基于语义视图的 SPARQL-SQL 查询转换方法①

李华昱, 龚 安

(中国石油大学(华东) 计算机与通信工程学院, 青岛 266580)

摘 要: 针对油气井工程领域关系数据库, 提出一种基于语义视图的 SPARQL-SQL 查询转换方法. 该方法采用特定本体描述数据源关系模式, 通过RDF三元组形式, 将关系表定义为领域本体之上的语义查询视图, 从而建立数据源与本体之间的映射关系, 并根据语义映射信息, 将提交的 SPARQL 语句进行解析与查询重写, 转换为面向关系数据源的SQL语句. 通过实现油气井虚拟数据中心, 验证了该方法的可行性与有效性, 并获得了良好的应用效果.

关键词: 语义视图; SPARQL; 语义集成; 查询重写; 油气井

SPARQL-SQL Query Transformation Based Semantic Views

LI Hua-Yu, GONG An

(College of Computer and Communication Engineering, China University of Petroleum, Qingdao 266580, China)

Abstract: For relational database of oil-gas well engineering domain, this paper presented a method of SPARQL-SQL query transformation based semantic views. In this method, a specific ontology is used to describe schema of relational database and tables are defined as semantic query views based on domain ontology. Through semantic mapping provided by semantic views, SPARQL query is parsed and rewrited to SQL statements on relational data sources. A virtual data center system is implemented with which feasibility and validity of methods provided in this paper are tested, and we achieved a good application effect.

Key words: semantic views; SPARQL; query rewriting; data integration; oil-gas well

本体能够清晰描述领域概念模型和数据语义信息, 利用本体对数据库进行集成可有效解决由术语定义、概念结构和相互关系的差异所造成的语义异构. 目前, 领域数据主要还是以关系数据库进行存储, 将基于本体的语义查询转换为面向关系数据库的 SQL 查询, 充分利用关系数据库函数, 是提高语义集成与查询效率的一种重要尝试. 因此, 实现 SPARQL-SQL 查询转换是实现和推广领域数据语义集成与应用的一项必要措施.

文献[1]采用(Subject, Predicate, Object)关系模型将 RDF 存储在关系数据库中, 并定义 Select、Projection、Join 和 Union 等关系运算用以表示 SPARQL 的 Select 和 Where 子句, 能够将 SPARQL 转换为对关系数据库中 RDF 数据的 SQL 查询. Hartig[2]等人提出了 SPARQL 查询图模型, 基于此模型定义了一组指导查询重写的转换规则. Cheboto[3,4]等人提出的查询转换方法包括 BGPtoSQL 和 SPARQL-SQL 算法: BGPtoSQL 首先将 SPARQL 查询表示为图形模型, 然后再将其转换为等价的 SQL 关系代数; SPARQL-SQL 则是在 BGPtoSQL 基础上完成语义查询转换. Harris[5]等人首先通过定义 Triples、Symbols、Datatypes 和 Languages 4 个关系表对 RDF 3 元组数据进行存储, 然后构造关系代数运算将SPARQL查询转换为等价的SQL查询语句. 文献[6]方法中, 利用关系数据库分别将 RDF 数据和元数据保存在关系数据表中, 通过构造查询操作树并进行关系代数操作将 SPARQL 转换为相应的 SQL

① 基金项目:中央高校基本科研业务费专项资金(14CX02030A)
收稿时间:2015-06-01;收到修改稿时间:2015-07-06

表达式.

上述语义集成和查询转换方法是为解决大规模 RDF 数据存储问题而提出的, 都假设关系数据库采用三元组表或者垂直表结构[7], 但是领域数据主要采用 ER 模型进行设计与实现, 并且数据规模庞大, 如果按照以上方法进行语义集成, 必须进行两种关系模型之间的转换, 特别在数据发生更新时, 转换频率会随之提高, 极大降低系统运行效率, 因此, 以上方法不适合领域数据语义集成.

中医药、电力、材料、石油等领域涉及大量的关系数据, SPARQL-SQL 查询转换方法影响语义集成效率的关键因素[8,9]. 针对石油工程领域异构数据集成问题, 本文提出一种基于语义视图的 SPARQL-SQL 查询转换方法. 该方法基于领域本体模型对提交的 SPARQL 语句进行解析; 然后, 根据语义视图中定义的映射信息将查询中的概念和属性转换为底层数据源中的关系表及属性, 从而能够在不加载和转换数据情况下, 完成 SPARQL 到底层关系数据源的 SQL 查询转换.

