Information system architecture enabled by DBMS

Pre-DBMS architecture (need of a unified data storage):

```
  Application
  ↓
Data sources
  ↓
  Application
  ↓
  Application
```

“Ideal information system architecture” with DBMS (’70s):

```
  Application
  ↓
Database
  ↓
  Application
  ↓
  Application
  ↓
  Application
```
Today in many organizations ...

- Distributed, redundant, application-dependent, and mutually incoherent data
- Desperate need of a coherent, conceptual, unified view of data
... even with just one data source

Fragment of a relational table in a Bank Information system:

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>0 00</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... even with just one data source

Negative value denotes a holding

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>0 00</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... even with just one data source

*S means that the customer is the leader of the group it belongs to*

*S means that the customer is the head of the group it belongs to*

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>000</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... even with just one data source

<table>
<thead>
<tr>
<th>CUC</th>
<th>TS_START</th>
<th>TS_END</th>
<th>ID_GRUP</th>
<th>FLAG_CP</th>
<th>FLAG_CF</th>
<th>FATTURATO</th>
<th>FLAG_FATT</th>
</tr>
</thead>
<tbody>
<tr>
<td>124589</td>
<td>30-lug-2004</td>
<td>1-gen-9999</td>
<td>92736</td>
<td>S</td>
<td>N</td>
<td>195000,00</td>
<td>N</td>
</tr>
<tr>
<td>140904</td>
<td>15-mag-2001</td>
<td>15-giu-2005</td>
<td>35060</td>
<td>N</td>
<td>N</td>
<td>230600,00</td>
<td>N</td>
</tr>
<tr>
<td>124589</td>
<td>5-mag-2001</td>
<td>30-lug-2004</td>
<td>92736</td>
<td>N</td>
<td>S</td>
<td>195000,00</td>
<td>S</td>
</tr>
<tr>
<td>-452901</td>
<td>13-mag-2001</td>
<td>27-lug-2004</td>
<td>92770</td>
<td>S</td>
<td>N</td>
<td>392000,00</td>
<td>N</td>
</tr>
<tr>
<td>129008</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>N</td>
<td>S</td>
<td>247000,00</td>
<td>S</td>
</tr>
<tr>
<td>-472900</td>
<td>10-mag-2001</td>
<td>1-gen-9999</td>
<td>62010</td>
<td>S</td>
<td>N</td>
<td>0 00</td>
<td>N</td>
</tr>
<tr>
<td>130976</td>
<td>7-mag-2001</td>
<td>9-lug-2003</td>
<td>75680</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N means that the FATTURATO field is not valid.
Large enterprises spend a great deal of time and money on data preparation and information integration (∼40% of information-technology shops’ budget).

Market for information integration software estimated to grow to $3.4 billion by 2019 [Gartner, 2015]

Data integration is a large and growing part of software development, computer science, and specific applications settings, such as scientific computing, semantic web, etc..

Data preparation and integration is crucial for “big data” processing (to make sense of big data!)

Basing the integrated view of data on a clean, rich and abstract conceptual representation of the data has always been both a goal and a challenge [Mylopoulos et al 1984]
Ontology-based Data Management is a new paradigm, rooted on the idea of using Database Theory fundamentals, and Logic-based Knowledge Representation and Reasoning techniques for a new way of managing data, and characterized by the following principles:

- Data may reside where they are (no need to move data)
- Build a conceptual specification of the domain of interest, in terms of knowledge structures
- Map such knowledge structures to concrete data sources
- Express all services over the knowledge structures
- Automatically translate knowledge services to data services
Ontology-based data management: architecture

Based on three main components:

- **Ontology**, a declarative, logic-based specification of the domain of interest, used as a unified, conceptual view for clients
- **Data sources**, representing external, independent, heterogeneous, storage (or, more generally, computational) structures
- **Mappings**, used to semantically link data at the sources to the ontology
Part I
- Ontology-based data management: The framework
- Queries in OBDM
- The nature of query answering in OBDM

Part II
- Ontology languages
- Modeling the domain through the ontology
- Modeling the mapping with the data sources

Part III
- Algorithms for query answering
- Beyond classical first-order queries
Outline of part I

1. Ontology-based data management: The framework
2. Queries in OBDM
3. The nature of query answering in OBDM
An ontology-based data management (OBDM or OBDA) system is a triple \( \Sigma = \langle \mathcal{O}, \mathcal{S}, \mathcal{M} \rangle \), where

