Unifying Data and Domain Knowledge Using Virtual Views

Lipyeow Lim, Haixun Wang & Min Wang
Background

- **DBMS originally designed for transaction data**
- **Many extensions for richer queries attempted**
  - OO DBMS and ORDBMS
  - OLAP (1990s)
  - Data Cube (ICDE 1996)
  - Data Mining (CACM 1996)
- **An unending quest**
  - Database or Knowledge-base?
  - New applications: the Semantic web, etc.
- **Move from simple transactional or analytical processing to semantics understanding and knowledge inferencing**
A motivating example

- **RDBMS allows us to query wines through attributes** ID, Type, Origin, Maker, Price.
- **Expressive power: relational complete (quite limited).**
- **Human intelligence operates in a quite different way.**

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Origin</th>
<th>Maker</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Burgundy</td>
<td>CotesDOr</td>
<td>ClosDeVougeot</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>Riesling</td>
<td>NewZealand</td>
<td>Corbans</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Zinfandel</td>
<td>EdnaValley</td>
<td>Elyse</td>
<td>15</td>
</tr>
</tbody>
</table>

A base table: Wine
Query 1:

- **Which wine originates from the US?**
  - Answer: Zinfandel
  - Zinfandel’s **Origin** EdnaValley is located in California.
  - Domain knowledge used: EdnaValley is in California, and California is in the US.

- **We could issue:**

```
SELECT ID
FROM Wine
WHERE Origin = 'US';
```

Nice! Except nothing will be returned

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Origin</th>
<th>Maker</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Burgundy</td>
<td>CotesDOr</td>
<td>ClosDeVougeot</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>Riesling</td>
<td>NewZealand</td>
<td>Corbans</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Zinfandel</td>
<td>EdnaValley</td>
<td>Elyse</td>
<td>15</td>
</tr>
</tbody>
</table>
Query 2:

- **Which wine is a red wine?**
  - Answer: Zinfandel & Burgundy
  - Domain knowledge used:
    - Zinfandel is red;
    - Burgundy can be either red or white, but Burgundy from CotesDor is always red

- **We could issue:**

  ```sql
  SELECT ID
  FROM Wine
  WHERE hasColor = 'red';
  ```

  But “hasColor” is not a column in the table
Domain Knowledge from OWL Ontology

- Eg. Wine Ontology from the web ontology language OWL (W3C)
- Extract class hierarchies, (transitive) properties, implications, etc from OWL

Transitive Property `locatedIn`

- World
  - French
  - US
    - Bordeaux
    - Bourgogne
    - California
    - CotesDor
    - EdnaValley

Class Hierarchy

- Region
  - OWL:Thing
  - Wine
    - WineSugar
    - WineBody
    - WineColor
    - Winery
    - WineGrape
    - locatedIn
    - hasSugar
    - hasBody
    - hasColor
    - hasMaker
    - madeFromGrape

Implications

- \((\text{Type}=\text{CotesDor}) \rightarrow (\text{type}=\text{RedBurgundy}) \land (\text{origin}=\text{CotesDorRegion})\)
- \((\text{Type}=\text{Zinfandel}) \rightarrow (\text{hasColor}=\text{Red})\)
- \((\text{Type}=\text{Zinfandel}) \rightarrow (\text{hasSugar}=\text{dry})\)
- \((\text{Type}=\text{RedWine}) \rightarrow (\text{hasColor}=\text{Red})\)
Challenges

- How to incorporate domain knowledge (ontology) into a RDBMS?
- How to integrate relational data with domain knowledge?
- How to query relational data with meaning?
- How to process such queries?