1 实现架构

系统采用油气井工程本体 WeSM(Well Semantic Model)作为语义数据模型, 通过语义映射, 将数据源中的关系表映射为定义在 WeSM 之上的语义视图 SV(Semantic View); 用户利用 SPARQL 查询构造器, 通过可视化形式设置语义查询需求, 并将其转换为 SPARQL 语句提交给查询转换部件, 进行 SPARQL-SQL 的查询重写; SQL 查询处理模块将转换后的 SQL 语句提交给数据源完成数据查询, 并将查询结果处理后展现给用户并完成语义查询, 系统架构如图 1 所示.

SPARQL-SQL 查询转换实现流程主要包括以下步骤:

(1)读取数据源模式

分别读取 n 个数据源 $S1,S2,...,Sn$ 的模式信息, 包括: 数据源连接字符串、数据表、属性、数据类型和约束等, 并将其以 OWL 实例形式存储在数据源描述本体 DSOnto(DataSource Ontology)中.

(2)建立语义视图

在每个数据源 D_i 与 WeSM 本体之间建立映射, 将 D_i 中每个关系表映射为 WeSM 上的语义视图, 记为 SV(Semantic View), SV 由一组 RDF 三元组构成, 关系表每一个属性被定义为 WeSM 上的一个查询路径.

(3)语义查询构建

基于 WeSM 语义模型, 通过可视化语义查询构造器将语义查询构造为符合规范的 SPARQL 语句 *Query*.

(4)SPARQL-SQL 查询转换

对 *Query* 进行解析, 通过语义视图并利用查询重写算法, 将其转换为面向数据源的子查询 $Q_{sql1},...,...Q_{sqln}$, 获取查询结果并进行处理后, 形成全局查询结果返回给用户.

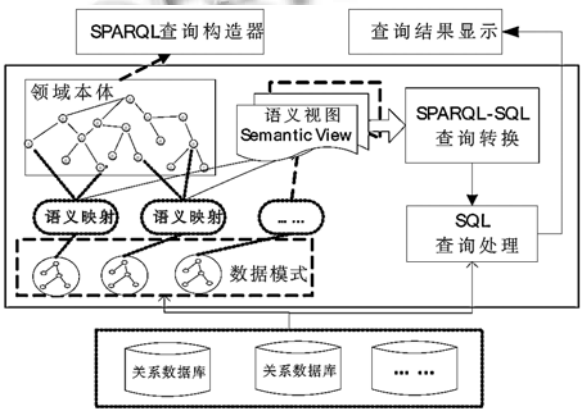

图 1 SPARQL-SQL 查询转换实现架构

2 语义映射

建立语义视图 SV, 首先需要描述数据源关系模式, 然后, 分别读取关系模式与 WeSM 语义信息, 通过建立映射关系, 实现 SVT 进行定义与表示.

2.1 数据源模式表示

基于 Relational.OWL[10]实现思想, 采用数据源描述本体 DSOnto 对关系数据源模式信息进行描述. DSOnto 主要对数据库、表、属性和对象之间的关系进行描述, 主要包括:

(1)定义描述数据模式的类: *dsonto:Database, dsonto:Table, dsonto:Column, dsonto:PK*, *dsonto:FK* 分别表示数据库、表、属性、主键和外键 5 个概念.

(2)定义描述类之间的关联关系的属性: *dsonto:hasTable, dsonto:hasColumn, dsonto:hasPK, dsonto:hasFK*, 定义数据库、表、属性、主键和外键之间的关联; *dsonto:belongtoDatabase, dsonto:belongtoTable* 等提供反向语义关联; dsonto:ref 描述外键引用的属性.

(3)定义类的属性: *dsonto:isNullable,*

dsonto:columnType 等属性描述类 *dsonto:Column* 是否为空和数据类型信息.

在利用 DSOnto 描述数据源模式时, 需要抽取数据库元数据作为实例, 然后将其填充到 DSOnto 本体的 ABox 实例数据中, 图 2 展示了 DSOnto 的部分层次结构.

