Mapping Relational Data to RDF in Virtuoso

Orri Erling (Program Manager, OpenLink Virtuoso), Ivan Mikhailov (Lead Developer, OpenLink Virtuoso).

Abstract

This paper discusses Virtuoso's new declarative Meta Schema Language for mapping SQL Data to RDF Ontologies. This functionality, commonly referred to as "RDF VIEWs", enables the exposure of pre-existing ODBC or JDBC accessible relational data as Virtual RDF Graphs thereby exposing said data to SPARQL query access directly or via Virtuoso's in-built support for SPARQL within SQL (SPASQL). The entire process results in RDF Data sets and optimized data access without physical regeneration of RDF Data Sets from SQL Data.

Considering that almost all of the present generation of Web 2.0 and Enterprise applications run atop relational databases, the need for efficiently exposing existing data to the burgeoning Semantic Web is obvious. For instance, it now possible to dynamically generate RDF Data Sets for popular ontologies such as SIOC, SKOS, FOAF, and ATOM/OWL without disruption to the existing database infrastructure that drives existing Web 1.0 or Web 2.0 solutions. The same applies to Enterprises seeking to exploit many of the collaborative prowess of the Web 2.0 application profiles. As part of this paper we provide a simple example of how the Weblog Data Space within the OpenLink Data Spaces collective is mapped to the AtomOWL Ontology en route to making the Weblog Data Space data accessible to SPARQL.

Introduction

The Virtuoso RDF meta schema is a built-in feature of Virtuoso's SPARQL to SQL translator. It recognizes triple patterns that refer to graphs for which an alternate representation is declared and translates these into SQL accordingly. The main purpose of this is mapping SPARQL queries onto existing relational databases.

There exists previous work from many parties for rendering relational data as RDF and opening it to SPARQL access. We can mention D2RQ, SPASQL, Squirrel RDF and others. The Virtuoso effort differs from these mainly in the following:

- Integration with a triple store. Virtuoso can process a query for which some triple patterns will go to local or remote relational data and some to local physical RDF triples.
- Integration with SQL. Since SPARQL and SQL share the same run time and query optimizer, the query compilation decisions are always made with the best knowledge of the data and its location. This is especially important when mixing triples and relational data or when dealing with relational data distributed across many outside databases.
- No limits on SPARQL. It remains possible to make queries with unspecified graph or

predicate against mapped relational data, even though these may sometimes be inefficient.
- Coverage of the whole relational model. Multi-part keys, etc., are supported in all places.

Rationale

Since most of the data that is of likely use for the emerging semantic web is stored in relational databases, the argument for exposing this to SPARQL access is clear. We note that historically, SQL access to relational data has essentially never been given to the public outside of the organization. If programmatic access to corporate IS has been available to the public, it has been through dynamic web pages or more recently web services. There are reasons of performance, security, maintainability and so forth for this.

The culture of the emerging semantic web is however taking a different turn. Since RDF and OWL offer a merge-able and query-able model for heterogeneous data, it is more meaningful and maintainable to expose selected data for outside query than it would be with SQL. Advances in hardware make this also less of a performance issue than it would have been in the client-server database era.

In the context of Virtuoso, since Virtuoso is originally a virtual/federated database, incorporating SPARQL to relational mapping is an evident extension of the product's mission as a multi-protocol, multi-platform connector between information systems.

Graph Patterns, Value and IRI Classes

In the simplest sense, any relational schema can be rendered into RDF by converting all primary keys and foreign keys into IRI's, assigning a predicate IRI to each column, and an rdf:type predicate for each row linking it to a RDF class IRI corresponding to the table. Then a triple with the primary key IRI as subject, the column IRI as predicate and the column's value as object is considered to exist for each column that is neither part of a primary or foreign key.

Strictly equating a subject value to a row and each column to a predicate is often good but is too restrictive for the general case. For example, multiple triples with the same object and predicate can exist. Or a single subject can get single-valued properties from multiple tables or in some cases stored procedures.

The basic elements of the Virtuoso RDF meta schema are:

- IRI Class - This declares that a key column or set of key columns gets converted into a URI in a certain way. When declaring that a table's primary key is converted into a URI according to one IRI class, one usually declares that all foreign keys referring to this class also get converted into an IRI as per this same class.
- Value Class - This is a mechanism for unit conversions, for converting enumerations into IRI's or any other scalar to scalar conversion. The difference between a value class and an IRI class is that a value resulting from converting a relational column according to a value class is not necessarily an IRI joinable to the subject or predicate of a triple.

- Quad pattern - This declares that a quad with any combination of specified or unspecified G, S, P or O is stored in a non-default way. The patterns are grouped together in RDF views. A quad pattern consists of the following:
    - Subject - This can be an IRI class followed by one of more columns of a table. The IRI class specifies how the column values are combined into an IRI string and how an IRI string is decomposed back into the key value. Alternately, this can be a constant IRI.
    - Predicate - This is most often a constant IRI but can also be an IRI class as with the subject.
    - Object - This can be an IRI class applied to columns, a value class applied to a column or a literal IRI or scalar.
    - Condition - This is an optional SQL search condition. If the search condition is true when applied to a row of the table designated in the subject, predicate and object patterns, then the triple is considered to exist, otherwise not.
    - Optional RDF type of the subject. If the subject is known to be of a type disjoint from this type, then the quad pattern will not apply. We note that when S, P, or O refers to columns of a table, all of these should be to the same table. For complex joins, the table in question can also be a SQL view joining multiple tables. Thus the search condition of the quad pattern does not have to be used as a join condition between different tables.
    - Graph - This is most often a constant IRI but can also be an IRI class as with the subject. It is very common for many quad patterns of one SQL schema to share one constant graph, but one meta schema may consist of quad patterns with any number of different graph declarations.
- Quad Storage -- This is a named set of quad patterns. This is used for compartmentalizing the RDF to SQL mapping. This allows specifying that a SPARQL query will be executed using only quad patterns of the given quad storage. Declarations of IRI classes, value classes and quad patterns are shared between all quad storages of an RDF meta schema but every quad storage contains only a subset of all available quad patterns. Two quad storages are always defined: a 'virtrdf:default' one usually consists of everything (all relational mappings plus a table of physical quads) and an 'virtrdf:empty' storage that refers to single table of physical quads).

A special case of IRI or value class is the identity class, which is simply marked by table.column. For this to occur as S or P, the column must be of the IRI ID type. For an O, the column may be of any type.

Example - Mapping Virtuoso Database Users and Roles to Accounts in the SIOC Ontology

Here we map the Virtuoso sys_users and sys_role_grant tables to the SIOC ontology. The tables are in part declared as follows, some columns are omitted for brevity:

```
create table sys_users (u_id int primary key, u_name varchar,
u_is_role int, u_e_mail, u_home varchar, u_dav_enable int...)
create table sys_role_grants (gi_super int, gi_sub int,
gi_direct, primary key (gi_super, gi_sub, gi_direct));
```

The column gi_super is the role and gi_sub is the principal to which the role is granted. Both refer to u_id in sys_users. A row of sys_users is either a role or an account, as indicated by u_is_role. All role memberships, including indirect ones are listed in sys_role_grants, the gi_direct flag shows whether this is a directly granted role.

We can map this into the sioc:User and sioc:Usergroup RDF classes. These are then related by the mutually inverse predicates sioc:member_of and sioc:has_member. Additionally, a group can have other groups as members, although not circularly.

To define this mapping, we need the following group of declarations that form one big statement:

```
sparql
pragma prefix oplsioc:
<http://www.openlinksw.com/schemas/oplsioc#>
pragma prefix sioc: <http://rdfs.org/sioc/ns#>
```

-- We put all the SIOC names in the original SIOC namespace. The names that pertain to the mapping go into the oplsioc namespace.

```
drop graph iri("^{URIQADefaultHost}^/dataspace") .
```

-- We have the notation ^{URIQADefaultGraph}^ to refer to the default host declared for the server on which the mapping is being loaded. This allows IRI's to be dereferenceable on the host itself. For example, the IRI's declared for the various keys can go to dynamic web pages giving information on the entities being described. In addition, the URIQA protocol server uses this host name to convert relative URIs of local web pages into absolute URIs that can be used by browsers.

```
create iri class oplsioc:user_iri
  "http://^{URIQADefaultHost}^/sys/user?id=%d" (in uid integer
not null) .

create iri class oplsioc:group_iri
  "http://^{URIQADefaultHost}^/sys/group?id=%d" (in gid integer
not null) .
create iri class oplsioc:membership_iri
  "http://^{URIQADefaultHost}^/sys/membersip?super=%d&sub=%d"
  (in super integer not null, in sub integer not null) .

create iri class oplsioc:dav_iri "http://^{URIQADefaultHost}^%s"
(in path varchar) .

create iri class oplsioc:grantee_iri using
  function DB.DBA.RDF_DF_GRANTEE_ID_URI (in id integer) returns
varchar ,
  function DB.DBA.RDF_DF_GRANTEE_ID_URI_INVERSE (in id_iri
varchar) returns integer .
```

```
make oplsioc:user_iri subclass of oplsioc:grantee_iri .
make oplsioc:group_iri subclass of oplsioc:grantee_iri .
```

-- Here we declare the entities that we will see in RDF. These are the user and usergroup and their abstract superclass called grantee. Note that the GI_SUB column of sys_role_grants can refer to either a user or a group and we have decided to give them different IRI's. This is not absolutely necessary since we can tell from the id whether something is a user or group but this may be good for readability and serves to illustrate features of the mapping.

We can map the group membership simply as a many to many without attributes. For some queries, though, we will need to see this relationship as an entity with its own attributes, for example whether the grant is direct and whether it is re-grantable. (the sql with admin option).