- \( \mathcal{O} \) is the ontology, expressed as a logical theory (here, a TBox in a Description Logic)
- \( \mathcal{S} \) is a relational database representing the data sources (note that federation tools are able to present a set of heterogeneous data sources as a single relational database)
- \( \mathcal{M} \) is a set of mapping assertions, each one of the form

\[
\Phi(\vec{x}) \sim \Psi(\vec{x})
\]

where

- \( \Phi(\vec{x}) \) is a FOL query over \( \mathcal{S} \), returning values for \( \vec{x} \)
- \( \Psi(\vec{x}) \) is a FOL query over \( \mathcal{O} \), whose free variables are from \( \vec{x} \).
## Ontology-based data management system – Example

### Ontology $O$

<table>
<thead>
<tr>
<th>DL notation</th>
<th>Classical FOL notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee $\sqsubseteq \exists$ worksFor</td>
<td>$\forall x \text{Employee}(x) \rightarrow \exists y \text{worksFor}(x, y)$</td>
</tr>
<tr>
<td>Employee $\sqsubseteq \exists$ empCode</td>
<td>$\forall x \text{Employee}(x) \rightarrow \exists y \text{empCode}(x, y)$</td>
</tr>
<tr>
<td>Employee $\sqsubseteq \exists$ salary</td>
<td>$\forall x \text{Employee}(x) \rightarrow \exists y \text{salary}(x, y)$</td>
</tr>
<tr>
<td>Project $\sqsubseteq \exists$ worksFor$^-$</td>
<td>$\forall x \text{Project}(x) \rightarrow \exists y \text{worksFor}(y, x)$</td>
</tr>
<tr>
<td>Project $\sqsubseteq \exists$ projectName</td>
<td>$\forall x \text{Project}(x) \rightarrow \exists y \text{projectName}(x, y)$</td>
</tr>
<tr>
<td>$\exists$ worksFor $\sqsubseteq$ Employee</td>
<td>$\forall x \forall y \text{worksFor}(x, y) \rightarrow \text{Employee}(x)$</td>
</tr>
<tr>
<td>$\exists$ worksFor$^-$ $\sqsubseteq$ Project</td>
<td>$\forall x \forall y \text{worksFor}(x, y) \rightarrow \text{Project}(y)$</td>
</tr>
</tbody>
</table>

- DLs use unary predicates (concepts, or classes), and binary predicates between classes (relations, or roles, or object properties), and other binary predicates relating classes to value types (attributes, or data properties).
- $\sqsubseteq$ corresponds to $\sqsubseteq$
- $R^-$ denotes the inverse of the relation $R$
- $\lambda x.C(x)$ is written as $C$
- $\lambda x.\exists y R(x, y)$ is written as $\exists R$
Ontology-based data management system – Example

Ontology $\mathcal{O}$ (TBox)

- Employee $\sqsubseteq \exists \text{worksFor}$
- Employee $\sqsubseteq \exists \text{empCode}$
- Employee $\sqsubseteq \exists \text{salary}$
- Project $\sqsubseteq \exists \text{worksFor}^{-}$
- Project $\sqsubseteq \exists \text{projectName}$
- $\exists \text{worksFor} \sqsubseteq \text{Employee}$
- $\exists \text{worksFor}^{-} \sqsubseteq \text{Project}$

Federated schema of the DB $\mathcal{S}$

- $D_1[SSN: String, \ PrName: String]$  
  Employees and Projects they work for
- $D_2[\text{Code}: String, \ Salary: Int]$  
  Employee’s Code with salary
- $D_3[\text{Code}: String, \ SSN: String]$  
  Employee’s Code with SSN

Mapping $\mathcal{M}$

- $M_1$: SELECT SSN,PrName  $\leadsto$  $V_1(\text{SSN,PrName})$  $\leadsto$  Employee($\text{pers}(\text{SSN}))$, Project($\text{proj}(\text{PrName}))$, projectName($\text{proj}(\text{PrName}), \text{PrName}$), workFor($\text{pers}(\text{SSN}), \text{proj}(\text{PrName})$)
  FROM $D_1$

- $M_2$: SELECT SSN,Salary  $\leadsto$  $V_2(\text{SSN,Salary})$  $\leadsto$  Employee($\text{pers}(\text{SSN}))$, salary($\text{pers}(\text{SSN}), \text{Salary}$)
  FROM $D_2$, $D_3$
  WHERE $D_2\.\text{Code} = D_3\.\text{Code}$

Note: in practice we often write mappings using an intermediate view symbol.
Semantics

Let $\mathcal{I} = (\Delta \mathcal{I}, \cdot \mathcal{I})$ be an interpretation for the ontology $\mathcal{O}$, where $\Delta \mathcal{I}$ is the domain and $\cdot \mathcal{I}$ is the interpretation function.

