# Ultrawrap: Using SQL Views for RDB2RDF

Juan F. Sequeda[1], Conor Cunningham[2], Rudy Depena[1]
and Daniel P. Miranker[1]
[1]Department of Computer Sciences - The University of Texas at Austin
[2]Microsoft Corporation

# Goals

- ✓ Automatically create SPARQL endpoint for legacy relational databases.
- ✓ Real-time consistency between the relational and RDF data
- ✓ Making maximal use of existing SQL infrastructure
- ➢ **Research question:** Do existing commercial SQL query engines already subsume all the algorithms needed to support effective SPARQL execution on relational data?

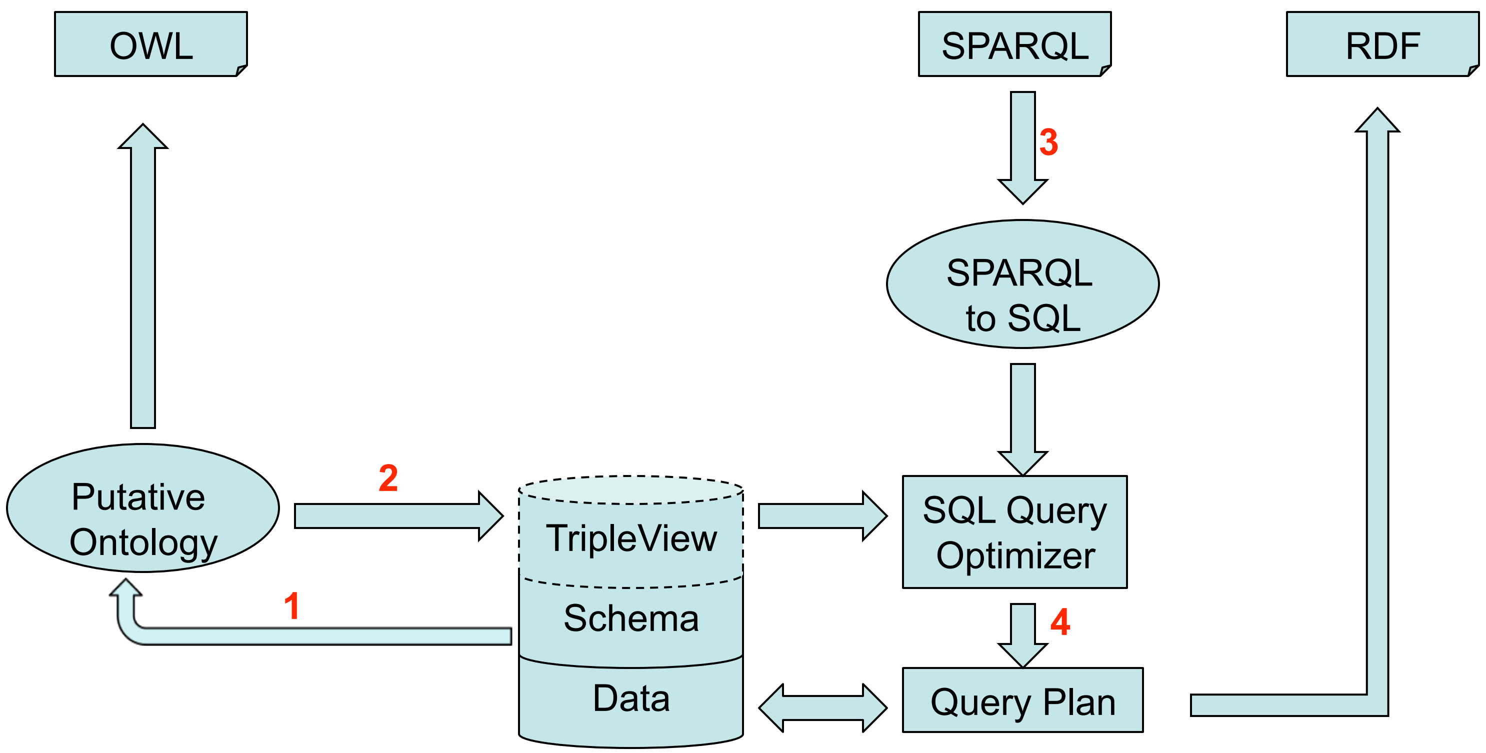
THE UNIVERSITY OF TEXAS AT AUSTIN

# Ultrawrap Architecture

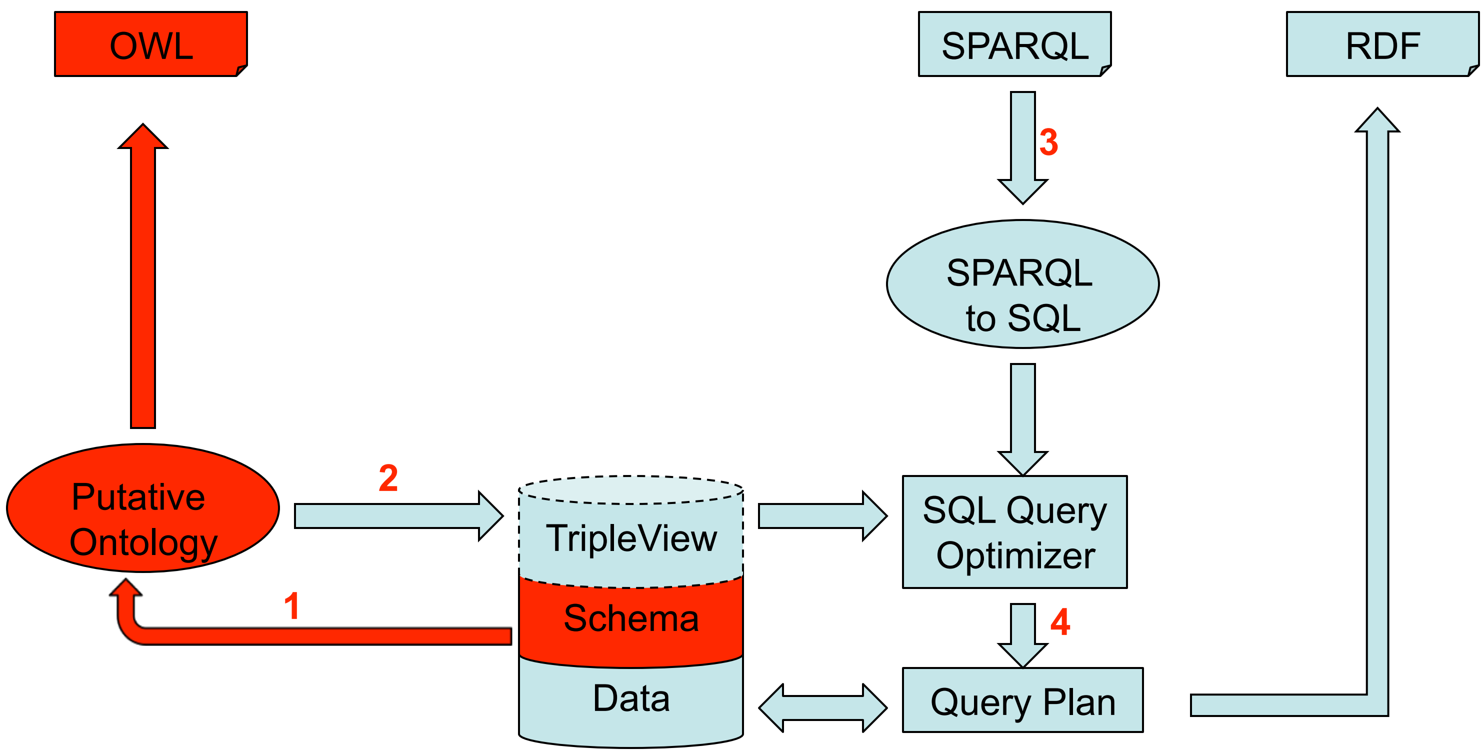
- Compile Time
  1. Create Putative Ontology (PO)
  2. Create Virtual Triple Store
- Run Time
  3. Naïve SPARQL to SQL translation
  4. SQL Optimizer is the rewriter
- Future
  5. Putative Ontology to Domain Ontology mapping

THE UNIVERSITY OF TEXAS AT AUSTIN

# Ultrawrap Architecture



# Step 1: Creating a Putative Ontology



# Ontology Quality? I.e. putative

- Putative: "commonly regarded as such"

- Putative Ontology (PO): automatic syntactic transformation from a data source schema to an ontology
  - data or information source ontology

- Evidence: SQL schema from E-R models make "interesting" ontologies

