All of the details for the RESTView examples

Malcolm Crowe, 1 January 2021

(In progress: Development has reached the end of section 2)

Pyrrho v7 sees a significant re-implementation of the Pyrrho database engine. The purpose of this document is to explain the details that are relevant to RESTView technology. Several documents in earlier versions of Pyrrho dealt with a number of different use cases. For clarity, and at the expense of length, these are all reviewed in this document.

One of these earlier documents gave details of an example described in an accepted conference paper. A new version of this account is in section 2 below. This dealt with a fairly simple example where the remote database was called A, and illustrated the use of a URL-type syntax for aacessing the remote database. However, the norma use case for RESTView envisages the use of SQL-style syntax for remote queries, so this use case is explored (for the same remote database) in section 1 , which gives full details of the basic RESTView approach for a very simple example. Section 2 returns to the example in the paper and shows the full details for the use of SQL-style remote syntax. Section 3 gives corresponding details for when the URL-style remote syntax is preferred. Section 4 gives a simple demo for the USING option of RESTView. Finally, in section 5, we return to SQL-style remote syntax for a demonstration of how query rewriting optimises filters and joins on remote views.

The following transcripts use the alpha version of Pyrrho v7 dated 25 December 2020, and localhost instead of servA. I have set a debugging -T –D flag on server A so that that we can see the use of RVVs and ETags, and -H on server B so that we can see the HTTP interaction. The name “MAC\Fred” in this document stands for the current Windows user. If just one server is being used, it is possible to provide all three flags -H -T -D.

Section 1: SQL-style remote syntax, with a simple view

After setting up the databases on A and B with B’s views defined, we see the transaction log contents for A and B.

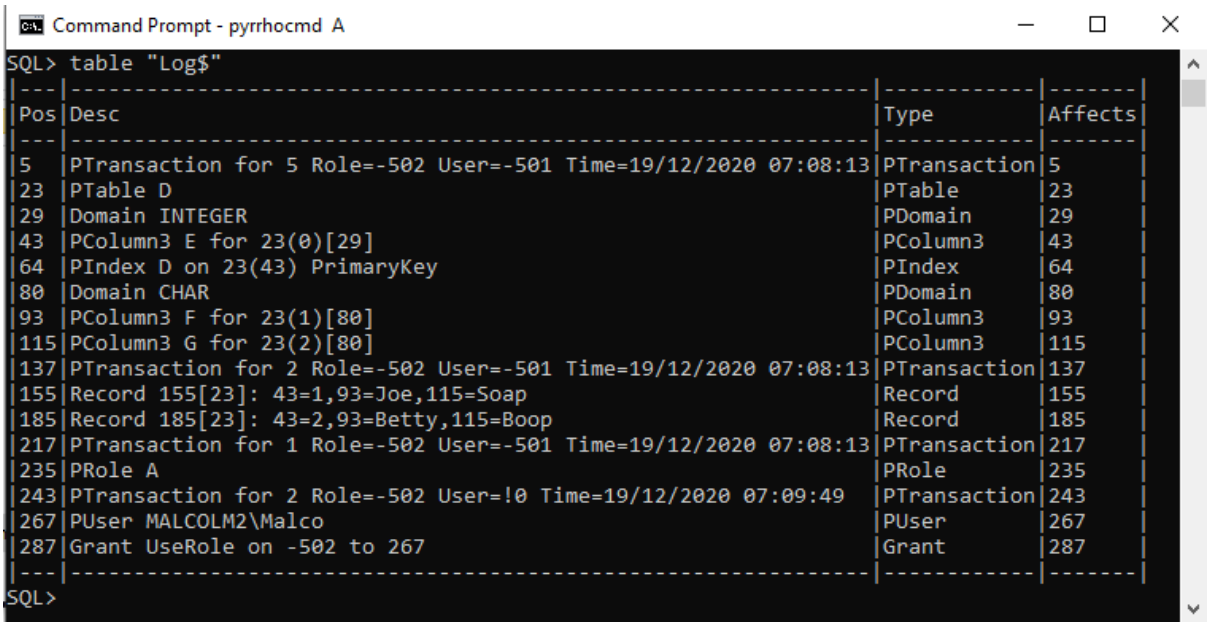
Database A:

create table D (e int primary key, f char, g char)

insert into D values (1,'Joe','Soap'), (2,'Betty','Boop')

create role A

grant A to "*MAC\Fred*"