For the case where we are interested in the attributes of the relationship, we define the iri class for membership, although this is not part of SIOC, hence the oplsioc namespace.

```
alter quad storage virtrdf:DefaultQuadStorage
  {
    create virtrdf:SysUsers as graph iri
("http://^{URIQADefaultHost}^/dataspace") option (exclusive)
      {
        oplsioc:user_iri (DB.DBA.SYS_USERS.U_ID)
            a sioc:user where (^{alias}^.U_IS_ROLE = 0) ;
            sioc:email U_E_MAIL ;
            sioc:login U_NAME ;
            oplsioc:home oplsioc:dav_iri (U_HOME) where
(^{alias}^.U_DAV_ENABLE = 1) ;
            oplsioc:name U_FULL_NAME where (^{alias}^.U_IS_ROLE =
0) .
        oplsioc:group_iri (DB.DBA.SYS_USERS.U_ID)
            a sioc:role where (^{alias}^.U_IS_ROLE = 1) .
        oplsioc:group_iri (DB.DBA.SYS_ROLE_GRANTS.GI_SUPER)
            sioc:has_member oplsioc:grantee_iri (GI_SUB) ;
            oplsioc:group_of_membership
                oplsioc:membership_iri (GI_SUPER, GI_SUB) .
        oplsioc:grantee_iri (DB.DBA.SYS_ROLE_GRANTS.GI_SUB)
            sioc:has_function oplsioc:group_iri (GI_SUPER) ;
            oplsioc:member_of
                oplsioc:membership_iri (GI_SUPER, GI_SUB) .
        oplsioc:membership_iri (DB.DBA.SYS_ROLE_GRANTS.GI_SUPER,
GI_SUB)
            oplsioc:is_direct GI_DIRECT ;
            rdf:type oplsioc:grant .
      }
  }
;
```

-- Now we declare the actual mapping. In the graph /dataspace of the default host of the server, the above rules will apply.

The first rule declare that `?s rdf:type sioc:User` is true of any IRI matching the

user_iri (sys_users.u_id) pattern. This means that:

- the constant part of the IRI is as defined for the user_iri IRI class.
- there is a row of DB.DBA.SYS_USERS with U_ID equal to the number extracted form the IRI as per the format string in the create iri class statement, i.e. `^{URIQADefaultHost}^/user?id=%d`.
- The rule is applicable U_IS_ROLE column of the row should be equal to zero.

After this, we map some scalar properties, for example email. In defining repeating predicates for the same subject, the subject does not have to be repeated if the semicolon is used as delimiter, just as with the SPARQL syntax for triple patterns. Since U_E_MAIL is just a string, no special mapping is needed. If we wanted it as a mailto:// IRI, we could define an IRI class that would convert the string between the SPARQL and SQL values.

The rule with `group_iri (SYS_USERS.U_ID) subject` is almost identical to the first rule with `user_iri (SYS_USERS.U_ID)` subject.

Then we declare the many-to-many between roles and users. The predicates are the SIOC has_function and has_member. They are identical, except that the subject and object are reversed. The group is always a group, the member is either a group or user, hence the use of the grantee_iri IRI class. The mapping functions of this IRI class will have to consult the SYS_USERS table to decide which kind of IRI to make:

```
create function DB.DBA.RDF_DF_GRANTEE_ID_URI (in id integer)
{
  declare is_role integer;
  is_role := coalesce ((select top 1 U_IS_ROLE from
DB.DBA.SYS_USERS where U_ID = id));
  if (is_role is null)
    return NULL;
  else if (is_role)
    return sprintf ('http://%s:/sys/group?id=%d',
get_uriqa_default_host(), id);
  else
    return sprintf ('http://%s:/sys/user?id=%d',
get_uriqa_default_host(), id);
}
;

create function DB.DBA.RDF_DF_GRANTEE_ID_URI_INVERSE (in id_iri
varchar)
{
  declare parts any;
  parts := sprintf_inverse (id_iri, concat ('http://',
get_uriqa_default_host(), '/sys/user?id=%d', 0);
  if (parts is not null)
    { -- The syntax of given IRI matches to the format pattern
for IRIs of user accounts
      if (not exists (select top 1 1 from DB.DBA.SYS_USERS where
U_ID = parts[0] and not U_IS_ROLE))
        return NULL; -- The ID specified is not valid user ID
      return parts[0]; -- Both syntax and the ID value are OK
```

```
    }
  parts := sprintf_inverse (id_iri, concat ('http://',
get_uriqa_default_host(), '/sys/group?id=%d', 0);
  if (parts is not null)
    {
      if (not exists (select top 1 1 from DB.DBA.SYS_USERS where
U_ID = parts[0] and U_IS_ROLE))
        return NULL;
      return parts[0];
    }
  return NULL;
}
;
```

The clause `options (exclusive)` instructs the SPARQL compiler that all quads of the graph `'http://^{URIQADefaultHost}^/dataspace'` are described by this set of declared quad patterns. Even if the table of physical quads contains quads with graph `'http://^{URIQADefaultHost}^/dataspace'` they may be ignored by the SPARQL query. Any group of quad patterns or a single quad pattern can be declared exclusive. Exclusive clauses may conflict. Consider an exclusive group of patterns for `.../dataspace` graph and an single exclusive quad pattern for `sioc:email` predicate (say, one might have a table DB.DBA.E_MAILS with G, S and O columns of triples and a `sioc:email` predicate is assumed for all quads stored there). Consider a query

```
<code>sparql select * where { graph <http://localhost/dataspace>
{ ?s sioc:email ?o } }</code>
```

that contains a triple pattern that matches both `.../dataspace` and `sioc:email` mapping rules. Each rule prevents the compiler from using another rule. To resolve ambiguities, a quad storage may specify the order of quad patterns so that the compiler will use the first suitable quad pattern and ignore the rest.

Sample Queries

Two Triple Patterns As A Single Table Lookup

Select all accounts that have both full names and home DAV collections.

```
sparql
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
select ?fullname ?home
where {
  graph <http://localhost/dataspace> {
    ?uid sioc:name ?fullname .
    ?uid oplsioc:home ?home } }

fullname         home
VARCHAR          VARCHAR
_______________________________________________
```

```
Ivan Mikhailov   http://localhost/DAV/home/iv_an_ru/
Orri Erling      http://localhost/DAV/home/oerling/
```

The generated SQL code contains only one table in FROM clause and its WHERE clause contains texts of all WHERE clauses of relevant quad patterns; duplicate conditions are removed.

```
SELECT "t0"."U_FULL_NAME" AS "fullname",
  case (isnull ("t0"."U_HOME")) when 0 then sprintf
('http://localhost%s', "t0"."U_HOME") else NULL end AS "home"
FROM DB.DBA.SYS_USERS AS "t0"
WHERE ("t0"."U_FULL_NAME" is not null)
  AND ("t0".U_HOME is not null) and ("t0".U_DAV_ENABLE = 1)
  AND ("t0".U_IS_ROLE = 0)
OPTION (QUIETCAST)
```

Simple Join of Three Tables

Select all pairs of grantees and their roles, return names instead of integer IDs:

```
sparql
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
select ?super ?sub
where {
  graph <http://localhost/dataspace> {
    ?superid sioc:has_member ?subid .
    ?superid sioc:id ?super .
    ?subid sioc:id ?sub .
     } }

super                sub
VARCHAR NOT NULL     VARCHAR NOT NULL
_____________________________________

SPARQL_UPDATE        SPARQL_SELECT
```

The query is translated into the following SQL:

```
SELECT "t1"."U_NAME" AS "super",
    "t2"."U_NAME" AS "sub"
FROM DB.DBA.SYS_ROLE_GRANTS AS "t0",
    DB.DBA.SYS_USERS AS "t1",
    DB.DBA.SYS_USERS AS "t2"
WHERE "t0"."GI_SUPER" = "t1"."U_ID" /* superid */
    AND "t0"."GI_SUB" = "t2"."U_ID" /* subid */
OPTION (QUIETCAST)
```

Optional Clause and Filtering of Table Rows

```
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
select ?uid ?login ?fullname ?home
where
  {
    graph <http://localhost/dataspace>
      {
        ?uid sioc:id ?login .
        ?uid sioc:name ?fullname .
        optional { ?uid oplsioc:home ?home }
      }
  }

uid                                  login               fullname
home
VARCHAR                              VARCHAR NOT NULL    VARCHAR
VARCHAR
_____________________________________________________________________
_____________________________________________

http://localhost/sys/user?id=2     dav                 WebDAV
System Administrator   NULL
http://localhost/sys/user?id=106   iv_an_ru            Ivan
Mikhailov                http://localhost/DAV/home/iv_an_ru/
http://localhost/sys/user?id=5     nobody              Special
account                NULL
http://localhost/sys/user?id=107   oerling             Orri Erling
http://localhost/DAV/home/oerling/
```

The query is translated into the following SQL:

```
SELECT sprintf ( 'http://localhost/sys/user?id=%d' , "t0"."uid" )
AS "uid",
    "t0"."login",
    "t0"."fullname",
    case (isnull ( "t1"."home" )) when 0 then sprintf (
'http://localhost%s' , "t1"."home" ) else NULL end AS "home"
FROM (
    SELECT "t00"."U_ID" AS "uid",
        "t00"."U_NAME" AS "login",
        "t01"."U_FULL_NAME" AS "fullname"
    FROM DB.DBA.SYS_USERS AS "t00",
        (SELECT "t01-int"."U_ID", "t01-int"."U_FULL_NAME"
         FROM DB.DBA.SYS_USERS AS "t01-int"
         WHERE ("t01-int"."U_FULL_NAME" is not null)
           AND ("t01-int".U_IS_ROLE = 0)
        ) AS "t01"
    WHERE "t00"."U_ID" = "t01"."U_ID"
    ) AS "t0"
  LEFT OUTER JOIN (
      SELECT "t10"."U_ID" AS "uid", "t10"."U_HOME" AS "home"
      FROM (
          SELECT "t10-int"."U_ID", "t10-int"."U_HOME" FROM
DB.DBA.SYS_USERS AS "t10-int"
```

```
          WHERE ("t10-int"."U_HOME" is not null)
            AND ("t10-int".U_DAV_ENABLE = 1)
          ) AS "t10") AS "t1"
  ON ( "t0"."uid" = "t1"."uid" )
OPTION (QUIETCAST)
```