THE UNIVERSITY OF TEXAS AT AUSTIN

# Formal Results

FOL rules transform SQL DDL to OWL

- Full mapping in Datalog
  - Stratified and safe
- Proof of total coverage of all key combinations

## Documentation

- Tirmizi, S. H., Sequeda, J.F., and Miranker, D.P. **Translating SQL Applications to the Semantic Web**. In *Proceedings of the 19th international Conference on Database and Expert Systems Applications (DEXA2008).*

- Sequeda, J.F. (1), Tirmizi , S.H.(1), Corcho, O. (2), Miranker, D.P. (1). **"Direct Mapping SQL Databases to the Semantic Web: A Survey"**. The University of Texas at Austin, Department of Computer Sciences(1), Universidad Politecnica de Madrid(2). Report# TR-09-04 (regular tech report). January 19th, 2009. 35 pages. ftp://ftp.cs.utexas.edu/pub/techreports/tr09-04.pdf

THE UNIVERSITY OF TEXAS AT AUSTIN

# Example From the Transformation System

Student

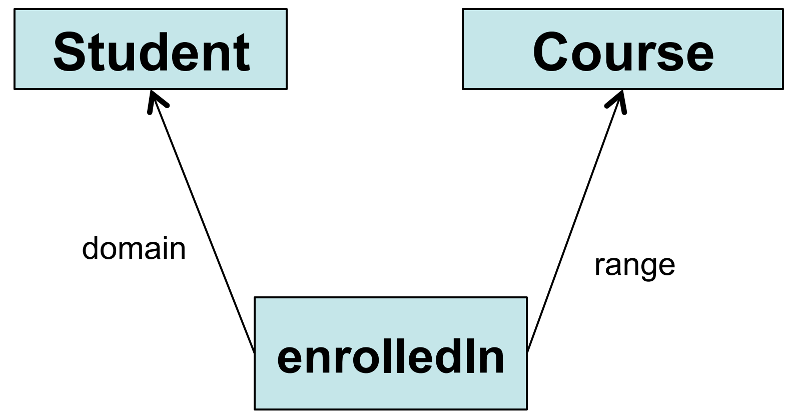
| SID | Name |
|---|---|
| 1 | Juan |
| 2 | Hamid |

Course

| CID | Name |
|---|---|
| 1 | CS386 |
| 2 | CS379H |

enrolled

| Student | Course |
|---|---|
| 1 | 2 |
| 2 | 1 |



# Transformation System

$BinRel(r,s,t) \leftarrow Rel(r) \wedge FK(xtr,r,\_,t) \wedge FK(xsr,r,\_,s) \wedge xtr \neq xsr \wedge Attr(y,r) \wedge \neg NonFK(y,r) \wedge FK(z,r,\_,u) \wedge fkey(z,r,u) \in \{fkey(xsr,r,s), fkey(xtr,r,t)\}$