Disclaimer
- Not re-inventing
  - Expert Systems
  - Datalog Systems
  - OWL/RDF & SparQL Systems

Put a little semantics into relational SQL systems
Overview of our solution

- Provide user with a unified view of the data and the domain knowledge.
- Through the virtual view, we offer a rich set of functionalities for knowledge inferencing out of the Spartan simplicity of SQL.
- Leverage hybrid relational-XML storage for managing domain knowledge
- Rewrite query on virtual view
- Leverage hybrid relational-XML query engine to process re-written query.
Virtual View Unifies Data & Ontology

- Users create virtual views over the relational data and the ontology
- Virtual columns/attributes not in original data
- Virtual columns not materialized -- inferred from the ontology

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
<th>origin</th>
<th>maker</th>
<th>price</th>
<th>locatedIn(origin)</th>
<th>hasColor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Burgundy</td>
<td>CotesDOr</td>
<td>ClosDeVougeot</td>
<td>30</td>
<td>{Bourgogne, France}</td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>Riesling</td>
<td>NewZealand</td>
<td>Corbans</td>
<td>20</td>
<td>{}</td>
<td>white</td>
</tr>
<tr>
<td>3</td>
<td>Zinfandel</td>
<td>EdnaValley</td>
<td>Elyse</td>
<td>15</td>
<td>{California, US}</td>
<td>red</td>
</tr>
</tbody>
</table>

Relational data

Ontology
The virtual view

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Origin</th>
<th>Maker</th>
<th>Price</th>
<th>LocatedIn</th>
<th>hasColor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Burgundy</td>
<td>CotesDOr</td>
<td>ClosDeVougeot</td>
<td>30</td>
<td>{Burgundy,France}</td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>Riesling</td>
<td>NewZealand</td>
<td>Corbans</td>
<td>20</td>
<td>{}</td>
<td>white</td>
</tr>
<tr>
<td>3</td>
<td>Zinfandel</td>
<td>EdnaValley</td>
<td>Elyse</td>
<td>15</td>
<td>{California,US}</td>
<td>red</td>
</tr>
</tbody>
</table>

- Wine Burgundy is originated from CotesDOr, which is a sub-region of Burgundy, which in turn, is a sub-region of France.
  - (type = Zinfandel) → (hasColor = red)
  - (type = Riesling) → (hasColor = white)
Creating the Virtual View

CREATE VIRTUAL VIEW WineView( Id, Type, Origin, Maker, Price, LocatedIn, HasColor) AS

SELECT W.*, O.locatedIn, O.hasColor, 
FROM Wine AS W, WineOntology AS O
WHERE O.type = W.type AND (O.type isA ‘Wine’) AND O.locatedIn = W.origin AND O.hasMaker = W.maker

Wine Ontology

Class Hierarchy

Wine

Wine Sugar

Wine Body

Wine Color

Winery

Wine Grape

Region

locatedIn

hasSugar

hasBody

hasColor

hasMaker

madeFromGrape

Burgundy

Riesling

DryRiesling

SweetRiesling

Wine Table

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
<th>origin</th>
<th>maker</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Burgundy</td>
<td>CotesDOr</td>
<td>ClosDeVougeot</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>Riesling</td>
<td>NewZealand</td>
<td>Corbans</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Zinfandel</td>
<td>EdnaValley</td>
<td>Elyse</td>
<td>15</td>
</tr>
</tbody>
</table>
Now we can write the semantic queries

- Which wine originates from the US?
  ```sql
  SELECT Id
  FROM WineView
  WHERE 'US' IN LocatedIn;
  ```

- Which wine is a red wine?
  ```sql
  SELECT Id
  FROM WineView
  WHERE hasColor = 'red';
  ```

<table>
<thead>
<tr>
<th>id</th>
<th>type</th>
<th>origin</th>
<th>maker</th>
<th>price</th>
<th>locatedIn(origin)</th>
<th>hasColor</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Burgundy</td>
<td>CotesDOr</td>
<td>ClosDeVougeot</td>
<td>30</td>
<td>{Bourgogne, French}</td>
<td>red</td>
</tr>
<tr>
<td>2</td>
<td>Riesling</td>
<td>NewZealand</td>
<td>Corbans</td>
<td>20</td>
<td>{}</td>
<td>white</td>
</tr>
<tr>
<td>3</td>
<td>Zinfandel</td>
<td>EdnaValley</td>
<td>Elyse</td>
<td>15</td>
<td>{California, US}</td>
<td>red</td>
</tr>
</tbody>
</table>
Physical Level Support