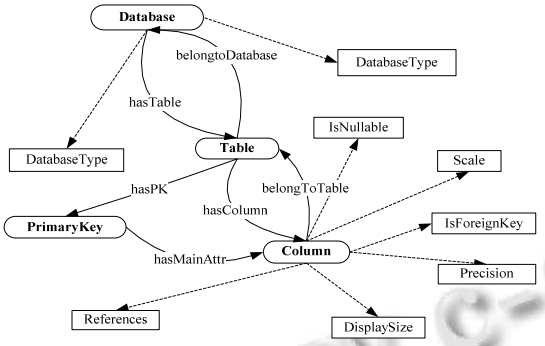

图 2　DSOnto 本体层次结构

2.2 语义视图定义

定义 1. *SV(X)* ← *SPARQL(X, Y)*

*SV(X)*是 SV 的头部, X 是关系表的属性集合; *SPARQL(X, Y)*是 SV 的定义主体, 由 RDF 三元组组成; *X* 和 *Y* 由变量和常量构成, *X* 只包括变量, *Y* 中可以同时包括常量和变量; *X* 中的变量称为映射变量, *Y* 中的变量称为绑定变量, 表示 WeSM 中的类或者属性.

图 3 给出了 SV 定义过程, 数据源 D_1 与 D_2 的关系表与 WeSM 模型中的类和属性之间建立语义映射.

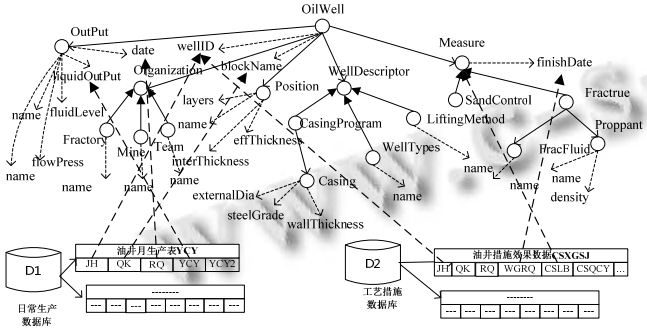

图 3　SV 语义映射

D_1 中的油井月产油表 T_{YCY} 的 SV_{YCY} 定义如下:

SV_{YCY} (JH, QK, RQ, YCY, YCY2) ←

Select ?JH ?QK ?RQ ?YCY ?YCY2

Where {

?X rdf:type wesm:OilWell. ?X wesm:wellID.

?JH.?X wesm:blockName ?QK. ?X wesm:OutPut ?Y.

?Y wesm:date ?RQ. ?Y wesm:oilOutPut ?YCY.

?Y wesm:fluidOutPut ?YCY2}

其中, *JH, QK, RQ, YCY, YCY2* 是映射变量, 分别表示井号, 区块, 日期, 月产油和月产液. *?X*、*?Y* 是绑定变量.

SV 只是关系表语义映射的形式化表示, 还需要将其表示为一种数据结构以记录 SV 中每一个映射变量 X 在 WeSM 中的映射路径, 从而为语义查询重写提供参照信息, SV 的数据结构记为 SVL, 定义如下:

定义 2. SVL 中每一个元素 L 表示映射变量 *X* 在 WeSM 中的映射路径:

SVL = {*L* | *l* ∈ *L*, *l* = <*node, nnode*>. *node, nnode*　*C* ∪ *P* }

P 表示 WeSM 中的属性, *C* 表示 WeSM 中的类; *L* 表示 SVT 中映射变量 *X*, 即关系表中的属性; *L* 由若干 l 元素构成, *l* = <*node, nnode*>是一个二元组, *node* 和 *nnode* 表示为 WeSM 中的类和属性. 所有关系表的 SVL 集合记为 *SetSVL*.