$Class(r) \leftarrow Rel(r) \wedge \neg BinRel(r,\_,\_)$

```
create table STUDENT{
SID integer primary key,
NAME varchar not null }
```

→

```
<owl:Class rdf:ID="Student"/>
```

```
create table COURSE{
CID integer primary key,
NAME varchar not null }
```

→

```
<owl:Class rdf:ID="Course"/>
```

$ObjP(r,s,t) \leftarrow BinRel(r,s,t) \wedge Rel(s) \wedge Rel(t) \wedge \neg BinRel(s,\_,\_) \wedge \neg BinRel(t,\_,\_)$

```
create table ENROLLED{
  Student integer foreign key
references STUDENT(SID),
  Course integer foreign key
references COURSE(CID),
  constraint REG_PK primary key
(Student, Course)}
```

→

```
<owl:ObjectProperty rdf:ID="enrolledIn">
 <rdfs:domain rdf:resource="#Student"/>
 <rdfs:range rdf:resource="#Course"/>
</owl:ObjectProperty>
```

THE UNIVERSITY OF TEXAS AT AUSTIN

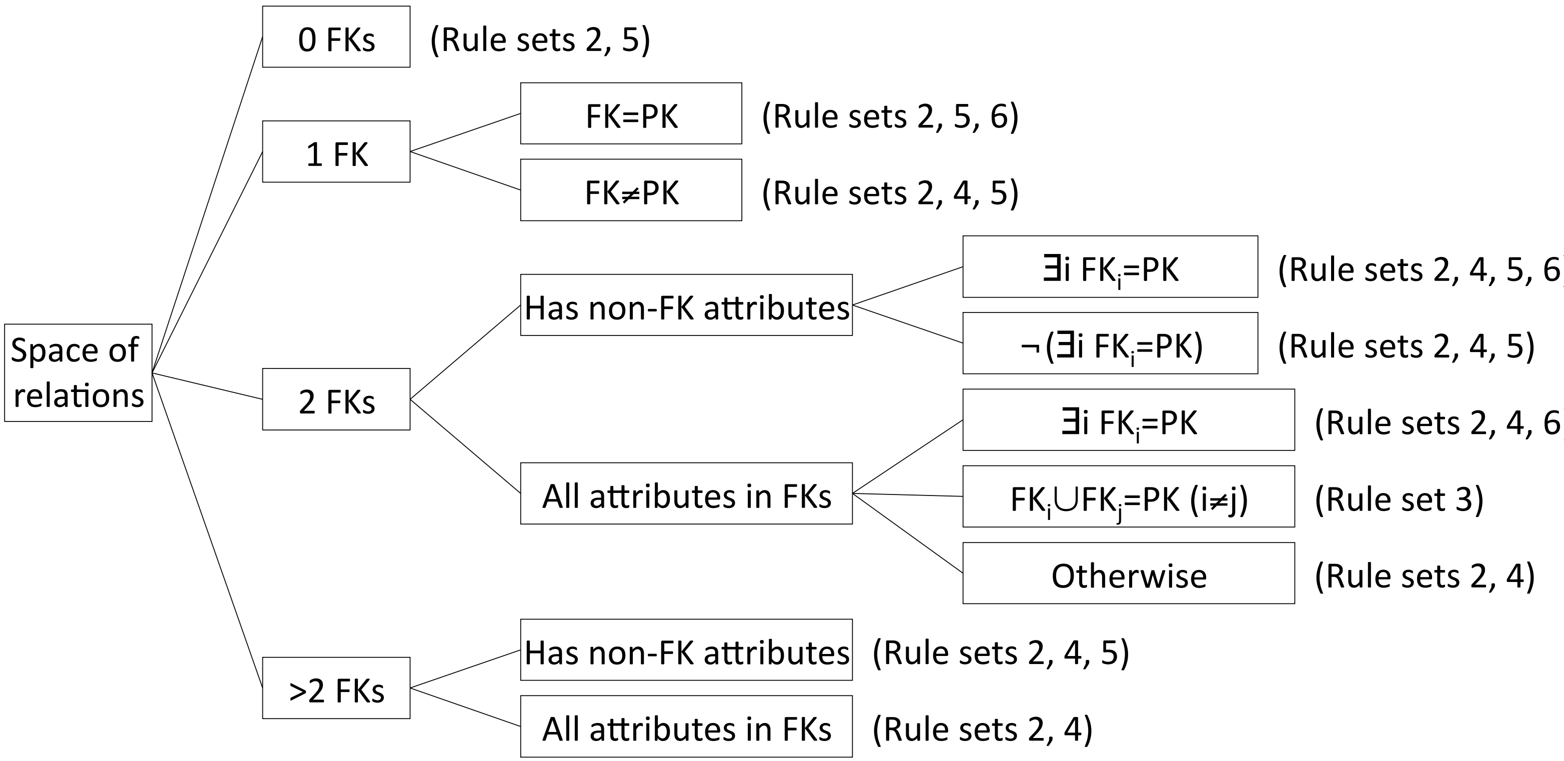
## Proof of total coverage of all key combinations

- PK: a relation only has a Primary Key
- C-PK: a relation only has a composite Primary Key
- S-FK: a relation only has one Foreign Key
- N-FK: a relation has at least two or more Foreign Keys

Grammar

- E → PK + T | C-PK +T
- E → S-FK
- E → N-FK
- T → S-FK | N-FK

# LR(0) Item set construction represents all possibilities




**The tree describes the complete space of relations when all possible combinations of primary and foreign keys are considered.**