Mix Of Relational Data and Plain RDF Quads

```
sparql
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
select ?home ?id ?login
where {
    graph ?g { ?id oplsioc:home ?home } .
    optional {
        graph <http://localhost/dataspace> {
            ?id sioc:id ?login }
      } }

home                                    id
login
VARCHAR                                 VARCHAR
VARCHAR NOT NULL
_______________________________________________________________________
______________

http://localhost/DAV/home/iv_an_ru/  http://localhost/sys/user?
id=106  iv_an_ru
http://localhost/DAV/home/oerling/   http://localhost/sys/user?
id=107  oerling
```

The context of the first triple pattern does not specify graph, so triples with oplsioc:home predicate may be retrieved from both the default storage of RDF quads (table RDF_QUADS) and mapping of relational data (table SYS_USERS). The graph of the second triple pattern matches 'exclusive' group of quad patterns so the compiled query will not try to locate sioc:id predicates in RDF_QUADS table and will make no second union. Thus the intermediate result of the compilation is like this:

```
(sys_users as t00 union all rdf_quad as t10) left outer join
sys_users as t1)
```

The optimizer will convert a join of unions into a union of joins so the final query is like

```
(sys_users as t00 left outer join sys_users as t01) union all
(rdf_quad as t10 left outer join sys_users as t11)
```

and can be executed efficiently:

```
SELECT DB.DBA.RDF_SQLVAL_OF_LONG ("t00"."home") AS "home",
    sprintf ('http://localhost/sys/user?id=%d', "t00"."id") AS
"id",
```

```
    "t01"."login"
FROM (
    SELECT "t00a"."U_ID" AS "id",
         case (isnull ("t00a"."U_HOME")) when 0 then
DB.DBA.RDF_MAKE_IID_OF_QNAME (sprintf ('http://localhost%s',
"t00a"."U_HOME")) else NULL end AS "home"
    FROM DB.DBA.SYS_USERS AS "t00a"
    WHERE ("t00a"."U_HOME" is not null) AND ("t00a".U_DAV_ENABLE
= 1)
    ) AS "t00"
  LEFT OUTER JOIN (
      SELECT "t01-t1"."U_ID" AS "id", "t01-t1"."U_NAME" AS
"login"
      FROM DB.DBA.SYS_USERS AS "t01-t1") AS "t01"
  ON ( "t00"."id" = "t01"."id")
UNION ALL
SELECT DB.DBA.RDF_SQLVAL_OF_LONG ("t10"."home"),
    DB.DBA.RDF_SQLVAL_OF_LONG ("t10"."id"),
    "t11"."login"
FROM (
        SELECT "t10a"."S" AS "id",
             DB.DBA.RDF_LONG_OF_OBJ ("t10a"."O") AS "home"
        FROM DB.DBA.RDF_QUAD AS "t10a"
        WHERE
            "t10a"."P" = DB.DBA.RDF_MAKE_IID_OF_QNAME_SAFE
(UNAME'http://www.openlinksw.com/schemas/oplsioc#home')) AS "t10"
  LEFT OUTER JOIN (
      SELECT "t11a"."U_ID" AS "id", "t11a"."U_NAME" AS "login"
      FROM DB.DBA.SYS_USERS AS "t11a") AS "t11"
  ON ( "t10"."id" = DB.DBA.RDF_MAKE_IID_OF_QNAME
(DB.DBA.RDF_DF_GRANTEE_ID_URI ("t11"."id")))
OPTION (QUIETCAST)
```

Simplifying Queries

In principle, we take each triple pattern from a SPARQL query and expand it into a union of all the quad mapping rule expansions that apply to it. If we have an unspecified graph and unspecified ?s ?p ?o, we indeed get the union of everything.

In practice, the graph and the predicates are most often given. This will considerably limit the number of applicable quad patterns. We further assume all IRI classes to be disjoint, unless exceptions are declared with the `create ... subclass ... statement` shown in the example with user_iri and grantee_iri. Thus many joins can be known to be empty at compile time.

In the simplest case, when we have two triple patterns with the same subject and the subject is mapped to a relational table, we get a join of the table with itself, with pk = pk. This self-join can however be optimized away. After this, there will be condition on multiple columns and normal SQL index and join type selection will apply.

As a last step, after pruning the union terms that are known not to be relevant, the unions are

taken to the top, so that we have a union of joins and not a join of unions. This is easier for the SQL compiler to optimize.

We note that throughout this process, some expansions of triple patterns can refer to relational tables and some to the default RDF_QUAD table or other native RDF storage tables. This is not a problem at any stage. Thus a mixed representation where some things are obtained from local or remote relational stores and some things are stored locally as physical triple is entirely feasible.

SQL Optimizer Support for Meta Schema

An equality of two IRI's can become an AND of key column equalities. When this takes place between SPARQL triple patterns mapped to the relational tables, the columns that make up the URI are simply compared pair-wise, as in any regular relational join. However, it is also possible that a URI composed from relational columns is compared with an IRI_ID stored in a physical triple. In this case, we have a comparison between and S, P or O of a triple and a function over one or more columns of a relational table.

If the relational table is first in join order, there is no problem. The function is evaluated and the result is used to retrieve the triple. However, if the triple is first in join order, we must be able to decompose the IRI into its constituent parts, that is, if the IRI is such that it can be returned by the key to IRI function corresponding to the columns.

The Virtuoso SQL compiler has a notion of inverse functions. This allows both join orders to be tried, in one case converting the key part(s) into an IRI ID and in the other case the IRI_ID into the key part(s).

Also, when a relational key is mapped into an IRI, making an IRI_ID for the text of the IRI is actually necessary only when there actually is a triple in the triple store that references this IRI. Thus, if joining a relational key to triples, the key to IRI_ID function can return NULL if no IRI_ID exists. The IRI_ID will be made only when an actual triple is created. When clients consume IRI's anyway as strings, the existence or non-existence of a physical IRI ID is entirely transparent. To facilitate this situation, the compiler recognizes the comparison of an IRI column to a string-valued function or variable. It will automatically insert a 'get IRI_ID for string if exists' function into the compilation.

We note that when we cast primary keys to IRI's and pass them back and forth, we may often get situations where key_to_iri (t1.k1, t1.k2) = key_to_iri (t2.k1, t2.k2) where t1 and t2 are different correlation names for a pk and fk tables and k1, k2 are the parts of a 2 part key. This is noticed by the SQL compiler and optimized into t1.k1 = t2.k1 and t1.k2 = t2.k2. Thus no function indices or the like are needed. It suffices that the key to iri functions are declared as having inverses and as being bijective with the sinv_create_inverse function. See the documentation for this.

Also, when joining native RDF triples and relational data, we get situations like rdf_quad.S = key_to_iri (t1.k1, t1.k2). We note that there may not be an IRI ID corresponding to the key_to_iri function's result. The key_to_iri function returns a string, whereas the S,P,O,G of the

RDF_QUAD table are IRI ID's referenced RDF_URL. The SQL compiler recognizes this and puts a cast there, one that will convert the string returned by the key_to_iri function into an IRI ID if there indeed is such an ID and NULL otherwise. Furthermore, because all key_to_iri functions have inverses, the condition can be rewritten as

```
iri_to_key_1 (rdf_iri_id_qname (rdf_quad.S)) = t1.k1 and
iri_to_key_2 (rdf_iri_id_qname (rdf_quad.S)) = t1.k2.
```

This allows both join orders to be used with all index usage options, again without requiring function indices or the like.

We note that all these optimizations will serve equally well whether the relational tables are local to Virtuoso or not. The native RDF tables will have to be local due to their use of the native IRI_ID data type.

Using Meta Schema for Triple Storage Optimization

If we know that some predicates are always single-valued within a certain graph, we can use a relational-like table with the subject as primary key and the objects of single-valued predicates as columns. Then we can make a single-column index on each of the columns.

Oracle has used SQL materialized views for creating such data structures on the fly. Their name for this is SPMJV for Subject-Predicate Matrix Join View. We can use the meta schema graph patterns for implementing a similar thing at the RDF to SQL transformation and RDF storage level.

Using Table Valued Functions for RDF on Demand

Data that we wish to make SPARQL query-able are not always in relational tables. For example, some data can come from web services. Caching all possible outputs of a web service in a relational table in the anticipation of SPARQL access does not make sense in the general case.

Also, data is sometimes de-normalized into non-first normal form columns, for example, the tags of an article can be all stored in a single text column with a text index on it. Still, for purposes of RDF rendition, we wish to consider each tag as a separate sioc:topic triple.

For these cases, Virtuoso offers table valued functions, also known as procedure views. A piece of procedure code can thus be invoked from the FROM clause of a SQL select and its result set will be considered as that of a derived table.

A graph pattern can refer to columns of procedure views just as it can refer to columns of actual tables. The SQL compiler will choose a reasonable join order based on cost information associated with the procedure view.