Database B:

In the paper, database B has the following:

create table H (e int primary key, k char, m int)

insert into H values (1,'Cleaner',12500), (2,'Manager',31400)

[create view W of (e int, f char, g char) as get

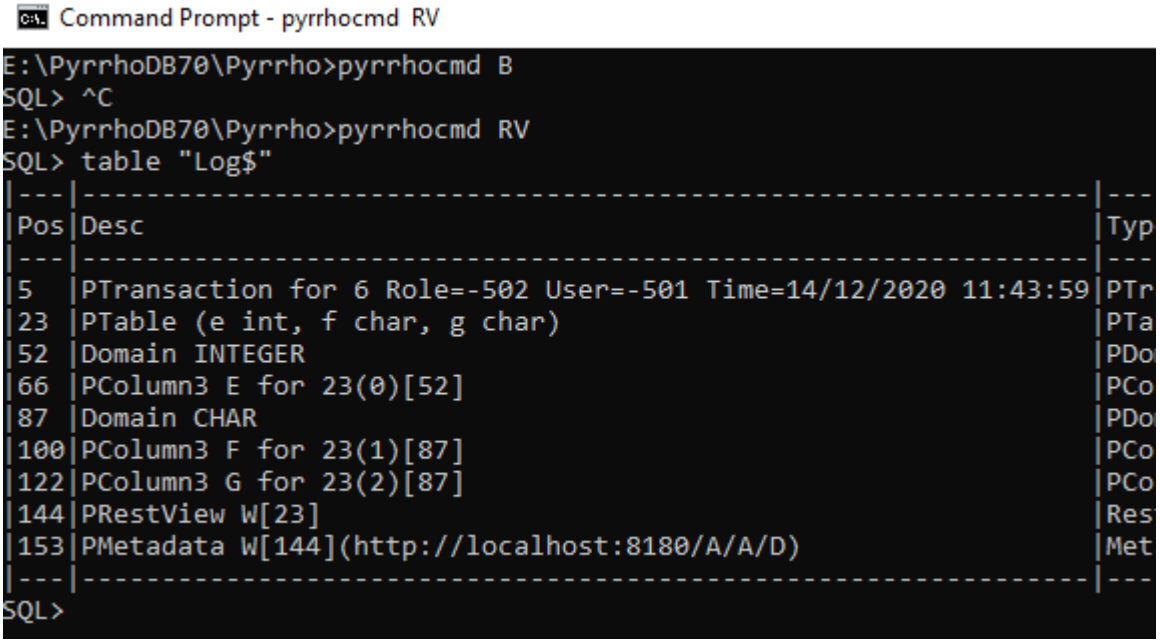
'http://localhost:8180/A/A/D']

create view V as select * from W natural join H

select e,f,m,check from V where e=1

The square brackets here are there because of the embedded newline added in the formatting of the paper. We return to this database in section 2 below. As explained in the paper, normal use does not require the CHECK column: it is here because the paper was discussion ETags and RVV.In this section, we consider an even smaller database with just the RESTView definition and focus on W:

Database RV:



We see at position 153 that the URL http://localhost:8180/A/A/D has been provided in metadata for the view W. W was defined in position 144 in terms of the anonymous structure 23 with columns E, F, G.

Consider what happens when we request “select * from w” on this database.

When this select statement is parsed, an instance of View W 144 is built in the context but moved to “heap uids” %0,.. (because the context might reference the view in more than one place):

```
{(23=Table Name=(e int, f char, g char)  23 Definer=-502 Ppos=23
        Domain TABLE (66,100,122)([66,Domain INTEGER],[100,Domain CHAR],[122,Domain CHAR])
        Enforcement=Select, Insert, Delete, Update KeyCols: ,
  66=TableColumn 66 Definer=-502 Ppos=66 Domain INTEGER Table=23 colDefault TypedValue Null,
  100=TableColumn 100 Definer=-502 Ppos=100 Domain CHAR Table=23 colDefault TypedValue Null,
  122=TableColumn 122 Definer=-502 Ppos=122 Domain CHAR Table=23 colDefault TypedValue Null,
  #7=QuerySpecification #7 RowType:(%2,%3,%4) TableExp ?,
  #8=SqlStar Name=* #8 CONTENT From:#7 CONTENT,
  %0=RestView Name=W %0 Definer=-502 Ppos=144 Query  Ppos: 144 Cols (E=66,F=100,G=122)
        Domain TABLE (66,100,122)([66,Domain INTEGER],[100,Domain CHAR],[122,Domain CHAR])  ViewQry: %8,
  %1=From Name=W %1 RowType:(%2,%3,%4) Target=%0,
  %2=SqlCopy Name=E %2 From:%1 Domain INTEGER copy from 66,
  %3=SqlCopy Name=F %3 From:%1 Domain CHAR copy from 100,
  %4=SqlCopy Name=G %4 From:%1 Domain CHAR copy from 122,
  %5=TableExpression %5 RowType:(%2,%3,%4) Target: %1,
  %6=QuerySpecification %6 RowType:() TableExp %5,
  %7=QueryExpression %7 RowType:() Left: %6 ,
  %8=CursorSpecification %8 RowType:(66,100,122) Source={select * from W} Union: %7)}
```