根据定义 2, 油井月产油表 T_{YCY} 的 SVL_{YCY} 结构为:

SVL_{YCY}(JH, QK, RQ, YCY, YCY2) = {

[<OilWell, wellID>],

[<OilWell, blockName>],

[<OilWell, OutPut>, < OutPut, date>],

[<OilWell, OutPut>, < OutPut, oilOutPut>],

[<OilWell, OutPut>, < OutPut, fluidOutPut>]}

SVL_{YCY} 包括 5 个 *L* 元素, 其中 L_{YCY} =[< *OilWell, OutPut*>, <*OilWell, OutPut, oilOutPut*>]包含 2 个 *l* 元素. 表示映射变量"*YCY*", 即表示 *YCY* 的映射路径是"*OilWell* 类的对象属性 *OutPut* 具备的数据类型属性 *oilOutPut*".

L 中每个 *l* 元素定义为 2 种类型: 类类型 l_C 和数据类型 lDTP. l_C 中的 *node* 和 *nnode* 元素, 分别表示 WeSM 中的类和该类的对象类型属性; *lDTP* 中的 *node* 和 *nnode* 元素, 分别表示 WeSM 中的类和该类的数据类型属性, 其中 l_{DTP} 中的专门定义为 *MapCol* 对象, 表示 *L* 所映射关系表的属性列.

如 L_{YCY} 中, <*OilWell, OutPut*>为 l_C 类型, <*OutPut, oilOutPut*>为 l_{DTP} 类型, 记为 $MapCol_{oilOutPut}$, 表示 *YCY* 属性映射为 *OutPut* 类的 *oilOutPut* 数据类型属性.

由于在语义查询时, SPARQL 查询模式 Select 子句

中出现的绑定变量最终都要分解为若干个数据类型属性, 因而定义专门的集合 SVL_{MapCol} 记录 l_{DTP} 类型元素(即 *MapCol* 对象), 为 SPARQL-SQL 查询转换提供映射信息, Set_{MapCol} 定义如下:

定义 3. $SVL_{MapCol} = \{ l \mid l \text{ is } mapCol> \}$

如月产油数据表 *YCY* 的 $SVL_{YCY\text{-}MapCol}$ 定义为:

$SVL_{YCY\text{-}MapCol}$ = {<*OilWell,wellID*>,

<*OilWell,lockName*>,

<*OutPut, date*>,< *OutPut, oilOutPut*>,< *OutPut, fluidOutPut*>}

$Set_{MapCol\text{-}YCY}$ 保存了 5 个 *MapCol* 对象, 分别对应 *YCY* 表的 *JH*, *QK*, *RQ*, *YCY* 和 *YCY2* 映射变量.

3 SPARQL-SQL查询转换

SPARQL 是采用图模式的匹配进行语义查询, 主要包括查询模式、数据集、图模式和结果修饰. 由于系统采用查询构造器将查询需求转化为 SPARQL 语句, 查询需求来源于关系数据库, 因此, 本文重点关注查询模式中的 Select 类型和图模式 Where 中的基本图模式、组图模式和值约束.

SPARQL-SQL 查询转换是将基于 WeSM 语义模型的全局查询转换为面向关系数据库的 SQL 查询, 主要包括: SPARQL 解析和 SPARQL-SQL 查询重写 2 个过程.

SPARQL 解析负责参照 WeSM 模型对查询语句进行解析, 将 Select 和 Where 子句中的绑定变量映射为 WeSM 中的类和数据类型属性, 生成符合 SVL 规范的匹配信息;

SPARQL-SQL 查询重写负责读入解析后的匹配信息, 执行查询重写算法, 生成一系列面向数据源的 SQL 查询计划.

3.1 SPARQL 解析

首先, 构造如下 4 个关系表, 保存 SPARQL 解析后的信息:

(1)SelectTable(CID,Type)

保存查询模式 Select 子句中出现的绑定变量及其类型(如类绑定变量和属性绑定变量).

(2)WhereInfo(CID,Property,Value)

保存图模式 Where 子句中出现的绑定变量、属性或类型标示符和对应的数值.

(3)ClassTable(CID,Type)

保存 WhereInfo 表中的类绑定变量及该变量在 WeSM 中对应的类. 如果某个绑定变量对应一个类层次关系, 则只保存最底层的类.

(4)ConditionTable(CID,Property,Compar,Value)

保存图模式 Where 中 Filter 值约束出现的属性绑定变量、变量对应的类名称、条件设置和数值信息. 其中, 采用 SVL 形式定义 FilterCol 对象, 对 Filter 子句信息进行定义, FilterCol 与 MapCol 结构相同, 都为二元组集合.