For example, consider

```
create procedure TV_PROC (in blog_id int, in post_uid int, in tag
varchar)
{
  __cost (20, 1, 1000, 100, 5);
   /* this means that the procedure costs 20 units to invoke
    * and returns 1 row if all parameters have a non-NULL value.
    * If the 1st is not given, multiply these by 1000.
    * If the 2nd is not given, by 100, if the 3rd is not given,
by 5. */
 ...
   /* if blog id is given, loop over all posts and tags.
    * If blog id and post id are given, loop over all tags of the
post.
    * If all are given, give a result row if the post has the tag
    * and so on, for all the 8 combinations of null or non-null
value of the three parameters */
}

create procedure view  BLOG_TAG_VIEW
as TV_PROC (in TV_BLOG_ID int, in TV_POST_ID int, in TV_TAG
varchar)
(TV_BLOG_ID int, TV_POST_ID int,  TV_TAG varchar);

sparql
graph ....
{
post_iri (BLOG_TAG.TV_BLOG_ID, BLOG_TAG.TV_POST_ID) sioc:topic
TV_TAG .
}
```

Now we can do a query like:

```
select * from <ods> where { ?me sioc:name "Orri" . ?me author_of
?post . ?post sioc:topic ?tag}
```

The join order will be left to right because the compiler knows that the tag procedure has a very high cost factor for not specifying the two first arguments, so it will place the procedure call at the end, where the post id is known, instead of iterating over all posts and tags and then seeing which of these has "Orri" as name of author.

Example: Mapping ODS Weblog Post Data to the AtomOWL Ontology

Consider a weblog application from OpenLink Data Spaces (ODS).

A blog consists of blog posts.

Every post may have comments.

Every blog is owned by a user.

A blog owner is a 'database user' registered in DB.DBA.SYS_USERS even if SQL access is disabled for him. A user may create as many blogs as he wishes.

The application uses a database schema that is not very convenient for the mapping to AtomOWL.

It is easy to form some set of subjects and predicates, because relational data are all in 3rd normal form.

It's not easy to form predicates exactly as described in an independent AtomOWL standard and to form all IRIs to match names of existing resources.

This example demonstrates the urgent need in extending basic data mapping functionality to map arbitrary joins of relational tables as opposed to VIEWs only.

When four tables form a group (e.g. user, blog, post and comment) come from different tables we need additional relational views.

```
create view BLOG.DBA.SYS_BLOG_INFO_FULL as
select
  bi.BI_WAI_NAME, -- Blog application instance name
  u.U_ID, u.U_NAME, u.U_E_MAIL, u.U_HOME
from
  BLOG.DBA.SYS_BLOG_INFO as bi
  left outer join DB.DBA.SYS_USERS as u on (U_ID = BI_OWNER)
;

create view BLOG.DBA.SYS_BLOG_FULL as
select
  b.B_POST_ID,  -- ID of a post
  b.B_BLOG_ID,  -- ID of a blog where the post appears
  b.B_TITLE,    -- Post title
  b.B_CONTENT,  -- Content of a post
  b.B_TS,       -- Creation time
  b.B_USER_ID,  -- Post author
  b.B_MODIFIED, -- Last modified
  bi.BI_WAI_NAME,
  u.U_ID, u.U_NAME, u.U_E_MAIL, u.U_HOME
from
  BLOG.DBA.SYS_BLOGS as b
  left outer join BLOG.DBA.SYS_BLOG_INFO as bi on (BI_BLOG_ID =
B_BLOG_ID)
  left outer join DB.DBA.SYS_USERS as u on (U_ID = BI_OWNER)
;

create view BLOG.DBA.SYS_BLOG_COMMENT_FULL as
select
  BM_ID,        -- Id of a comment
  BM_TITLE,     -- Comment title
  BM_COMMENT,   -- Text of the comment
  BM_NAME,      -- Author name
  BM_E_MAIL,    -- Author e-mail
  BM_TS,        -- Creation time
```

```
  BM_IS_PUB,    -- Whether the comment message is public or
private
  B_BLOG_ID, B_CONTENT, B_TITLE, B_POST_ID, B_TS, B_MODIFIED,
  BI_WAI_NAME,
  U_ID, U_NAME, U_E_MAIL, U_HOME
from
  BLOG.DBA.BLOG_COMMENTS bm
  left outer join BLOG.DBA.SYS_BLOGS as b on (BM_BLOG_ID =
B_BLOG_ID and BM_POST_ID = B_POST_ID)
  left outer join BLOG.DBA.SYS_BLOG_INFO as bi on (BI_BLOG_ID =
B_BLOG_ID)
  left outer join DB.DBA.SYS_USERS as u on (U_ID = BI_OWNER)
;

create view BLOG.DBA.SYS_BLOG_CATEGORY_FULL as
select
  MTC_ID,       -- ID if a category
  MTC_BLOG_ID,  -- Id of a blog
  MTC_NAME,     -- Category name
  BI_WAI_NAME,  -- Blog application instance name
  U_ID, U_NAME, U_E_MAIL, U_HOME
from
  BLOG.DBA.MTYPE_CATEGORIES as mtc
  left outer join BLOG.DBA.SYS_BLOG_INFO as bi on (MTC_BLOG_ID =
BI_BLOG_ID)
  left outer join DB.DBA.SYS_USERS as u on (U_ID = BI_OWNER)
;

create view BLOG.DBA.SYS_BLOG_POST_CATEGORY_FULL as
select
  B_BLOG_ID, B_CONTENT, B_TITLE, B_POST_ID, B_TS, B_USER_ID,
B_MODIFIED,
  MTC_ID, MTC_BLOG_ID, MTC_NAME,
  BI_WAI_NAME,
  U_ID, U_NAME, U_E_MAIL, U_HOME
from
  BLOG.DBA.MTYPE_CATEGORIES as mtc
  join BLOG.DBA.SYS_BLOGS as b on (B_BLOG_ID = MTC_BLOG_ID)
  left outer join BLOG.DBA.SYS_BLOG_INFO as bi on (B_BLOG_ID =
BI_BLOG_ID)
  left outer join DB.DBA.SYS_USERS as u on (U_ID = BI_OWNER)
;
```

Now for any combination of tables we have a view that consists of tables with proper join conditions. Virtuoso has a good chance to create good execution plans for joins of such views because a 'view' in Virtuoso is expanded like a macro before any join order decisions are made. When a FROM clause of a SQL SELECT statement contains a view, the occurrence of a view in FROM clause is replaced with the full SELECT statement of the view body. So when a view P is a join of tables of P1 and P2, view Q is a join of Q1 and Q2 and the query is a join of P and Q then the compilation 'flattens' it and runs same way as if the query were a plain join of P1, P2, Q1 and Q2.

The created views use demonstrate an optimization technique where LEFT OUTER JOINS are

used instead of INNER JOINS as a way of eliminating the generation of redundant table joins. Thus, If we define a VIEW as: Table A left OUTER JOIN Table B and the JOIN condition has equalities for all columns matching the unique key of Table B (e.g., primary key columns), and only columns of Table A are queried from the VIEW, then the join with B will neither alter returned columns nor change the number of resulting rows, implying that they can be omitted. This let us declare a single VIEW using: "COMMENTS left join BLOGS left join BLOG_INFO left join SYS_USERS" avoiding the creation of redundant: "COMMENTS left join BLOGS" and "COMMENTS left join BLOGS left join BLOG_INFO".

```
sparql
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix atom: <http://atomowl.org/ontologies/atomrdf#>
prefix odsWeblog: <http://www.openlinksw.com/dataspace-weblog#>
create iri class odsWeblog:feed_iri
"http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U" (
  in memb varchar not null, in inst varchar not null) .
create iri class odsWeblog:feed_inst_iri
"http://^{URIQADefaultHost}^/dataspace/%U/weblog-inst/%U" (
  in memb varchar not null, in inst varchar not null) .
create iri class odsWeblog:post_iri
"http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U/%U" (
  in memb varchar not null, in inst varchar not null, in post
varchar not null) .
create iri class odsWeblog:post_inst_iri
"http://^{URIQADefaultHost}^/dataspace/%U/weblog-post-inst/%U/%U"
(
  in memb varchar not null, in inst varchar not null, in post
varchar not null) .
create iri class odsWeblog:text_iri
"http://^{URIQADefaultHost}^/dataspace/%U/weblog-text/%U/%U" (
  in memb varchar not null, in inst varchar not null, in post
varchar not null) .
create iri class odsWeblog:cmt_iri
"http://^{URIQADefaultHost}^/dataspace/%U/weblog/%U/%U/%d" (
  in memb varchar not null, in inst varchar not null, in post
varchar not null, in cmtid integer) .
create iri class odsWeblog:cat_iri
"http://^{URIQADefaultHost}^/dataspace/%U/weblog-cat/%U/%U" (
  in memb varchar not null, in inst varchar not null, in catid
varchar not null) .
;
```