THE UNIVERSITY OF TEXAS AT AUSTIN

# All Key Combinations Enumerated

- PK + S-FK: a relation has a Primary Key and only one Foreign Key
  - PK = S-FK: the Foreign Key is the Primary Key
  - PK ∩ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes
- PK + N-FK: a relation has a Primary Key and two (2) Foreign Keys
  - PK ∩ N-FK = 0: the Foreign Key and the Primary Key do not share any attributes
  - PK ⊂ N-FK: one of the Foreign Keys is also the Primary Key
- PK + N-FK: a relation has a Primary Key and more than two (> 2) Foreign Keys
  - PK ∩ N-FK = 0: the Foreign Key and the Primary Key do not share any attributes
  - PK ⊂ N-FK: one of the Foreign Keys is also the Primary Key
- C-PK + S-FK: a relation has a Composite Primary Key and only one Foreign Key.
  - C-PK ∩ S-FK = 0: the Foreign Key and the Primary Key do not share any attributes
  - S-FK ⊂ C-PK: the Foreign Key is part of the Primary Key
- C-PK + N-FK: a relation has a Composite Primary Key and two (2) Foreign Keys
  - C-PK ∩ N-FK = 0: all the Foreign Keys and the Primary Key do not share any attributes
  - N-FK ⊆ C-PK: all the Foreign Keys are part of the Primary Key
  - C-PK ∩ N-FK ≠ 0, C-PK – N-FK ≠ 0, N-FK – C-PK ≠ 0: The Foreign Keys and Primary Key share common attributes
- C-PK + N-FK: a relation has a Composite Primary Key and more than two (> 2) Foreign Keys
  - C-PK ∩ N-FK = 0: all the Foreign Keys and the Primary Key do not share any attributes
  - N-FK ⊆ C-PK: all the Foreign Keys are part of the Primary Key
  - C-PK ∩ N-FK ≠ 0, C-PK – N-FK ≠ 0, N-FK – C-PK ≠ 0: The Foreign Keys and Primary Key share common attributes

THE UNIVERSITY OF TEXAS AT AUSTIN

# Ultrawrap creates PO automatically

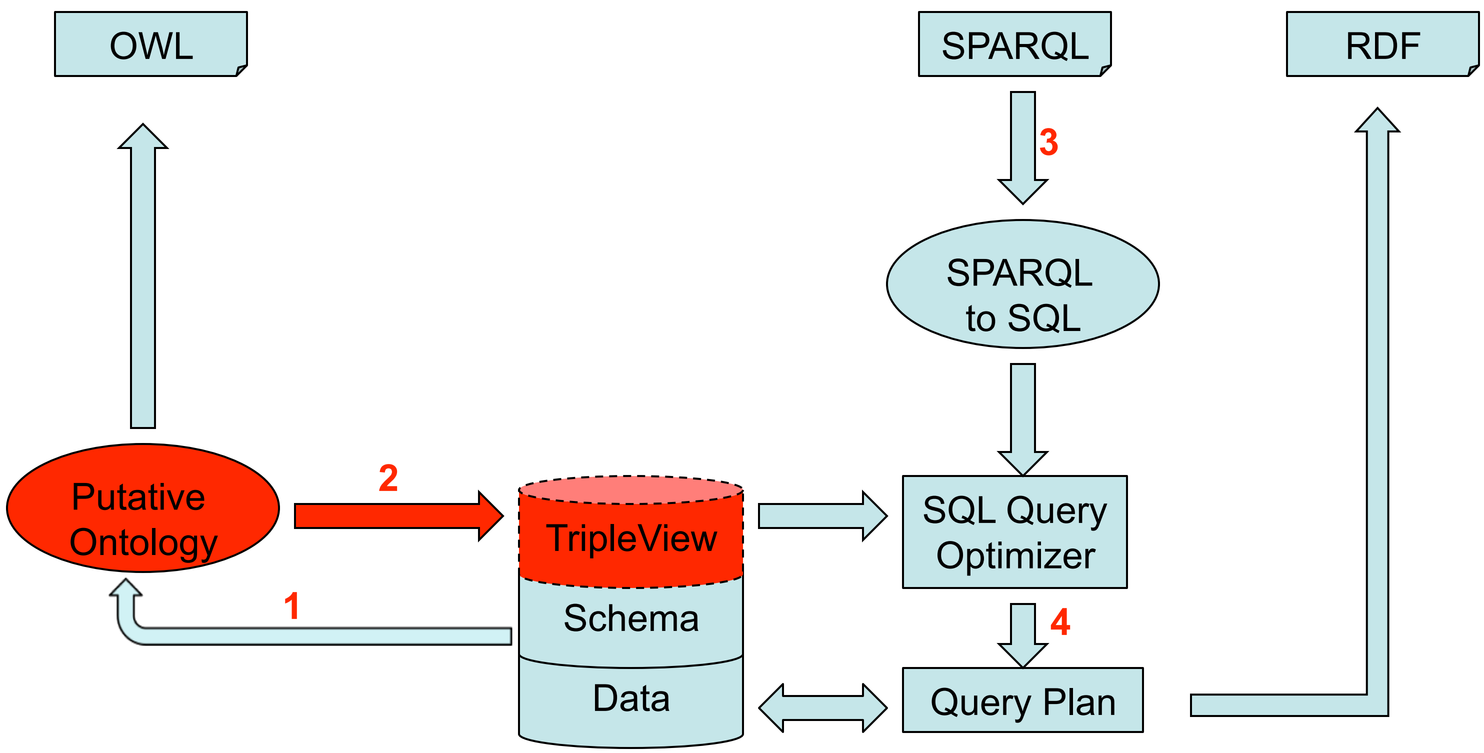
- Reads the Data Dictionary
  - Specific to each vendor
- Currently supporting Microsoft SQL Server and MySQL