表 1 给出了 SPARQL 解析算法. 该算法首先将解析后的基本信息保存到 4 个关系表中; 然后, 通过在每个关系表上定义相关运算函数, 生成符合 SVL 规范的 MapCol 和 FilterCol 实例.

表 1 SPARQL 解析算法

输入: SPARQL 语句 *Q* 输出: Q 应的 *MapCol* 和 *FilterCol* 实例数据

```
I    对 Q 进行解析, 将处理后的信息保存到 4 个表中
II   解析 Select 子句, 生成 MapCol 对象实例
1    for (int i=0;i<SelectTable.rowCount();i++)  {
3    String binding=SelectTable.getValueAt(i,0; //获取元组中的绑定变量
4     String mark=SelectTable.getValueAt(i,1); //获取 binding 的类型标识
5     if mark.equals("ClassType") {    //判断是否是类绑定变量
6        OntClass ontClass=ClassTable.find(binding); //获取 binding 类
7        Iterator iter = ontClass.listDTP(); //获取 ontClass 数据类型属性
8        While (iter.hasNext())//为每个 DTP 构造 MapCol 实例
9        MapCol mapc=new MapCol(iter.next());}
10   if mark.equals("PropertyType"){   //判断是否是属性绑定变量
11       OntClass ontClass=ClassTable.find(binding); //获取属性对应类
12       MapCol mapc=new MapCol(ontClass,binding);}
III  解析 Where 子句, 生成 FilterCol 对象实例
13   for (int i=0;i<conditionTable.rowCount();i++) {
14    String cName = conditionTable.getValueAt(i,0); //获取类名称
15    String dtpName= conditionTable.getValueAt(i,1); //获取属性名称
16    String comName = conditionTable.getValueAt(i,2); //获取比较类型
17    String value= conditionTable.getValueAt(i,3); //获取比较数值常量
18    FilterCol fc=new FilterCol(cName,dtpName,comName,value);}
```

例如, 查询在 2010 年 1 月份实施了增产措施的油井及措施类型名称, SPARQL 语句如下所示:

```
Select ?yj ?cslx
Where { ?yj df:type wesm:OilWell.
        ?yj wesm:measure ?mea.
        ?mea wesm:meaName ?cslx.
```

?mea wesm:finshdate ?fdate.

FILTER (?fdate ='201001')}

根据 SPARQL 解析算法, 首先生成 4 个数据表的元组记录, 如图 4 所示.

WhereInfo

CID	Property	Value
?yj	rdf:type	:OilWell
?yj	: measure	?mea
?mea	: meaName	?cslx
?mea	: date	?fdate

SelectTable

CID	Type
?yj	ClassType
?cslx	PropertyType

ClassTable

CID	Type
?yj	OilWell
?mea	Measure

ConditionTable

CID	Property	Compar	Value
?fdate	:date	=	201001

图 4　SPARQL 解析后填充 4 个表实例

然后, 算法读入 4 个表数据记录, 生成 3 个 *MapCol* 和 1 个 *FilterCol* 对象实例数据, 其中, $FilterCol_{Date}$ 的值约束为'201001'.

$MapCol_{WellID}$ = {<*OilWell, wellID*>}

$MapCol_{BlockName}$ = { <*OilWell, blockName*>}

$MapCol_{MeaName}$ = { <*Measure, meaName*>}

$FilterCol_{Date}$ = { <*Measure, date*>)}

3.2 SPARQL 查询重写

通过表 1 给出的解析算法, 对 SPARQL 语句 *Query* 进行解析并生成实例数据, 然后在所有关系表的 SVL 集合 *SetSVL* 之上实现查询重写, 主要包括以下步骤:

(1)对 *Query* 解析生成 *MapCol* 实例集合 Q_{MapCol} 和 *FilterCol* 实例集合 $Q_{FilterCol}$.