Note that the AtomOWL mapping for blogs will use IRI classes oplsioc:user_iri, oplsioc:group_iri as they are already declared in the example above. As soon as views are prepared and IRI classes are tested to ensure that composed IRIs match to IRIs of actual web pages the rest of quad map declaration is quite straightforward and every single mapping is as simple as in previous 'users and groups' use case.

```
sparql
```

```
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
prefix sioc: <http://rdfs.org/sioc/ns#>
prefix atom: <http://atomowl.org/ontologies/atomrdf#>
prefix odsWeblog: <http://www.openlinksw.com/dataspace-weblog#>
create quad storage virtrdf:ODS
  {
    create virtrdf:Weblog2 as graph iri
("http://^{URIQADefaultHost}^/dataspace/weblog2") option
(exclusive)
      {
        # Feed
        odsWeblog:feed_iri (BLOG.DBA.SYS_BLOG_INFO_FULL.U_NAME,
BI_WAI_NAME)
            a atom:Feed ;
            atom:feedInstance odsWeblog:feed_inst_iri
(BLOG.DBA.SYS_BLOG_INFO_FULL.U_NAME, BI_WAI_NAME) .

        # Feed Instance
        odsWeblog:feed_inst_iri
(BLOG.DBA.SYS_BLOG_INFO_FULL.U_NAME, BI_WAI_NAME)
            a atom:FeedInstance ;
            atom:title BI_WAI_NAME .
        odsWeblog:feed_inst_iri
(BLOG.DBA.SYS_BLOG_CATEGORY_FULL.U_NAME, BI_WAI_NAME)
            atom:category odsWeblog:cat_iri
(BLOG.DBA.SYS_BLOG_CATEGORY_FULL.U_NAME, BI_WAI_NAME, MTC_ID) .
            # The following mappings are excluded from the
example: atom:contributor, atom:extension, atom:generator,
            # atom:icon, atom:link, atom:logo, atom:rights,
atom:subtitle, atom:updated.

        # Post
        odsWeblog:post_iri (BLOG.DBA.SYS_BLOG_FULL.U_NAME,
BI_WAI_NAME, B_POST_ID)
            a atom:Entry ;
            atom:entryInstance odsWeblog:post_inst_iri
(BLOG.DBA.SYS_BLOG_FULL.U_NAME, BI_WAI_NAME, B_POST_ID) .

        # Post Instance
        odsWeblog:post_inst_iri (BLOG.DBA.SYS_BLOG_FULL.U_NAME,
BI_WAI_NAME, B_POST_ID)
            a atom:EntryInstance ;
            atom:author oplsioc:user_iri (U_ID) ;
            atom:containingFeed odsWeblog:feed_inst_iri
(BLOG.DBA.SYS_BLOG_FULL.U_NAME, BI_WAI_NAME) ;
            atom:content odsWeblog:text_iri (U_NAME, BI_WAI_NAME,
B_POST_ID) ;
            atom:contributor oplsioc:user_iri (B_USER_ID) ;
            atom:published B_TS ;
            atom:title B_TITLE ;
            atom:updated B_MODIFIED .
        odsWeblog:post_inst_iri
(BLOG.DBA.SYS_BLOG_POST_CATEGORY_FULL.U_NAME, BI_WAI_NAME,
B_POST_ID)
            atom:category
```

```
              odsWeblog:cat_iri
(BLOG.DBA.SYS_BLOG_POST_CATEGORY_FULL.U_NAME, BI_WAI_NAME,
MTC_ID) .
            # The following mappings are excluded from the
example: atom:extension, atom:link, atom:rights
            # atom:sourceFeed atom:summary.

        # Text
        odsWeblog:text_iri (BLOG.DBA.SYS_BLOG_FULL.U_NAME,
BI_WAI_NAME, B_POST_ID)
            a atom:Text ;
            rdf:value B_CONTENT .
            # The following mappings are excluded from the
example: atom:contentBase, atom:contentLang,
            # atom:contentSource, atom:contentType, atom:mimeType

        # Comment
        odsWeblog:cmt_iri (BLOG.DBA.SYS_BLOG_COMMENT_FULL.U_NAME,
BI_WAI_NAME, B_POST_ID, BM_ID)
            a atom:EntryInstance where (^{alias}^.BM_IS_PUB = 1)
.

        odsWeblog:cat_iri
(BLOG.DBA.SYS_BLOG_CATEGORY_FULL.U_NAME, BI_WAI_NAME, MTC_ID)
            a atom:Category ;
            atom:categoryLabel MTC_NAME .
            # atom:categoryScheme and atom:categoryTerm are
excluded from the example, as well as
            # a view that is specific for them.
      }
  }
;
```

The disadvantage of current SPARQL implementation is that it can create enormous intermediate SQL queries. A relatively simple SPARQL query like five triple patterns where every IRI reference comes from a join of four tables results in a query with tens of joins and the full optimization before the first run takes seconds of CPU time. Additional problem is a risk of running out of optimization time limit so the server will stop optimization before it reached the optimal plan.

We are presently developing a rich syntax of quad map declarations, that allows the application developer to describe join rules in quad mapping.

- There will be no need in explicit CREATE VIEW statements.
- The SPARQL optimizer will never create redundant joins.
- The SPARQL optimizer will be able to prove more about ranges of SPARQL variables and better eliminate joins of triple patterns that can not produce any bindings.
- The generator of an intermediate SQL code will create the smallest possible query and minimize the SQL optimization time.

Mapping Virtuoso Demo Database to a Custom Ontology

for the Northwind Data Model

Consider a Virtuoso Demo Database.

This example uses the following tables from the Virtuoso Demo Database SQL Schema: Customers, Orders, Order Details, Products, Product Categories, Employee, Region, Country, Province.

Bearing in mind the similarities between the SQL schemas of Virtuoso's Demo database and Microsoft's Northwind Database (used by ACCESS and Microsoft SQL Server), we have chosen to broaden the scope of this example by deriving an OWL ontology from a Conceptual Data Schema Language (CSDL) file that uses XML Schema to conceptually model the Northwind Schema. The aforementioned Ontology is then used as the basis for declaring mappings between the SQL Database and the OWL Ontology.