# Quality of a Putative Ontology

- Quality of the PO depends on the SQL DDL
  - Is it normalized?
  - Are all the constraints explicit?
- If the Quality is sufficient, all we need to do is rename
- Need to map the Putative Ontology to a Domain Ontology
- Evidence: SQL schema from E-R models make “interesting” ontologies
- SQL schemas made without any previous modeling make “poor” ontologies

THE UNIVERSITY OF TEXAS AT AUSTIN

# Step 2. Create Virtual Triple Store



## Step 2. Create Virtual Triple Store

- Represent all relational data as triples using a view definition
  - Promise of avoiding self joins (optimizer will do this)
- Triple table approach: one table with three columns (s,p,o)
  - No symbol/lookup table. Strings are in the view
- Actually, the view is (s,spk, p, o, opk) where spk and opk are the index values
  - Optimizer needs to know the index values

## Step 2. Create Virtual Triple Store

- Create SELECT statements that output triples

```
SELECT "Product"+id as s, id as spk, "product_label" as p, label as o, null
    as opk FROM Product
```

| S | SPK | P | O | OPK |
|---|---|---|---|---|
| Product1 | 1 | product_label | Label of Product 1 | null |
| Product2 | 2 | product_label | Label of Product 2 | null |

```
SELECT "Product"+ProductID as s, ProductID as spk, "product_productfeature"
    as p, "ProductFeature"+ProductFeatureID as o, ProductFeatureID as opk
    FROM ProductFeatureProduct
```

| S | SPK | P | O | OPK |
|---|---|---|---|---|
| Product1 | 1 | product_productfeature | ProductFeature45 | 45 |
| Product1 | 1 | product_productfeature | ProductFeature98 | 98 |

- Use the PO as basis to create all the SELECT statements

# Step 2. Create Virtual Triple Store

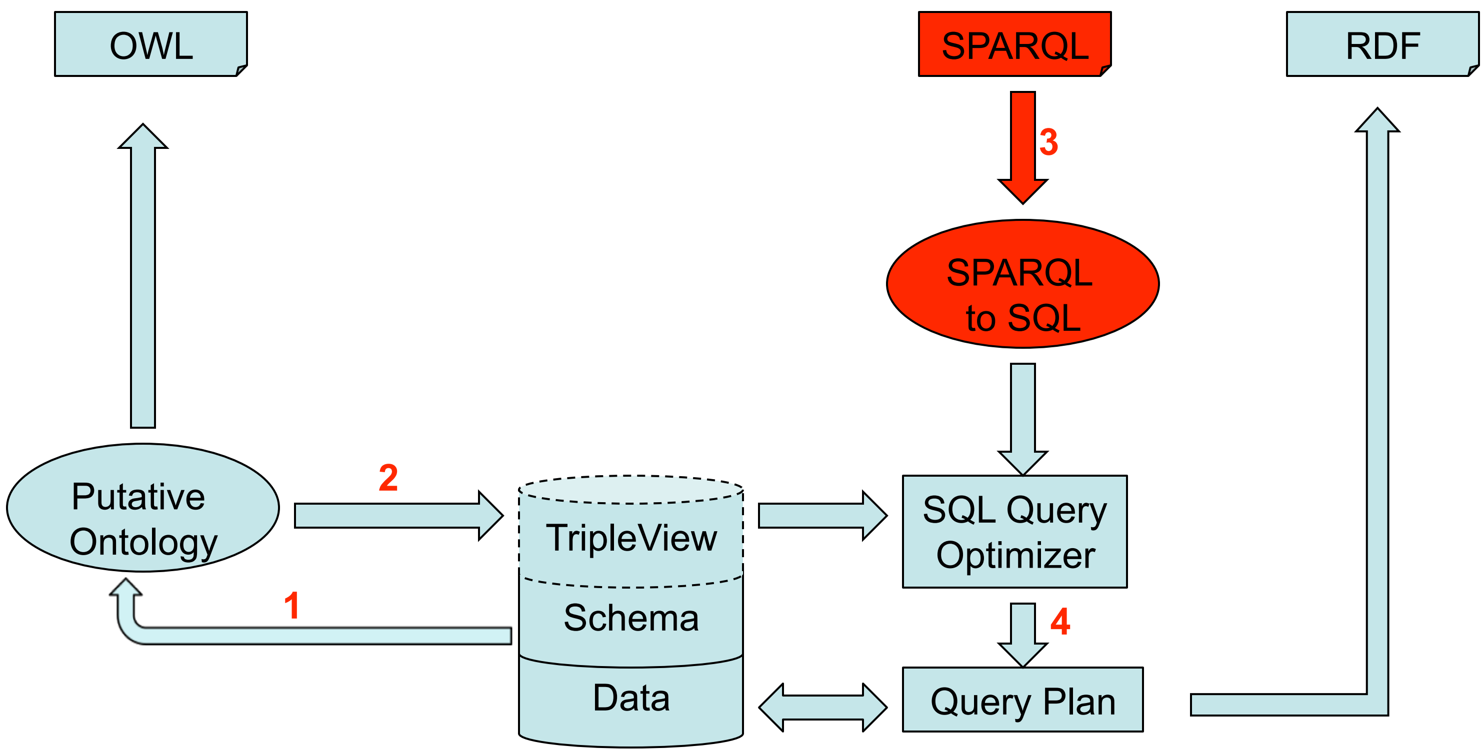
- Triple View is a union of all the SELECT statementss

```
CREATE VIEW TripleView(s,spk, p, o, opk) as
SELECT “Product”+id as s, id as spk, “rdf:type” as p, “Product” as o, null
   as opk FROM Product
UNION
SELECT “Product”+id as s, id as spk, “label” as p, label as o, null as opk
   FROM Product
UNION
SELECT “Product”+ProductID as s, ProductID as spk, “product_productfeature”
   as p, “ProductFeature”+ProductFeatureID as o, ProductFeatureID as opk
   FROM ProductFeatureProduct
UNION …
```