- Leverage Hybrid relational-XML DBMSs for storing domain knowledge
  - indices for XML
  - a hybrid query compiler supports XQuery and SQL/X

![Ontology Repository Diagram]
Hybrid Relational-XML DBMS

- CREATE TABLE **ClassHierarchy** (id INTEGER, name VARCHAR(27), **hierarchy** XML);
- INSERT INTO **ClassHierarchy** VALUES (1, ’Wine’,

XMLParse(’<?xml version=’1.0’>
    <wine>
        <WhiteWine><WhiteBurgundy>...<WhiteBurgundy>...
            <WhiteWine>
                <DessertWine><SweetRiesling/>...
                    <DessertWine>...
                </DessertWine>...
            </WhiteWine>
        </WhiteWine>
    </wine>’)

);

- Example: find class ids and class names of all class hierarchies that contain the XPath /Wine/DessertWine/SweetRiesling:

```sql
SELECT id, name
FROM **ClassHierarchy** AS C
WHERE XMLExists(’$/Wine/DessertWine/SweetRiesling’
    PASSING BY REF C.order AS ”t”)
```
Query Re-writing

- Query expansion on virtual columns using implications.

```
SELECT V.Id
FROM WineView AS V
WHERE V.hasColor=White;
```

- Since the following implications exists, we use them to expand the query predicate

```
(Type=WhiteWine) → (hasColor=white)

(Type=Riesling) → (hasColor=white)
```

```
SELECT V.Id
FROM WineView AS V
WHERE US ∈ V.locatedIn;
```

- Subsumption checking via XPath & XMLExists SQL/XML function

```
SELECT W.Id
FROM Wine AS W, TransitiveProperty AS T
WHERE T.ontID='wine'
AND T.propID='locatedIn'
AND XMLExists(T.tree//USRegion//W.origin);
```

- Since locatedIn is a virtual column on the transitive closure of W.origin, we rewrite the query to
But the expansion is not that simple

```
SELECT ID FROM V WHERE (A=v1)
```

Virtual View $V$

```
<table>
<thead>
<tr>
<th>ID</th>
<th>B</th>
<th>D</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
SELECT ID FROM V WHERE B=v2 AND D=v4
```

Implication Graph

For details see the Algorithm in the paper

Transitive Tree for property $C$
Experiments

- Investigate time to rewrite the queries on virtual views

Data Generation
- trees for transitive properties parametrized by
  - Number of nodes
  - Maximum fanout
- graphs for implications parametrized by
  - Number of relationships
  - Number of values
  - Number of levels in the graph
  - Density: number of rules between two consecutive levels
  - Fanout: number of atoms in a rule body

Measurement: rewriting time averaged over 5 randomly generated data sets.

Performance for baseline rewriting algorithm and optimized rewriting algorithm (using memoization)
Implication Graph Density

- Number of rules did not affect rewriting performance as much as density of the implication rule graph.
- Baseline algorithm is not scalable. Memoization is much better.
- In general, the rewriting time is reasonable ( < 0.5 s)
Size of transitive property trees

- Rewriting time scales linearly with size of trees.
Conclusion

- Framework for **putting a little semantics** into relational SQL systems.
- Users register ontologies in DBMS and links them with relational data by creating **virtual views**
- Virtual columns in the virtual views are not materialized
- Queries on the virtual columns are rewritten to predicates on base table columns.
- Future work: performance issues
Questions
Implication Graph

- A=v1 ← G=v7
- A=v1 ← B=v2 ^ C=v3
- B=v2 ← H=v8
- C=v5 ← D=v4
- C=v5 ← F=v6

\[ A=v1 \rightarrow G=v7 \]
\[ A=v1 \rightarrow B=v2 ^ C=v3 \]
\[ B=v2 \rightarrow H=v8 \]
\[ C=v5 \rightarrow D=v4 \]
\[ C=v5 \rightarrow F=v6 \]

\[ A=v1 \quad : \quad \text{Clause (e.g., x.hasBody=Medium)} \]
\[ ^\wedge \quad : \quad \text{Operator (e.g., AND)} \]