XML Schema for Northwind (CSDL File)

```
<?xml version="1.0" encoding="utf-8"?>
<Mapping xmlns:cs="urn:schemas-microsoft-
com:windows:storage:mapping:CS" cs:Space="C-S"
xmlns="urn:schemas-microsoft-com:windows:storage:mapping:CS">
  <cs:EntityContainerMapping cs:StorageEntityContainer="dbo"
cs:CdmEntityContainer="NorthwindEntities">
    <cs:EntitySetMapping cs:Name="Categories"
cs:TableName="Categories" cs:TypeName="NorthwindModel.Category">
      <cs:ScalarProperty cs:Name="CategoryID"
cs:ColumnName="CategoryID" />
      <cs:ScalarProperty cs:Name="CategoryName"
cs:ColumnName="CategoryName" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Customers"
cs:TableName="Customers" cs:TypeName="NorthwindModel.Customer">
      <cs:ScalarProperty cs:Name="CustomerID"
cs:ColumnName="CustomerID" />
      <cs:ScalarProperty cs:Name="CompanyName"
cs:ColumnName="CompanyName" />
      <cs:ScalarProperty cs:Name="ContactName"
cs:ColumnName="ContactName" />
      <cs:ScalarProperty cs:Name="ContactTitle"
cs:ColumnName="ContactTitle" />
      <cs:ScalarProperty cs:Name="Address"
cs:ColumnName="Address" />
      <cs:ScalarProperty cs:Name="City" cs:ColumnName="City" />
      <cs:ScalarProperty cs:Name="Region" cs:ColumnName="Region"
/>
      <cs:ScalarProperty cs:Name="PostalCode"
cs:ColumnName="PostalCode" />
      <cs:ScalarProperty cs:Name="Country"
cs:ColumnName="Country" />
      <cs:ScalarProperty cs:Name="Phone" cs:ColumnName="Phone" />
      <cs:ScalarProperty cs:Name="Fax" cs:ColumnName="Fax" />
```

```
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Employees"
cs:TableName="Employees" cs:TypeName="NorthwindModel.Employee">
      <cs:ScalarProperty cs:Name="EmployeeID"
cs:ColumnName="EmployeeID" />
      <cs:ScalarProperty cs:Name="LastName"
cs:ColumnName="LastName" />
      <cs:ScalarProperty cs:Name="FirstName"
cs:ColumnName="FirstName" />
      <cs:ScalarProperty cs:Name="Title" cs:ColumnName="Title" />
      <cs:ScalarProperty cs:Name="TitleOfCourtesy"
cs:ColumnName="TitleOfCourtesy" />
      <cs:ScalarProperty cs:Name="BirthDate"
cs:ColumnName="BirthDate" />
      <cs:ScalarProperty cs:Name="HireDate"
cs:ColumnName="HireDate" />
      <cs:ScalarProperty cs:Name="Address"
cs:ColumnName="Address" />
      <cs:ScalarProperty cs:Name="City" cs:ColumnName="City" />
      <cs:ScalarProperty cs:Name="Region" cs:ColumnName="Region"
/>
      <cs:ScalarProperty cs:Name="PostalCode"
cs:ColumnName="PostalCode" />
      <cs:ScalarProperty cs:Name="Country"
cs:ColumnName="Country" />
      <cs:ScalarProperty cs:Name="HomePhone"
cs:ColumnName="HomePhone" />
      <cs:ScalarProperty cs:Name="Extension"
cs:ColumnName="Extension" />
      <cs:ScalarProperty cs:Name="PhotoPath"
cs:ColumnName="PhotoPath" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="OrderLines" cs:TableName="Order
Details" cs:TypeName="NorthwindModel.OrderLine">
      <cs:ScalarProperty cs:Name="OrderID"
cs:ColumnName="OrderID" />
      <cs:ScalarProperty cs:Name="ProductID"
cs:ColumnName="ProductID" />
      <cs:ScalarProperty cs:Name="UnitPrice"
cs:ColumnName="UnitPrice" />
      <cs:ScalarProperty cs:Name="Quantity"
cs:ColumnName="Quantity" />
      <cs:ScalarProperty cs:Name="Discount"
cs:ColumnName="Discount" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Orders" cs:TableName="Orders"
cs:TypeName="NorthwindModel.Order">
      <cs:ScalarProperty cs:Name="OrderID"
cs:ColumnName="OrderID" />
      <cs:ScalarProperty cs:Name="OrderDate"
cs:ColumnName="OrderDate" />
      <cs:ScalarProperty cs:Name="RequiredDate"
cs:ColumnName="RequiredDate" />
      <cs:ScalarProperty cs:Name="ShippedDate"
cs:ColumnName="ShippedDate" />
```

```
      <cs:ScalarProperty cs:Name="Freight"
cs:ColumnName="Freight" />
      <cs:ScalarProperty cs:Name="ShipName"
cs:ColumnName="ShipName" />
      <cs:ScalarProperty cs:Name="ShipAddress"
cs:ColumnName="ShipAddress" />
      <cs:ScalarProperty cs:Name="ShipCity"
cs:ColumnName="ShipCity" />
      <cs:ScalarProperty cs:Name="ShipRegion"
cs:ColumnName="ShipRegion" />
      <cs:ScalarProperty cs:Name="ShipPostalCode"
cs:ColumnName="ShipPostalCode" />
      <cs:ScalarProperty cs:Name="ShipCountry"
cs:ColumnName="ShipCountry" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Products"
cs:TableName="Products" cs:TypeName="NorthwindModel.Product">
      <cs:ScalarProperty cs:Name="ProductID"
cs:ColumnName="ProductID" />
      <cs:ScalarProperty cs:Name="ProductName"
cs:ColumnName="ProductName" />
      <cs:ScalarProperty cs:Name="QuantityPerUnit"
cs:ColumnName="QuantityPerUnit" />
      <cs:ScalarProperty cs:Name="UnitPrice"
cs:ColumnName="UnitPrice" />
      <cs:ScalarProperty cs:Name="UnitsInStock"
cs:ColumnName="UnitsInStock" />
      <cs:ScalarProperty cs:Name="UnitsOnOrder"
cs:ColumnName="UnitsOnOrder" />
      <cs:ScalarProperty cs:Name="ReorderLevel"
cs:ColumnName="ReorderLevel" />
      <cs:ScalarProperty cs:Name="Discontinued"
cs:ColumnName="Discontinued" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Regions" cs:TableName="Region"
cs:TypeName="NorthwindModel.Region">
      <cs:ScalarProperty cs:Name="RegionID"
cs:ColumnName="RegionID" />
      <cs:ScalarProperty cs:Name="RegionDescription"
cs:ColumnName="RegionDescription" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Shippers"
cs:TableName="Shippers" cs:TypeName="NorthwindModel.Shipper">
      <cs:ScalarProperty cs:Name="ShipperID"
cs:ColumnName="ShipperID" />
      <cs:ScalarProperty cs:Name="CompanyName"
cs:ColumnName="CompanyName" />
      <cs:ScalarProperty cs:Name="Phone" cs:ColumnName="Phone" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Suppliers"
cs:TableName="Suppliers" cs:TypeName="NorthwindModel.Supplier">
      <cs:ScalarProperty cs:Name="SupplierID"
cs:ColumnName="SupplierID" />
      <cs:ScalarProperty cs:Name="CompanyName"
cs:ColumnName="CompanyName" />
```

```
      <cs:ScalarProperty cs:Name="ContactName"
cs:ColumnName="ContactName" />
      <cs:ScalarProperty cs:Name="ContactTitle"
cs:ColumnName="ContactTitle" />
      <cs:ScalarProperty cs:Name="Address"
cs:ColumnName="Address" />
      <cs:ScalarProperty cs:Name="City" cs:ColumnName="City" />
      <cs:ScalarProperty cs:Name="Region" cs:ColumnName="Region"
/>
      <cs:ScalarProperty cs:Name="PostalCode"
cs:ColumnName="PostalCode" />
      <cs:ScalarProperty cs:Name="Country"
cs:ColumnName="Country" />
      <cs:ScalarProperty cs:Name="Phone" cs:ColumnName="Phone" />
      <cs:ScalarProperty cs:Name="Fax" cs:ColumnName="Fax" />
    </cs:EntitySetMapping>
    <cs:EntitySetMapping cs:Name="Territories"
cs:TableName="Territories"
cs:TypeName="NorthwindModel.Territory">
      <cs:ScalarProperty cs:Name="TerritoryID"
cs:ColumnName="TerritoryID" />
      <cs:ScalarProperty cs:Name="TerritoryDescription"
cs:ColumnName="TerritoryDescription" />
    </cs:EntitySetMapping>
    <cs:AssociationSetMapping cs:Name="EmployeeTerritories"
cs:TypeName="NorthwindModel.EmployeeTerritories"
cs:TableName="EmployeeTerritories">
      <cs:EndProperty cs:Name="Employees">
        <cs:ScalarProperty cs:ColumnName="EmployeeID"
cs:Name="EmployeeID"/>
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Territories">
        <cs:ScalarProperty cs:ColumnName="TerritoryID"
cs:Name="TerritoryID"/>
      </cs:EndProperty>
    </cs:AssociationSetMapping>
    <cs:AssociationSetMapping cs:Name="FK_Employees_Employees"
cs:TypeName="NorthwindModel.FK_Employees_Employees"
cs:TableName="Employees">
      <cs:EndProperty cs:Name="Manager">
        <cs:ScalarProperty cs:Name="EmployeeID"
cs:ColumnName="ReportsTo" />
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Employees">
        <cs:ScalarProperty cs:Name="EmployeeID"
cs:ColumnName="EmployeeID" />
      </cs:EndProperty>
      <cs:Condition cs:ColumnName="ReportsTo" cs:IsNull="false"
/>
    </cs:AssociationSetMapping>
    <cs:AssociationSetMapping cs:Name="FK_Orders_Customers"
cs:TypeName="NorthwindModel.FK_Orders_Customers"
cs:TableName="Orders">
      <cs:EndProperty cs:Name="Customer">
        <cs:ScalarProperty cs:Name="CustomerID"
```

```
cs:ColumnName="CustomerID" />
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Orders">
        <cs:ScalarProperty cs:Name="OrderID"
cs:ColumnName="OrderID" />
      </cs:EndProperty>
      <cs:Condition cs:ColumnName="CustomerID" cs:IsNull="false"
/>
    </cs:AssociationSetMapping>
    <cs:AssociationSetMapping cs:Name="FK_Orders_Employees"
cs:TypeName="NorthwindModel.FK_Orders_Employees"
cs:TableName="Orders">
      <cs:EndProperty cs:Name="Employee">
        <cs:ScalarProperty cs:Name="EmployeeID"
cs:ColumnName="EmployeeID" />
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Orders">
        <cs:ScalarProperty cs:Name="OrderID"
cs:ColumnName="OrderID" />
      </cs:EndProperty>
      <cs:Condition cs:ColumnName="EmployeeID" cs:IsNull="false"
/>
    </cs:AssociationSetMapping>
    <cs:AssociationSetMapping cs:Name="FK_Orders_Shippers"
cs:TypeName="NorthwindModel.FK_Orders_Shippers"
cs:TableName="Orders">
      <cs:EndProperty cs:Name="Shipper">
        <cs:ScalarProperty cs:Name="ShipperID"
cs:ColumnName="ShipVia" />
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Orders">
        <cs:ScalarProperty cs:Name="OrderID"
cs:ColumnName="OrderID" />
      </cs:EndProperty>
      <cs:Condition cs:ColumnName="ShipVia" cs:IsNull="false" />
    </cs:AssociationSetMapping>
    <cs:AssociationSetMapping cs:Name="FK_Products_Categories"
cs:TypeName="NorthwindModel.FK_Products_Categories"
cs:TableName="Products">
      <cs:EndProperty cs:Name="Category">
        <cs:ScalarProperty cs:Name="CategoryID"
cs:ColumnName="CategoryID" />
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Products">
        <cs:ScalarProperty cs:Name="ProductID"
cs:ColumnName="ProductID" />
      </cs:EndProperty>
      <cs:Condition cs:ColumnName="CategoryID" cs:IsNull="false"
/>
    </cs:AssociationSetMapping>
    <cs:AssociationSetMapping cs:Name="FK_Products_Suppliers"
cs:TypeName="NorthwindModel.FK_Products_Suppliers"
cs:TableName="Products">
      <cs:EndProperty cs:Name="Supplier">
        <cs:ScalarProperty cs:Name="SupplierID"
```

```
cs:ColumnName="SupplierID" />
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Products">
        <cs:ScalarProperty cs:Name="ProductID"
cs:ColumnName="ProductID" />
      </cs:EndProperty>
      <cs:Condition cs:ColumnName="SupplierID" cs:IsNull="false"
/>
    </cs:AssociationSetMapping>
    <cs:AssociationSetMapping cs:Name="FK_Territories_Region"
cs:TypeName="NorthwindModel.FK_Territories_Region"
cs:TableName="Territories">
      <cs:EndProperty cs:Name="Region">
        <cs:ScalarProperty cs:Name="RegionID"
cs:ColumnName="RegionID" />
      </cs:EndProperty>
      <cs:EndProperty cs:Name="Territories">
        <cs:ScalarProperty cs:Name="TerritoryID"
cs:ColumnName="TerritoryID" />
      </cs:EndProperty>
      <cs:Condition cs:ColumnName="RegionID" cs:IsNull="false" />
    </cs:AssociationSetMapping>
  </cs:EntityContainerMapping>
</Mapping>
```

OWL Ontology for Northwind (in N3)

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix virtrdf: <http://www.openlinksw.com/schemas/virtrdf#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix northwind: <http://www.openlinksw.com/schemas/northwind#>
.

northwind: rdf:type owl:Ontology ;
        rdfs:label "Northwind" ;
        rdfs:comment "Northwind database classes and properties"
;
        virtrdf:catName "Northwind" ;
        virtrdf:version "1.00" .
northwind:Product rdf:type rdfs:Class ;
        rdfs:label "Product" .
northwind:has_category rdf:type rdf:Property ;
        rdfs:range northwind:Category ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "Category" .
northwind:has_supplier rdf:type rdf:Property ;
        rdfs:range northwind:Supplier ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "Supplier" .
northwind:ProductName rdf:type rdf:Property ;
```