**Def.: Mapping satisfaction (sound mappings)**

We say that $\mathcal{I}$ satisfies $\Phi(\bar{x}) \leadsto \Psi(\bar{x})$ wrt a database $S$, if the sentence

$$\forall \bar{x} \ (\Phi(\bar{x}) \rightarrow \Psi(\bar{x}))$$

is true in $\mathcal{I} \cup S$.

**Def.: Model**

$\mathcal{I} = (\Delta \mathcal{I}, \cdot \mathcal{I})$ is a model of $\Sigma = \langle \mathcal{O}, S, \mathcal{M} \rangle$ if:

- $\mathcal{I}$ is a model of $\mathcal{O}$, i.e., it satisfies all axioms in $\mathcal{O}$;
- $\mathcal{I}$ satisfies $\mathcal{M}$ wrt $S$, i.e., satisfies every assertion in $\mathcal{M}$ wrt $S$.

**Def.: Semantics**

The semantics of $\Sigma$ is the set $sem(\Sigma)$ of all models of $\Sigma$. 
Ontology-based data management (OBDM): topics

- Ontology-based [data access | query answering] (OBDA | OBQA)
- Ontology-based data quality (OBDQ)
- Ontology-based data governance (OBDG)
- Ontology-based data restructuring (OBDR)
- Ontology-based business intelligence (OBBI)
- Ontology-based data exchange and coordination (OBDE)
- Ontology-based data update (OBDU)
- Ontology-based service and process management (OBDS)

General requirements:

- large data collections
- efficiency with respect to size of data (data complexity)
Outline of part I

1. Ontology-based data management: The framework
2. Queries in OBDM
3. The nature of query answering in OBDM
Conjunctive queries

- are the most common kind of first-order queries
- also known as select-project-join SQL queries
- allow for easy optimization in relational DBMSs

Definition

A conjunctive query (CQ) is a first-order query of the form

\[
\{ (\bar{x}) \mid \exists \bar{y}. \ r_1(\bar{x}_1, \bar{y}_1) \land \cdots \land r_m(\bar{x}_m, \bar{y}_m) \}
\]

where

- \(\bar{x}\) is the union of the \(\bar{x}_i\)'s, and \(\bar{y}\) is the union of the \(\bar{y}_i\)'s
- \(r_1, \ldots, r_m\) are relation symbols (not built-in predicates)
Conjunctive queries

- are the most common kind of first-order queries
- also known as `select-project-join` SQL queries
- allow for easy optimization in relational DBMSs

**Definition**

A **conjunctive query** (CQ) is a first-order query of the form

\[ \{ (\vec{x}) \mid \exists \vec{y}. \ r_1(\vec{x}_1, \vec{y}_1) \land \cdots \land r_m(\vec{x}_m, \vec{y}_m) \} \]

where

- \(\vec{x}\) is the union of the \(\vec{x}_i\)'s, and \(\vec{y}\) is the union of the \(\vec{y}_i\)'s
- \(r_1, \ldots, r_m\) are relation symbols (not built-in predicates)

We use the following abbreviation:

\[ \{ (\vec{x}) \mid r_1(\vec{x}_1, \vec{y}_1), \ldots, r_m(\vec{x}_m, \vec{y}_m) \} \]
We consider the complexity of the recognition problem, i.e., checking whether a tuple of constants is in the answer to a query:

- measured wrt the size of the database $\leadsto$ data complexity
- measured wrt the size of the query and the database $\leadsto$ combined complexity

Complexity of relational calculus

- data complexity: polynomial, actually in LogSpace (or, in terms of circuit complexity, in $AC_0$)
- combined complexity: PSPACE-complete

Complexity of conjunctive queries

- data complexity: in LogSpace
- combined complexity: NP-complete
We consider the complexity of the recognition problem, i.e., checking whether a tuple of constants is in the answer to a query:

- measured wrt the size of the database \(\sim\) data complexity
- measured wrt the size of the query and the database \(\sim\) combined complexity

### Complexity of relational calculus

- data complexity: polynomial, actually in \(\operatorname{LogSpace}\) (or, in terms of circuit complexity, in \(\operatorname{AC}_0\))
- combined complexity: \(\operatorname{PSPACE}\)-complete