- BSBM generates ~80 select statements in order to represent all relational data as triples

THE UNIVERSITY OF TEXAS AT AUSTIN

# Step 3: Naïve SPARQL to SQL Translation



# Step 3: Naïve SPARQL to SQL Translation

## SPARQL Query

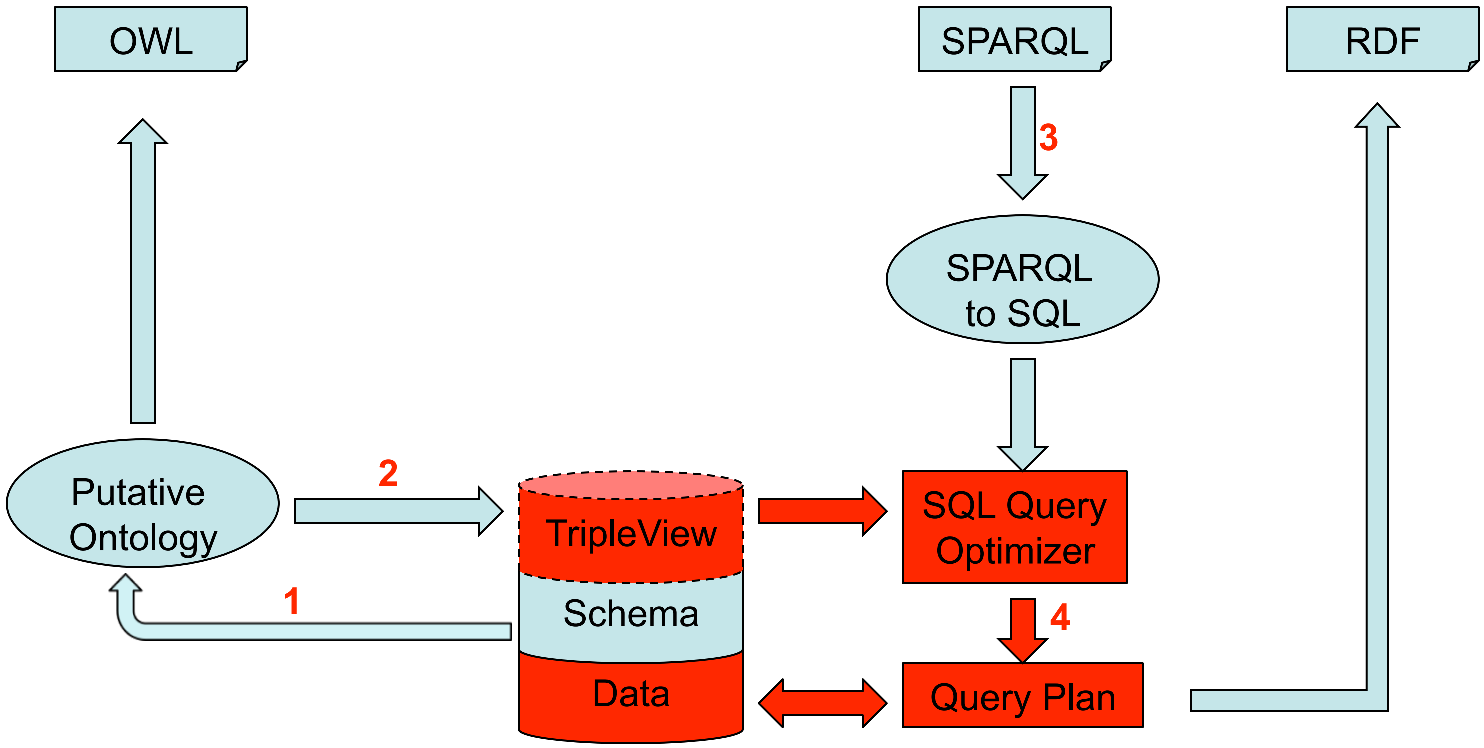
```
SELECT ?product ?label
WHERE{
?product producttype_product
   ProductType47.
?product product_label ?label.
?product product_productfeature
   ProductFeature76.
?product product_productfeature
   ProductFeature4242.
?product product_propertyNum1 ?v.
FILTER (?v >500)
}
```

## SQL Query on the Triple View

```
SELECT t1.o as product, t2.o as
   label
FROM TripleView t1, t2, t3, t4, t5
WHERE
    t1.p = ‘producttype_product’
and t1.opk = 47
and t2.p = ‘product_label’
and t3.spk = t1.spk
and t3.p = ‘product_productfeature’
and t3.opk = 76
and t4.spk = t1.spk
and t4.p = ‘product_productfeature’
and t4.opk = 4242
and t5.spk = t1.spk
and t5.p = ‘product_propertyNum1’
and t5.o > 500
```

Syntactic transformation from a SPARQL query to an equivalent SQL query on the Triple View

# Step 4: SQL Query Optimizer is the Rewrite system




THE UNIVERSITY OF TEXAS AT AUSTIN

# Step 4: SQL Query Optimizer is the Rewrite system

```
TripleView(1, label, ABC) :- Product(1,ABC, _, _)
TripleView(1, propNum1, 1) :- Product(1,_, 1, _)
TripleView(1, propNum1, 2) :- Product(1,_, _, 2)
```