```
        rdfs:range xsd:string ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "ProductName" .
northwind:QuantityPerUnit rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "QuantityPerUnit" .
northwind:UnitPrice rdf:type rdf:Property ;
        rdfs:range xsd:double ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "UnitPrice" .
northwind:UnitsInStock rdf:type rdf:Property ;
        rdfs:range xsd:integer ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "UnitsInStock" .
northwind:UnitsOnOrder rdf:type rdf:Property ;
        rdfs:range xsd:integer ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "UnitsOnOrder" .
northwind:ReorderLevel rdf:type rdf:Property ;
        rdfs:range xsd:integer ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "ReorderLevel" .
northwind:Discontinued rdf:type rdf:Property ;
        rdfs:range xsd:integer ;
        rdfs:domain northwind:Product ;
        rdfs:cardinality "1" ;
        rdfs:label "Discontinued" .
northwind:Supplier rdf:type rdfs:Class ;
        rdfs:label "Supplier" .
northwind:CompanyName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "CompanyName" .
northwind:ContactName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "ContactName" .
northwind:ContactTitle rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "ContactTitle" .
northwind:Address rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
```

```
        rdfs:label "Address" .
northwind:City rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "City" .
northwind:Region rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "Region" .
northwind:PostalCode rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "PostalCode" .
northwind:Country rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "Country" .
northwind:Phone rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "Phone" .
northwind:Fax rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "Fax" .
northwind:HomePage rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Supplier ;
        rdfs:cardinality "1" ;
        rdfs:label "HomePage" .
northwind:Category rdf:type rdfs:Class ;
        rdfs:label "Category" .
northwind:CategoryName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Category ;
        rdfs:cardinality "1" ;
        rdfs:label "CategoryName" .
northwind:Description rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Category ;
        rdfs:cardinality "1" ;
        rdfs:label "Description" .
northwind:Shipper rdf:type rdfs:Class ;
        rdfs:label "Shipper" .
northwind:CompanyName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Shipper ;
        rdfs:cardinality "1" ;
        rdfs:label "CompanyName" .
```

```
northwind:Phone rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Shipper ;
        rdfs:cardinality "1" ;
        rdfs:label "Phone" .
northwind:Customer rdf:type rdfs:Class ;
        rdfs:label "Customer" .
northwind:CompanyName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "CompanyName" .
northwind:ContactName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "ContactName" .
northwind:ContactTitle rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:donorthwind:main northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "ContactTitle" .
northwind:Address rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "Address" .
northwind:City rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "City" .
northwind:Region rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "Region" .
northwind:PostalCode rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "PostalCode" .
northwind:Country rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "Country" .
northwind:Phone rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
        rdfs:cardinality "1" ;
        rdfs:label "Phone" .
northwind:Fax rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Customer ;
```

```
        rdfs:cardinality "1" ;
        rdfs:label "Fax" .
northwind:Employee rdf:type rdfs:Class ;
        rdfs:label "Employee" .
northwind:LastName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "LastName" .
northwind:FirstName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "FirstName" .
northwind:Title rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "Title" .
northwind:TitleOfCourtesy rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "TitleOfCourtesy" .
northwind:BirthDate rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "BirthDate" .
northwind:HireDate rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "HireDate" .
northwind:Address rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "Address" .
northwind:City rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "City" .
northwind:Region rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "Region" .
northwind:PostalCode rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "PostalCode" .
northwind:Country rdf:type rdf:Property ;
```

```
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "Country" .
northwind:HomePhone rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "HomePhone" .
northwind:Extension rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "Extension" .
northwind:Notes rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "Notes" .
northwind:ReportsTo rdf:type rdf:Property ;
        rdfs:range xsd:integer ;
        rdfs:domain northwind:Employee ;
        rdfs:cardinality "1" ;
        rdfs:label "ReportsTo" .
northwind:Order rdf:type rdfs:Class ;
        rdfs:label "Order" .
northwind:has_customer rdf:type rdf:Property ;
        rdfs:range northwind:Customer ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "Customer" .
northwind:has_employee rdf:type rdf:Property ;
        rdfs:range northwind:Employee ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "Employee" .
northwind:OrderDate rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "OrderDate" .
northwind:RequiredDate rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "RequiredDate" .
northwind:ShippedDate rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "ShippedDate" .
northwind:order_ship_via rdf:type rdf:Property ;
        rdfs:range northwind:Shipper ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
```

```
        rdfs:label "Shipper" .
northwind:Freight rdf:type rdf:Property ;
        rdfs:range xsd:double ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "Freight" .
northwind:ShipName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "ShipName" .
northwind:ShipAddress rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "ShipAddress" .
northwind:ShipCity rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "ShipCity" .
northwind:ShipRegion rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "ShipRegion" .
northwind:ShipPostalCode rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "ShipPostalCode" .
northwind:ShipCountry rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Order ;
        rdfs:cardinality "1" ;
        rdfs:label "ShipCountry" .
northwind:OrderLine rdf:type rdfs:Class ;
        rdfs:label "OrderLine" .
northwind:has_order_id rdf:type rdf:Property ;
        rdfs:range northwind:Order ;
        rdfs:domain northwind:OrderLine;
        rdfs:cardinality "1" ;
        rdfs:label "Order" .
northwind:has_product_id rdf:type rdf:Property ;
        rdfs:range northwind:Product ;
        rdfs:domain northwind:OrderLine;
        rdfs:cardinality "1" ;
        rdfs:label "Product" .
northwind:UnitPrice rdf:type rdf:Property ;
        rdfs:range xsd:double ;
        rdfs:domain northwind:OrderLine;
        rdfs:cardinality "1" ;
        rdfs:label "UnitPrice" .
northwind:Quantity rdf:type rdf:Property ;
        rdfs:range xsd:integer ;
```

```
        rdfs:domain northwind:OrderLine;
        rdfs:cardinality "1" ;
        rdfs:label "Quantity" .
northwind:Discount rdf:type rdf:Property ;
        rdfs:range xsd:double ;
        rdfs:domain northwind:OrderLine;
        rdfs:cardinality "1" ;
        rdfs:label "Discount" .
northwind:Country rdf:type rdfs:Class ;
        rdfs:label "Country" .
northwind:Name rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "Name" .
northwind:Code rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "Code" .
northwind:SmallFlagDAVResourceName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "SmallFlagDAVResourceName" .
northwind:LargeFlagDAVResourceName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "LargeFlagDAVResourceName" .
northwind:SmallFlagDAVResourceURI rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "SmallFlagDAVResourceURI" .
northwind:LargeFlagDAVResourceURI rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "LargeFlagDAVResourceURI" .
northwind:Lat rdf:type rdf:Property ;
        rdfs:range xsd:double ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "Lat" .
northwind:Lng rdf:type rdf:Property ;
        rdfs:range xsd:double ;
        rdfs:domain northwind:Country ;
        rdfs:cardinality "1" ;
        rdfs:label "Lng" .
northwind:Province rdf:type rdfs:Class ;
        rdfs:label "Province" .
northwind:has_country_code rdf:type rdf:Property ;
        rdfs:range northwind:Country ;
        rdfs:domain northwind:Provinces ;
```

```
        rdfs:cardinality "1" ;
        rdfs:label "Country Code" .
northwind:ProvinceName rdf:type rdf:Property ;
        rdfs:range xsd:string ;
        rdfs:domain northwind:Province ;
        rdfs:cardinality "1" ;
        rdfs:label "ProvinceName" .
```

IRI Generator (Entity ID Constructor) Declarations

```
sparql
prefix demo: <http://www.openlinksw.com/schemas/demo#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
prefix sioc: <http://rdfs.org/sioc/ns#>
create iri class demo:Category
"http://^{URIQADefaultHost}^/demo/Category?id=%d" (in category_id
integer not null) .
create iri class demo:Shipper
"http://^{URIQADefaultHost}^/demo/Shipper?id=%d" (in shipper_id
integer not null) .
create iri class demo:Supplier
"http://^{URIQADefaultHost}^/demo/Supplier?id=%d" (in supplier_id
integer not null) .
create iri class demo:Product
"http://^{URIQADefaultHost}^/demo/Product?id=%d" (in product_id
integer not null) .
create iri class demo:Customer
"http://^{URIQADefaultHost}^/demo/Customer?id=%d" (in customer_id
integer not null) .
create iri class demo:Employee
"http://^{URIQADefaultHost}^/demo/Employee?id=%d" (in employee_id
integer not null) .
create iri class demo:Order
"http://^{URIQADefaultHost}^/demo/Order?id=%d" (in order_id
integer not null) .
create iri class demo:OrderLine
"http://^{URIQADefaultHost}^/demo/OrderLine?id=%d&prod_id=%d" (in
order_id integer not null, in product_id integer not null) .
create iri class demo:Province
"http://^{URIQADefaultHost}^/demo/Province?
country=%s&province=%s" (in country_name varchar not null, in
province_name varchar not null) .
create iri class demo:Country
"http://^{URIQADefaultHost}^/demo/Country?country=%s" (in
country_name varchar not null) .
;
```

SQL to RDF Mapping (Quad Patterns)

```
sparql

prefix demo: <http://www.openlinksw.com/schemas/demo#>
prefix oplsioc: <http://www.openlinksw.com/schemas/oplsioc#>
```

```
prefix sioc: <http://rdfs.org/sioc/ns#>

create quad storage virtrdf:Northwind
from Demo.demo.Products as products
from Demo.demo.Suppliers as suppliers
from Demo.demo.Shippers as shippers
from Demo.demo.Categories as categories
from Demo.demo.Customers as customers
from Demo.demo.Employees as employees
from Demo.demo.Orders as orders
from Demo.demo.Order_Details as order_lines
from Demo.demo.Countries as countries
from Demo.demo.Provinces as provinces
{
        create virtrdf:Demo as graph iri
("http://^{URIQADefaultHost}^/demo") option (exclusive)
        {
                demo:Product (products.ProductID)
                        a demo:Product
                                as virtrdf:Product-ProductID ;
                        demo:has_category demo:Category
(products.CategoryID)
                                as virtrdf:Product-
product_has_category ;
                        demo:has_supplier demo:Supplier
(products.SupplierID)
                                as virtrdf:Product-
product_has_supplier ;
                        demo:ProductName products.ProductName
                                as virtrdf:Product-
name_of_product ;
                        demo:QuantityPerUnit
products.QuantityPerUnit
                                as virtrdf:Product-
quantity_per_unit ;
                        demo:UnitPrice products.UnitPrice
                                as virtrdf:Product-unit_price ;
                        demo:UnitsInStock products.UnitsInStock
                                as virtrdf:Product-units_in_stock
;
                        demo:UnitsOnOrder products.UnitsOnOrder
                                as virtrdf:Product-units_on_order
;
                        demo:ReorderLevel products.ReorderLevel
                                as virtrdf:Product-reorder_level
;
                        demo:Discontinued products.Discontinued
                                as virtrdf:Product-discontinued .
                demo:Supplier (suppliers.SupplierID)
                        a demo:Supplier
                                as virtrdf:Supplier-SupplierID ;
                        demo:CompanyName suppliers.CompanyName
                                as virtrdf:Supplier-company_name
;
                        demo:ContactName suppliers.ContactName
```