### Complexity of conjunctive queries

- data complexity: in \(\operatorname{LogSpace}\)
- combined complexity: \(\operatorname{NP}\)-complete
Complexity of relational calculus

We consider the complexity of the recognition problem, i.e., checking whether a tuple of constants is in the answer to a query:

- measured wrt the size of the database \( \sim \) data complexity
- measured wrt the size of the query and the database \( \sim \) combined complexity

Complexity of relational calculus

- data complexity: polynomial, actually in \( \text{LogSpace} \) (or, in terms of circuit complexity, in \( \text{AC}_0 \))
- combined complexity: \( \text{PSPACE} \)-complete

Complexity of conjunctive queries

- data complexity: in \( \text{LogSpace} \)
- combined complexity: \( \text{NP} \)-complete
Queries in OBDM

- The domain $\Delta$ is fixed, and we do not distinguish an element of $\Delta$ from the constant denoting it $\leadsto$ standard names
- Queries to $\Sigma = \langle O, S, M \rangle$ are first-order queries over the alphabet $A_O$ of the ontology
- When “evaluating” $q$ over $\Sigma$, we have to consider that there may be many interpretation in $sem(\Sigma)$
- We consider those answers to $q$ that hold for all models in $sem(\Sigma)$ $\leadsto$ certain answers
Semantics of queries to $\Sigma$

**Definition**

Given an OBDM system $\Sigma$ and query $q$ posed to $\Sigma$, the set of certain answers to $q$ wrt $\Sigma$ is

$$\text{cert}(q, \Sigma) = \bigcap \left\{ q^M \mid M \in \text{sem}(\Sigma) \right\}$$

- Query answering in OBDM means to compute the certain answers, i.e., it corresponds to logical implication.
- Complexity is usually measured wrt the size of the source db $S$, i.e., we consider data complexity.
- When we want to look at query answering as a decision problem, we consider the problem of deciding whether a given tuple $\vec{c}$ is a certain answer to $q$ wrt $\Sigma$, i.e., whether $\vec{c} \in \text{cert}(q, \Sigma)$. 
Which languages?

- Which language for expressing the ontology?
  - We use Description Logics (OWL), but which one?

- Which language for expressing the mappings?
  - We use logic, but which fragment?

- Which language for expressing queries over the ontology?
  - At least classical conjunctive queries, but we aim at using SPARQL

Challenge: optimal compromise between expressive power and data complexity.
Ontology-based data management: The framework

Queries in OBDM

The nature of query answering in OBDM
Abstracting from the mapping

For the moment, let us abstract from the mapping: we assume that all the semantics of mappings can be captured by computing $\mathcal{M}(S)$, which is the database obtained by treating mappings as assertions translating the data at the sources into facts expressed over the alphabet of the ontology.

$\mathcal{M}(S)$ can indeed be seen as a set of facts built on the alphabet of $\mathcal{O}$ (i.e., a set of ground atomic formulas in logic, or simply, an **ABox**, in DL terminology). In other words, formally, we can consider our system as constituted by the pair $\langle \mathcal{O}, \mathcal{A} \rangle$ where $\mathcal{O}$ is the TBox, and $\mathcal{A}$ is the (virtual) ABox.

In practice, instead of computing $\mathcal{M}(S)$ and consider queries over such set of facts, one can use $\mathcal{M}$ to rewrite a query expressed over $\mathcal{M}(S)$ into a query expressed over $S$, using $\mathcal{M}$.
Which ontology language?

$q(x) \leftarrow \supervisedBy(x, y), \text{ComputerScientist}(y), \text{hates}(y, z), \text{ComputerEngineering}(z)$
Is ontology-based query answering essentially the same problem as query answering in databases?

In other words, is query answering just evaluating a formula over a (finite) interpretation?
QA in OBDM – Example (*)

Note that ComputerProfessor is partitioned into ComputerScientist and ComputerEngineer.

(*) [Andrea Schaerf 1993]
QA in OBDM – Example (cont’d)

To determine this answer, we need to resort to reasoning by cases on the instances.

\[ q(x) \leftarrow \text{supervisedBy}(x, y), \text{ComputerScientist}(y), \text{hates}(y, z), \text{ComputerEngineer}(z) \]

**Answer:** ???
To determine this answer, we need to resort to reasoning by cases on the instances.
### Complexity of conjunctive query answering in DLs

<table>
<thead>
<tr>
<th></th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain databases</td>
<td>NP-complete</td>
<td>in \texttt{LogSpace} (^{(1)})</td>
</tr>
<tr>
<td>OWL 2</td>
<td>?</td>
<td>\texttt{coNP-hard} (^{(2)})</td>
</tr>
</tbody>
</table>

\(^{(1)}\) Going beyond probably means not scaling with the data.