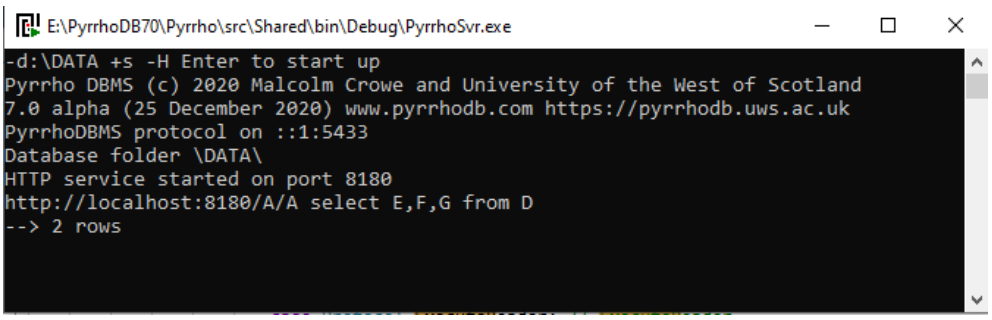
Here we see RestView W moved from its defining position 144 to %0, with columns %2,%3,%4 copied from 66,100,122, and query objects %1,%5,%6,%7,%8 set up as if the query “table w” had just been parsed.

The next step is the creation of rowsets:

```
{(#1=SelectRowSet #7(%2,%3,%4) Finder: (%2=#7[%2],%3=#7[%3],%4=#7[%4]) Source: #10,
  #7=SelectRowSet #7(%2,%3,%4) Finder: (%2=#7[%2],%3=#7[%3],%4=#7[%4]) Source: #10,
  #10=TableExpRowSet #10) Finder: (%2=%5[%2],%3=%5[%3],%4=%5[%4]) Source: %6,
  %1=RestRowSet %1(%2,%3,%4) Finder: (%2=%1[%2],%3=%1[%3],%4=%1[%4])(E=66,F=100,G=122)http://localhost:8180/A/A/D,
  %5=TableExpRowSet %5(%2,%3,%4) key (%2,%3,%4) Finder: (%2=%5[%2],%3=%5[%3],%4=%5[%4]) Source: %1,
  %6=SelectRowSet %6) Finder: (%2=%5[%2],%3=%5[%3],%4=%5[%4]) Source: %5,
  %8=SelectRowSet %6) Finder: (%2=%5[%2],%3=%5[%3],%4=%5[%4]) Source: %5)}
```

We note the presence of a RestRowSet here, including the names and uids of the remote columns and the URL of the remote view definition.