(2)依次选取 $SVL_i \in Set_{SVL}$, 将 SVL_i 对应的 *MapCol* 集合 $SVL_{i\text{-}MapCol}$ 与 Q_{MapCol} 和 $Q_{FilterCol}$ 进行如下匹配:

1)若 $Q_{MapCol} \cup Q_{FilterCol} \in SVL_{i\text{-}MapCol}$, 即满足查询所需的数据来源于同一个关系表, 则将 SVL_i 添加到集合 $Set_{Inclusion}$ 中. 如果该条件不满足, 进入 2):

2)若| $SVL_{i\text{-}MapCol} \cap Q_{MapCol}$ | > 0 或 | $SVL_{i\text{-}MapCol} \cap Q_{FilterCol}$ | > 0, 将 SVL_i 放入集合 $Set_{Intersection}$ 中. 即语义查询所需数据来自多个关系表, 将所有相关表的 SVL_i 保存在该集合中.

(3)若| $Set_{Inclusion}$ | > 0, 依次获取 $SVL_i \in Set_{Inclusion}$, SVL_i 对应关系表记为 T_i, 将 Q_{MapCol} 对应映射变量 *mcol0,...,mcoli* 在表 T_i 进行投影操作; 对 $Q_{FilterCol}$ 映射变量 *fcol0,...,fcoli* 在表 T_i 中进行选择操作, 重写公式:

$SQL' \leftarrow \pi_{(mcol0,...mcoli),\sigma(fcol0,...fcoli)}T_0 ... \cup \pi_{(mcol0,...mcoli),\sigma(fcol0,...fcoli)}T_n$

SQL' 表示查询来自单个数据表所有可能结果的并集.

(4) 如果| $Set_{Intersection}$|>0, 依次获取 $SVL_i \in Set_{Intersection}$,

1)定义如下 2 个集合:

$Inter_{i\text{-}MapCol} = SVL_{i\text{-}MapCol} \cap Q_{MapCol}$

$Inter_{i\text{-}FilterCol} = SVL_{i\text{-}MapCol} \cap Q_{FilterCol}$

2 个集合分别包括的 *MapCol* 和 *FilterCo* 元素既是 *Query* 中出现的绑定变量又是包含于 SVL_i 中的映射变量.

2)如果| $Inter_{i\text{-}MapCol}$ | > 0, 获取每一个 $MapCol_k \in Inter_{i\text{-}MapCol}$, 将其对应的映射变量 *mcol0,...,mcoli* 在 T_k 上进行投影; 如果| $Inter_{i\text{-}FilterCol}$ | > 0, 获取每一个 $FilterCol_k$, 将其对应的映射变量 *fcol0,...,fcolj* 在 T_k 中进行选择操作, 关系代数为:

$SQL_k \leftarrow \pi_{(mcol0,...,mcoli),\sigma(fcol0,...,fcolj)}T_k$,

SQL_k 表示每一个 T_k 能够为语义查询提供相关数据的集合. 通过该过程, 获得多个 SQL_k:

$SQL_0 \leftarrow \pi_{(mcol0,...,mcoli),\sigma(fcol0,...fcolj)}T_0$,

$SQL_1 \leftarrow \pi_{(mcol0,...,mcoli),\sigma(fcol0,...fcolj)}T_1$, ...,

$SQL_k \leftarrow \pi_{(mcol0,...,mcoli),\sigma(fcol0,...fcolj)}T_k$.

3)由于 $Set_{Intersection}$ 集合中各个 SVL_i 对应表之间必然存在参照完整性约束, 表现为相关表主键 *pki*、外键 *fki* 之间存在等值条件, 记为 *Conditon = { Ti.fki = Tj.pkj | Ti ref Tj}*, 为此, 需要对 SQL_0, ..., SQL_k 相关联的表进行等值连接, 再在 Q_{MapCol} 对应映射变量 *qcol0,...,qcolk* 上进行投影, 关系代数为:

$SQL'' \leftarrow \pi_{(qcol0,...,qcolk),\sigma(Conditon)}(SQL_0 \infty SQL_1 \infty ... \infty SQL_k)$

(5)将步骤 3 结果 Q'和步骤 4 结果 Q''的进行求并集运算, 即:

$Result = SQL' \cup SQL''$

根据 DSonto 关系模式和数据库连接信息, 在对应数据源上执行 SQL' 和 SQL'' 的数据源, 获取的最终查询结果, 并通过相关处理后返回给用户.