SQL Query on the TripleView

```
Query(X, Y):-TripleView(X, label, Y),
TripleView(X, propNum1, 1),
TripleView(X, propNum2, 2)
```

SQL Query on the Relational Data

```
SELECT id, label FROM product
WHERE propNum1 = 1 and propNum2 = 2
Query(X, Y) :- Product(X, Y, 1, 2)
```

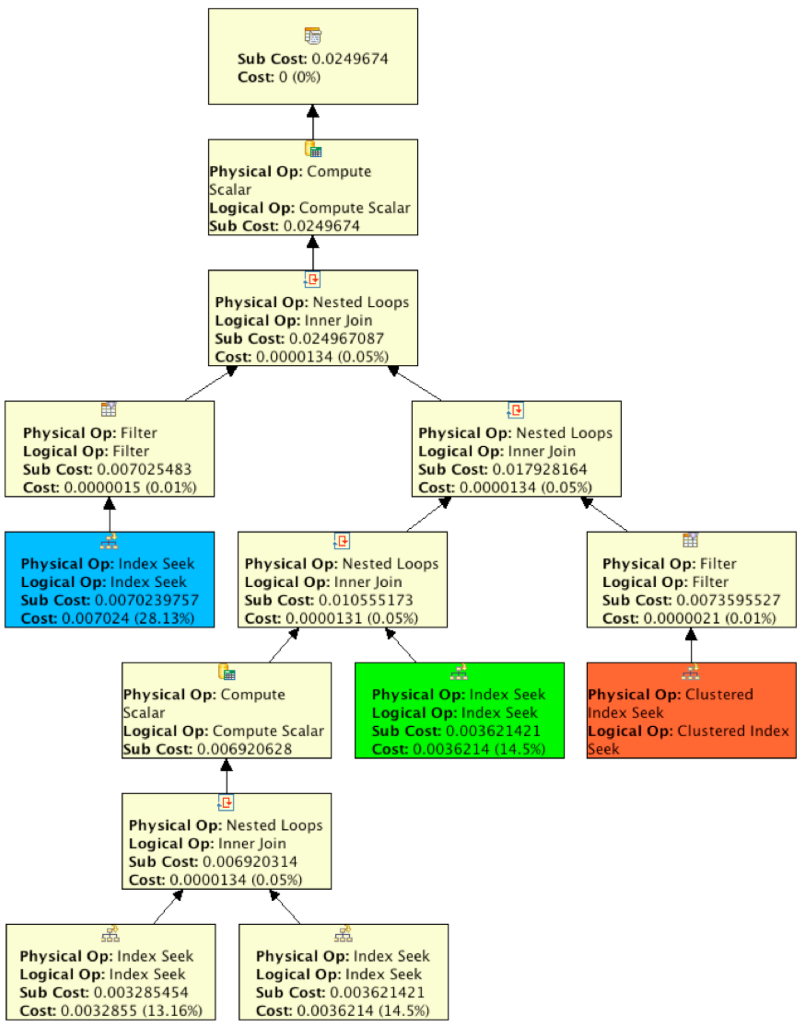
Evaluate SQL Query on the TripleView

```
Query(X,Y):-Product(X,Y,1,_),Product(X,Y,_, 2)
Query(X,Y):- Product(X, Y, 1, 2)
```