\(^{(2)}\) Already for a TBox with a single disjunction (see example above).

---

**Questions**

- Can we find interesting DLs for which the query answering problem can be solved efficiently (in \texttt{LogSpace} wrt data complexity)?
- If yes, can we leverage relational database technology for query answering in OBDM?
Complexity of conjunctive query answering in DLs

<table>
<thead>
<tr>
<th></th>
<th>Combined complexity</th>
<th>Data complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain databases</td>
<td>NP-complete</td>
<td>in LogSpace ^{(1)}</td>
</tr>
<tr>
<td>OWL 2</td>
<td>?</td>
<td>coNP-hard ^{(2)}</td>
</tr>
</tbody>
</table>

(1) Going beyond probably means not scaling with the data.
(2) Already for a TBox with a single disjunction (see example above).

Questions
- Can we find interesting DLs for which the query answering problem can be solved efficiently (in LogSpace wrt data complexity)?
- If yes, can we leverage relational database technology for query answering in OBDM?
Query rewriting

Query answering can **always** be thought as done in two phases:

1. **Rewriting (wrt the ontology):** produce from $q$ and the TBox $O$ a new query $r_{q,O}$.

2. **Query evaluation:** evaluate $r_{q,O}$ over $\mathcal{M}(S)$ seen as a complete database (and without considering $O$).

$\rightsquigarrow r_{q,O}$ is the so-called perfect rewriting of $q$ w.r.t. $O$ exactly when the query evaluation step produces $\text{cert}(q, \langle O, \mathcal{M}(S) \rangle)$, for every $S$.

Note: The “always” holds if we pose no restriction on the language in which to express the rewriting $r_{q,O}$.

Note: if we have built $\mathcal{M}(S)$, then instead of evaluating $r_{q,O}$ over $\mathcal{M}(S)$, we rewrite $r_{q,O}$ wrt $\mathcal{M}$, and then we evaluate the resulting query over $S$. 
Let $Q$ be a class of queries (or query language) and $\mathcal{L}$ an ontology language.

**Def.: $Q$-rewritability**

Query answering is **$Q$-rewritable** if for every TBox $\mathcal{O}$ of $\mathcal{L}$ and for every query $q$, the perfect rewriting $r_{q,\mathcal{O}}$ of $q$ w.r.t. $\mathcal{O}$ can be expressed in the query language $Q$.

The notion of **FOL-rewritability** is particularly interesting, where FOL denotes the class of queries expressible in First-Order Logic.
The certain answers to the above query are computed by evaluating:

\[ q(x) \leftarrow \text{ComputerScientist}(x) \]

It can indeed be shown that we need transitive closure in the language of the rewriting.

\[ \forall x \forall y \text{ ComputerScientist}(x) \land \text{hates}(x, y) \rightarrow \text{ComputerScientist}(y) \]
The certain answers to the above query are computed by evaluating:

$$q'(x) \leftarrow \text{ComputerScientist}(x)$$

$$q'(x) \leftarrow \text{ComputerScientist}(y), \text{hates}^+(y, x)$$

It can indeed be shown that we need transitive closure in the language of the rewriting.
Questions

1. Can we find interesting DLs for which query answering is FOL-rewritable?
2. Even more specifically, can we find interesting DLs for which query answering is UQC-rewritable?

If yes, we can indeed leverage relational database technology for query answering in OBDM (RDBMs are generally very good at optimizing UCQs).
The expressiveness of the ontology language affects the query language into which we are able to rewrite CQs:

- When we can rewrite into **UCQ**.
  - Query evaluation can be “optimized” via **RDBMS**.
- When we can rewrite into **FOL/SQL**.
  - Query evaluation can be done in SQL, i.e., via **RDBMS**.
- When we can rewrite into **non recursive Datalog**.
  - Query evaluation can be still done via **RDBMS**, but with subqueries/views.
- When we can rewrite into an **NLOGSPACE-hard** language.
  - Query evaluation requires (at least) **linear recursion**.
- When we can rewrite into a **PTIME-hard** language.
  - Query evaluation requires full recursion (e.g., Datalog).
- When we can rewrite into a **coNP-hard** language.
  - Query evaluation requires (at least) **Disjunctive Datalog**.