4　系统实现

根据提出的实现方法, 基于 Java 技术构建了基于语义的油气井虚拟数据中心 WeVDC 原型系统. WeVDC 的数据源来自大港油田某单位的 3 个关系数

据库, 2 个采用 Oracle 9i 关系数据源, 1 个采用 SQL Server2005.

图 5 给出了语义视图定义界面. 左边显示了 WeSM 领域模型层次结构, 右边树形控件显示了连接成功后数据源的关系模式; 选择某个数据表后, 可以在映射设置面板进行映射配置, 中间下部表格显示已经完成语义视图定义的映射信息, 图 6 显示了语义查询构造界面.

图 5 语义视图定义界面

图 6 语义查询构造界面

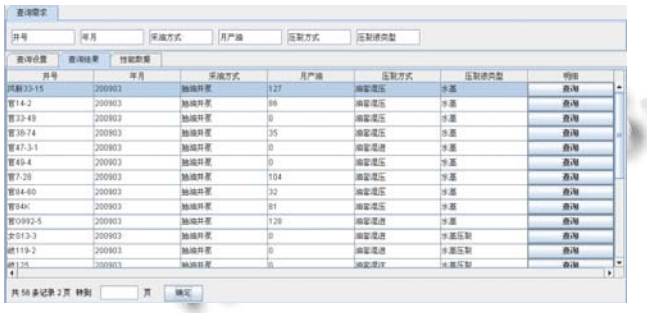

图 7 查询结果显示界面

用户从左侧选择油气井领域概念, 确定要查询的对象; 右侧查询设置会显示该查询对象具有的数据类型属性和对象属性; 对于基本属性, 可以在前面复选框勾选并设置查询需求, 由于对象属性会包含其他对象属性和基本属性, 可以通过勾选前面的展开复选框进一步查看该属性的具体信息.

完成查询设置后, 点击工具栏中的查询按钮, 系统会生成相应的 SPARQL 查询, 通过查询转换过程, 系统从所注册的数据源中进行查询, 图 7 显示了查询结果界面. 点击每一行右侧的“查询”按钮后, 会提出该查询相关的明细记录.

5 结语

本文针对油气井工程领域的关系数据源, 给出了一种语义数据集成方法. 该方法通过在关系表与 WeSM 之间建立语义映射, 借助语义查询重写算法将基于 WeSM 的语义查询重写为对底层关系数据源的 SQL 访问, 该方法不需要进行数据导出和转换, 为领域语义数据集成提供了一个有效的解决方案. 通过油气井工程虚拟数据中心 WeVDC 原型系统, 对提出的方法进行了验证, 并获得了良好的应用效果.

参考文献

1 张丽丽.基于 SQL 转换的关系数据库 SPARQL 查询方法的研究[硕士学位论文].沈阳:东北大学,2011.

2 Hartig O, Heese R. The SPARQL query graph model for query optimization. Proc. of the 4th European Semantic Web Conference. Austria. Springer-Verlag. 2007. 564–578.

3 Chebotko A, Lu SY, Farshad F. Semantics preserving SPARQL to-SQL translation. Data & Knowledge Engineering, 2009, 68(10): 973–1000.

4 Wang LQ, Lu SY, Fei XB, et al. Atomicity and provenance support for pipelined scientific workflows. Future Generation Computer Systems, 2009, 25(5): 568–576.

5 Harris S, Shadbolt N. SPARQL query processing with conventional relational database systems. WISE 2005 Workshops, LNCS 3807. 2005. 235–244.

6 张伟奇.基于关系型数据库的 RDF 存储引擎[硕士学位论文].天津:天津大学,2011.

7 Abadi DJ, Marcus A, Madden SR. SW-store: A vertically partitioned DBMS for Semantic Web data management. The International Journal on Very Large Data Bases, 2009, 18(2): 385–406.

8 于彤,刘静,刘丽红,贾李蓉,杨硕,张竹绿,李敬华,于琦.面向中药数据库的语义集成框架.中国数字医学,2015,1:78–81.

9 乔金风.智能电网语义数据集成关键技术研究与应用[硕士学位论文].保定:华北电力大学,2014.

10 De Laborda CP, Conrad S. Relational. OWL: A data and schema representation format based on OW. Proc. of the 2nd Asia-Pacific Conference on Conceptual Modelling. Australia, Australian Computer Society. 2006. 89–96.