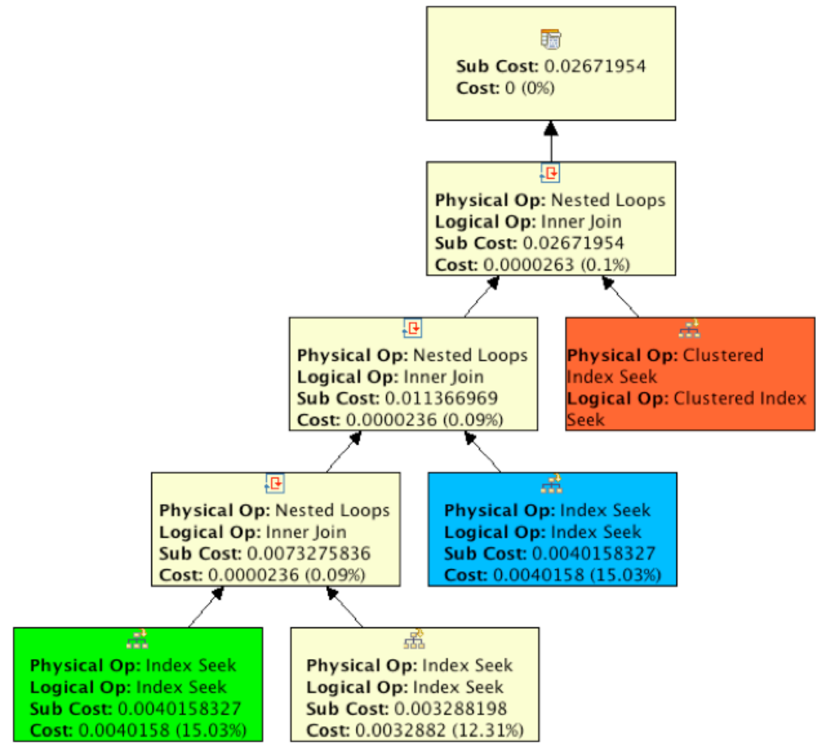
THE UNIVERSITY OF TEXAS AT AUSTIN

# Step 4: SQL Query Optimizer is the Rewrite system

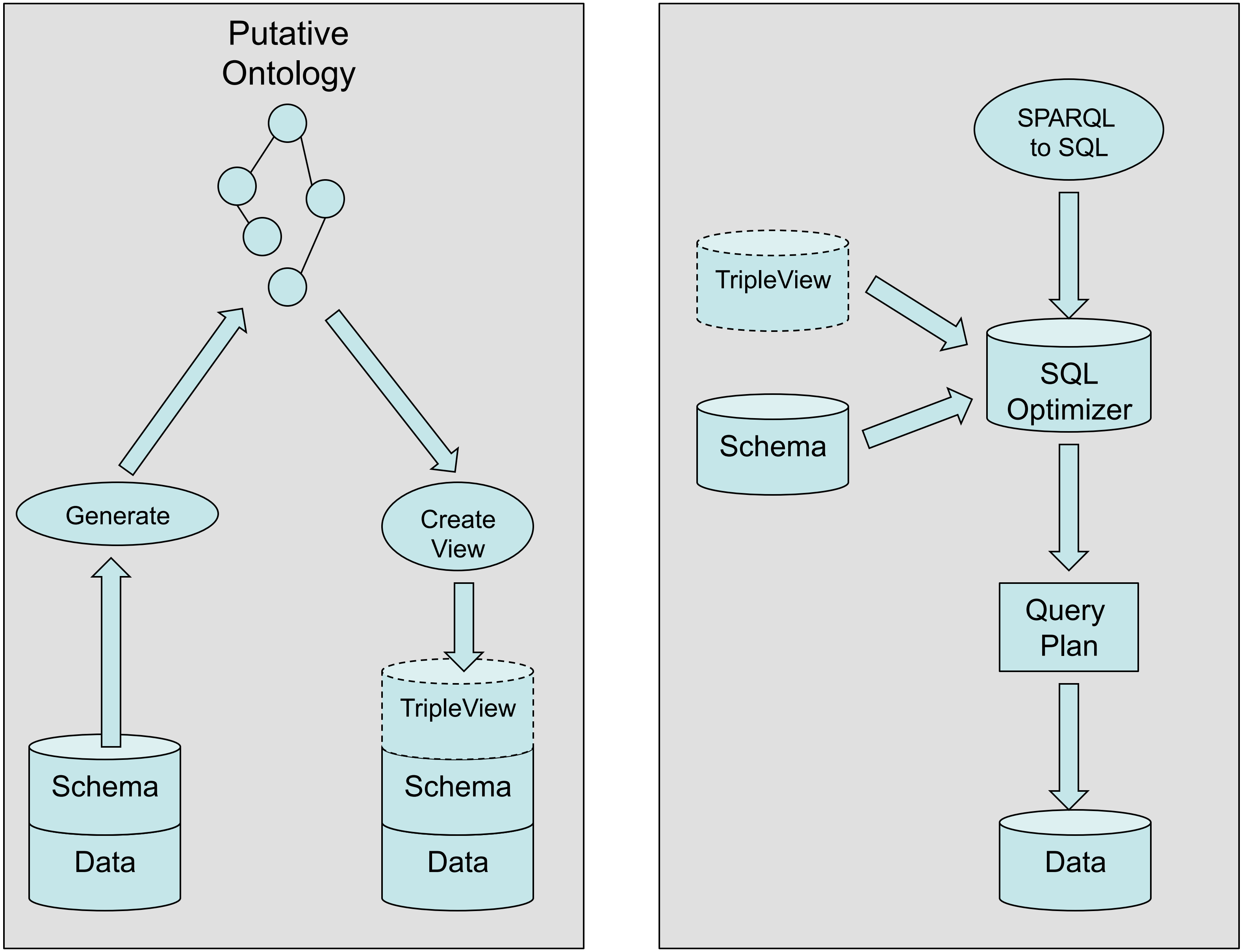
- TripleView Plan




- Optimal Plan



# Ultrawrap Architecture Summary




Compile Time

Run Time

# Current Implementation

- Running on Microsoft SQL Server
- Jess Rule Engine
- Initial test on BSBM on 1 million triples, execution time is close to running time of native SQL queries on relational data

| **Query** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Jena SDB | 4.41 | 4.33 | 6.27 | 7.12 | 12.36 | 1.45 | 11.94 | 6.69 | 8.38 | 5.39 | 2.76 | 4.34 |
| Sesame | 2.49 | **0.86** | 3.52 | 3.78 | 7.31 | 1.76 | 15.51 | 3.02 | 1.17 | 3.63 | 1.49 | **0.65** |
| Virtuoso RDF Views | 8.29 | 2.77 | 9.79 | 16.13 | 1.89 | **0.09** | 16.59 | 6.83 | 2.14 | 6.96 | 9.50 | 3.44 |
| D2R Server | 5.03 | 5.28 | 7.93 | 7.63 | 222.73 | 0.94 | 10.96 | 12.46 | 13.37 | 7.16 | 30.61 | 2.55 |
| Ultrawrap | **1** | 1.00 | **1.00** | **1.00** | **1.00** | 1.00 | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | 1.00 |
| Native SQL | 0.94 | 0.67 | 0.90 | 0.80 | 1.09 | 0.62 | 0.67 | 0.72 | 1.03 | 1.02 | 0.94 | 0.30 |

THE UNIVERSITY OF TEXAS AT AUSTIN

## Current Work

- Shifted problem to ontology-to-ontology mapping
  - Version 0: query only the Putative Ontology
  - Version 1: Manually mapping layer between Domain Ontology to Putative Ontology
  - Version 2: Automatic identify mappings
- Testing on Oracle, PostgreSQL, Virtuoso
- Road Map
  - Dec 2009: Version 0
  - Feb 2010: Version 1 running on other RDBMS

THE UNIVERSITY OF TEXAS AT AUSTIN

# Thank You


THE UNIVERSITY OF
TEXAS
AT AUSTIN

Powered by The Miranker Lab

**R**esearch
**I**n
**B**ioinformatics and the
**S**emantic Web