```
                        as virtrdf:Supplier-contact_name
;
                demo:ContactTitle suppliers.ContactTitle
                        as virtrdf:Supplier-contact_title
;
                demo:Address suppliers.Address
                        as virtrdf:Supplier-address ;
                demo:City suppliers.City
                        as virtrdf:Supplier-city ;
                demo:Region suppliers.Region
                        as virtrdf:Supplier-region ;
                demo:PostalCode suppliers.PostalCode
                        as virtrdf:Supplier-postal_code ;
                demo:Country suppliers.Country
                        as virtrdf:Supplier-country ;
                demo:Phone suppliers.Phone
                        as virtrdf:Supplier-phone ;
                demo:Fax suppliers.Fax
                        as virtrdf:Supplier-fax ;
                demo:HomePage suppliers.HomePage
                        as virtrdf:Supplier-home_page .
        demo:Category (categories.CategoryID)
                a demo:Category
                        as virtrdf:Category-CategoryID ;
                demo:CategoryName categories.CategoryName
                        as virtrdf:Category-home_page ;
                demo:Description categories.Description
                        as virtrdf:Category-description ;
                demo:Picture categories.Picture
                        as virtrdf:Category-picture .
        demo:Shipper (shippers.ShipperID)
                a demo:Shipper
                        as virtrdf:Shipper-ShipperID ;
                demo:CompanyName shippers.CompanyName
                        as virtrdf:Shipper-company_name ;
                demo:Phone shippers.Phone
                        as virtrdf:Shipper-phone .
        demo:Customer (customers.CustomerID)
                a demo:Customer
                        as virtrdf:Customer-CustomerID ;
                demo:CompanyName customers.CompanyName
                        as virtrdf:Customer-company_name
;
                demo:ContactName customers.ContactName
                        as virtrdf:Customer-contact_name
;
                demo:ContactTitle customers.ContactTitle
                        as virtrdf:Customer-contact_title
;
                demo:Address customers.Address
                        as virtrdf:Customer-address ;
                demo:City customers.City
                        as virtrdf:Customer-city ;
                demo:Region customers.Region
                        as virtrdf:Customer-region ;
```

```
                        demo:PostalCode customers.PostalCode
                                as virtrdf:Customer-postal_code ;
                        demo:Country customers.Country
                                as virtrdf:Customer-country ;
                        demo:Phone customers.Phone
                                as virtrdf:Customer-phone ;
                        demo:Fax customers.Fax
                                as virtrdf:Customer-fax .
                demo:Employee (employees.EmployeeID)
                        a demo:Employee
                                as virtrdf:Employee-EmployeeID ;
                        demo:LastName employees.LastName
                                as virtrdf:Employee-last_name ;
                        demo:FirstName employees.FirstName
                                as virtrdf:Employee-first_name ;
                        demo:Title employees.Title
                                as virtrdf:title ;
                        demo:TitleOfCourtesy
employees.TitleOfCourtesy
                                as virtrdf:Employee-
title_of_courtesy ;
                        demo:BirthDate employees.BirthDate
                                as virtrdf:Employee-birth_date ;
                        demo:HireDate employees.HireDate
                                as virtrdf:Employee-hire_date ;
                        demo:Address employees.Address
                                as virtrdf:Employee-address ;
                        demo:City employees.City
                                as virtrdf:Employee-city ;
                        demo:Region employees.Region
                                as virtrdf:Employee-region ;
                        demo:PostalCode employees.PostalCode
                                as virtrdf:Employee-postal_code ;
                        demo:Country employees.Country
                                as virtrdf:Employee-country ;
                        demo:HomePhone employees.HomePhone
                                as virtrdf:Employee-home_phone ;
                        demo:Extension employees.Extension
                                as virtrdf:Employee-extension ;
                        demo:Photo employees.Photo
                                as virtrdf:Employee-photo ;
                        demo:Notes employees.Notes
                                as virtrdf:Employee-notes ;
                        demo:ReportsTo employees.ReportsTo
                                as virtrdf:Employee-reports_to .
                demo:Order (orders.OrderID)
                        a demo:Order
                                as virtrdf:Order-Order ;
                        demo:has_customer demo:Customer
(orders.CustomerID)
                                as virtrdf:Order-
order_has_customer ;
                        demo:has_employee demo:Employee
(orders.EmployeeID)
                                as virtrdf:Order-
```

```
order_has_employee ;
                        demo:OrderDate orders.OrderDate
                                as virtrdf:Order-order_date ;
                        demo:RequiredDate orders.RequiredDate
                                as virtrdf:Order-required_date ;
                        demo:ShippedDate orders.ShippedDate
                                as virtrdf:Order-shipped_date ;
                        demo:order_ship_via demo:Shipper
(orders.ShipVia)
                                as virtrdf:Order-order_ship_via ;
                        demo:Freight orders.Freight
                                as virtrdf:Order-freight ;
                        demo:ShipName orders.ShipName
                                as virtrdf:Order-ship_name ;
                        demo:ShipAddress orders.ShipAddress
                                as virtrdf:Order-ship_address ;
                        demo:ShipCity orders.ShipCity
                                as virtrdf:Order-ship_city ;
                        demo:ShipRegion orders.ShipRegion
                                as virtrdf:Order-ship_region ;
                        demo:ShipPostal_code
orders.ShipPostalCode
                                as virtrdf:Order-ship_postal_code
;
                        demo:ShipCountry orders.ShipCountry
                                as virtrdf:ship_country .
                demo:OrderLine (order_lines.OrderID,
order_lines.ProductID)
                        a demo:OrderLine
                                as virtrdf:OrderLine-OrderLines ;
                        demo:has_order_id demo:Order
(order_lines.OrderID)
                                as
virtrdf:order_lines_has_order_id ;
                        demo:has_product_id demo:Product
(order_lines.ProductID)
                                as
virtrdf:order_lines_has_product_id ;
                        demo:UnitPrice order_lines.UnitPrice
                                as virtrdf:OrderLine-unit_price ;
                        demo:Quantity order_lines.Quantity
                                as virtrdf:OrderLine-quantity ;
                        demo:Discount order_lines.Discount
                                as virtrdf:OrderLine-discount .
                demo:Country (countries.Name)
                        a demo:Country
                                as virtrdf:Country-Name ;
                        demo:Code countries.Code
                                as virtrdf:Country-Code ;
                        demo:SmallFlagDAVResourceName
countries.SmallFlagDAVResourceName
                                as virtrdf:Country-
SmallFlagDAVResourceName ;
                        demo:LargeFlagDAVResourceName
countries.LargeFlagDAVResourceName
```

```
                                as virtrdf:Country-
LargeFlagDAVResourceName ;
                        demo:SmallFlagDAVResourceURI
countries.SmallFlagDAVResourceURI
                                as virtrdf:Country-
SmallFlagDAVResourceURI ;
                        demo:LargeFlagDAVResourceURI
countries.LargeFlagDAVResourceURI
                                as virtrdf:Country-
LargeFlagDAVResourceURI ;
                        demo:Lat countries.Lat
                                as virtrdf:Country-Lat ;
                        demo:Lng countries.Lng
                                as virtrdf:Country-Lng .
                demo:Province (provinces.CountryCode,
provinces.Province)
                        a demo:Province
                                as virtrdf:Province-Provinces ;
                        demo:has_country_code demo:Country
(provinces.CountryCode)
                                as virtrdf:has_country_code ;
                        demo:Province provinces.Province
                                as virtrdf:Province-Province .
        }
}
;
```

Conclusions

We have described how we can arbitrarily map relational database schemas to RDF ontologies en route to generating virtual RDF Data Sets (Graphs) that are then accessible to SPARQL Queries from within SQL or via the SPARQL Query Protocol. All of this is achieved without compromising the inherent flexibility of the RDF data model or the SPARQL Query Language.

It should also be noted that all of the functionality demonstrated also applies Virtual DBMS functionality realm of Virtuoso. Thus, you can now map of 3rd party ODBC or JDBC accessible SQL data to RDF on the fly. Likewise, you can also use the same Virtual DBMS layer to map data exposed via local or 3rd party SOAP or REST based Web Services to RDF.

In addition to providing immense power and flexibility at the data mapping level, we have also paid great attention to the low level optimization of Virtuoso's underlying RDF storage engine (Triple or Quad Store).

Future Research & Development

Future areas of research and development include:

- Supporting declaration of data representation conditioned by the inferred domain of the triple pattern subject.
- Optimized mapping of joins of relational tables without creating redundant views.

- Supporting disjunction.
- Federating SPARQL queries across multiple SPARQL endpoints.

Other SQL to RDF Mapping Projects

- D2RQ Project
- Squirrel Project
- SPASQL Project

References

Relational Databases & The Semantic Web - Tim Berners-Lee

CategoryRDF CategoryAtomOWL CategoryTutorial CategorySQL CategoryOntology CategorySIOC CategoryVOS CategoryVirtuoso CategoryODS