V3:

  \[ Q' (T, Y, D) : \text{V2}(I, T, Y), \text{V3}(I, D) \]

Answering queries using views – formal definition

- Given a query Q and a set of views V1, ..., Vm, an equivalent rewriting of the query using the views is a query expression Q’ that refers only to views V1, ..., Vm and Q is equivalent to Q’
  - In SQL, a query refers only to views if all the relations mentioned in the from clause are views
  - In conjunctive queries, that means all the subgoals are views with the exception of interpreted atoms.

- Example:

  \[ Q'' (T, Y, D) : \text{V2}(I, T, Y), \text{V3}(I, D) \]

  is an equivalent rewriting of Q using V2 and V3
Example

- There may not always exist an equivalent rewriting of a query using a set of views.
- In that case, we may want to know what is the best we can do with the set of views.
- If we have:

  \[
  V_4(I,T,Y) \leftarrow \text{Movie}(I,T,Y,G), Y \geq 1960, G = 'Comedy'
  \]

  It cannot be used to completely answer Q:

  \[
  Q(T,Y,D) \leftarrow \text{Movie}(I,T,Y,G), Y \geq 1950, G = 'Comedy', \text{Director}(I,D), \text{Actor}(I,D)
  \]

  The best we can obtain in terms of rewriting is:

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

  \[
  V_3(I,D) \leftarrow \text{Director}(I,D), \text{Actor}(I,D)
  \]

  Q''' is called a maximally-contained rewriting of Q using V3 and V4.

Maximally-contained rewritings

- Let Q be a query and V1, ...,Vm a set of view definitions, and L a query language. The query Q’ is a maximally-contained rewriting of Q using V1, ..., Vm wrt L if:
  - Q’ is a query in L that refers only to views V1,..., Vm.
  - Q’ is contained in Q, and
  - There is no rewriting Q1 \in L, such that Q’ \subseteq Q1 and Q1 is not equivalent to Q’.

- When a rewriting Q’ is contained in Q but it is not a maximally-contained rewriting, we call it a contained rewriting.

  - There are algorithms to compute the rewriting of a query using a set of views, but we’ll skip them in this course.
Example

- If we have:
  \[ V_5(I,T,Y) \leftarrow \text{Movie}(I,T,Y,G), \ Y \geq 1950, \ Y \leq 1955, \ G = \text{Comedy} \]

- Then, if the language of rewriting consists of unions of conjunctive queries, the following is a maximally-contained rewriting of Q:
  \[ Q'''(T,Y,D) \leftarrow V_4(I,T,Y), \ V_3(I,D) \]
  \[ Q'''(T,Y,D) \leftarrow V_5(I,T,Y), \ V_3(I,D) \]

When is a view relevant to a query?

- If the set of relations it mentions overlaps with that of the query

- If the query applies predicates to attributes it has in common with the view, then the view must apply either equivalent or logically weaker predicates to be part of an equivalent rewriting

- If the view applies a logically stronger predicate, it may be part of a contained rewriting
Examples

To answer the query:
\[ Q(T,Y,D) : Movie(I,T,Y,G), Y \geq 1950, G = \text{`Comedy'}, \ Director(I,D), \ Actor(I,D) \]
Can the following views be used?

\[ V6(T,Y) : Movie(I,T,Y,G), Y \geq 1950, G = \text{`Comedy'} \]
Similar to V2, but doesn’t include the ID attribute of Movie in the head, so cannot be joined with V3

\[ V7(I,T,Y) : Movie(I,T,Y,G), Y \geq 1950, G = \text{`Comedy'}, \ Award(I,W) \]
Applies an additional condition (Award) that doesn’t exist in the query and cannot be used in an equivalent rewriting

\[ V8(I,T) : Movie(I,T,Y,G), Y \geq 1940, G = \text{`comedy'} \]
Applies a weaker predicate on the year than in the query, but the year is not included in the head

Next Lecture

- Describing Data Sources
Equivalent query rewrites

- Given a query Q and a set of views V1, ..., Vm, the query Q’ is an equivalent rewriting of Q using V1, ...Vm if
  - Q’ refers only to the views in V1, ...,Vm
  - Q’ is equivalent to Q

Example:
- Q’’ (T, Y, D) :- V2(I, T, Y), V3(I, D) is an equivalent rewriting of Q using V2 and V3