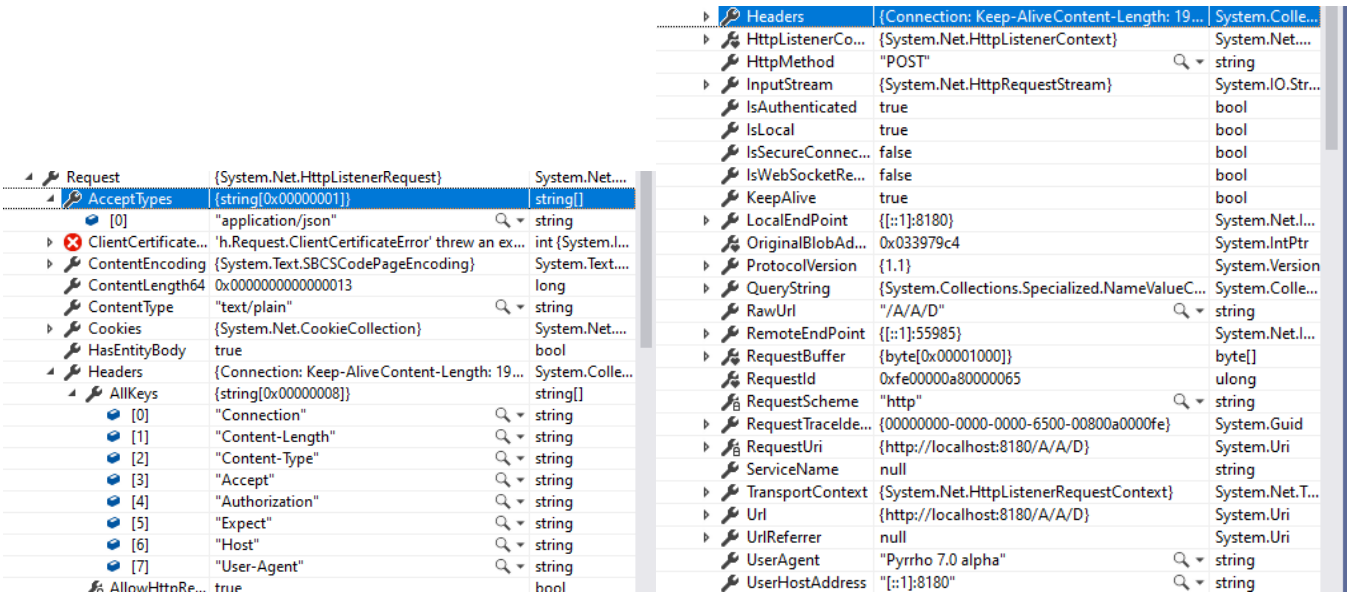
When a rowset is traversed, Pyrrho first checks to see if the RowSet needs to be built. For RestRowSet, the Build method prepares an HTTP POST request to be sent to the given URL, and we can see this because of the -H flag.



By default in v7, this is a query naming the given columns and remote object name as shown. Note that in this feedback, the given URL `http://localhost:8180/A/A/D` has been split to separate the database object name D from the database name and role requested. This is to enable SQL query rewriting (in the requesting server) for the sort of example considered later in this document. The actual HTTP1.1 request that is sent uses the complete URL along with this SQL, and is shown below in the discussion of the HTTPServer behaviour.

It is possible that the server making the request also provides the HTTP service requested, as in this case. But the request is handled in a separate connection and Pyrrho will not know of any relationship between the requestor and the user agent. (For example, the REST service described here might be provided by the Restif server on behalf of another DBMS such as MySql or SQL Server, and such a request may be from other implementations, e.g. RESTClient or in the future another DBMS.)

Here we provide a brief account of how the request is handler by Pyrrho’s built-in HTTPServer. In this case the HTTP request as received contains the following (see HttpServer method)



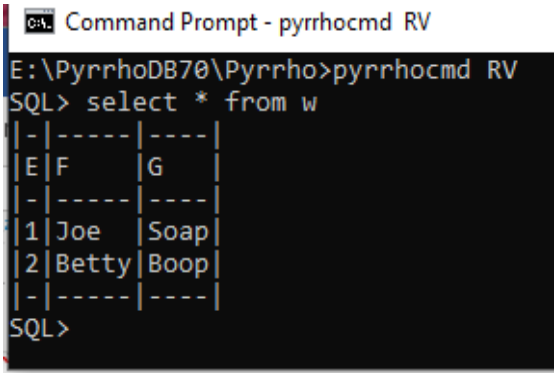
After unpacking the Authorization credentials and database name A, it checks that the URL extension is not .htm, and if not, it creates a connection string and starts a transaction on the requested database. Next, it detects the text/plain content and selects the SqlWebOutput variant of its service. Then the complete url, split by /s, and any ETag supplied, is passed to Transaction.Execute,

Transaction.Execute currently has two modes of operation: H for HTTP1.1 requests and R for a REST service. The latter provides a transaction mechanism for sending a sequence of HTTP requests to be transacted on the server. The current example has not set up an explicit transaction, and so the mode is H. The service can be accessed from a web browser, in which case the url can have a complicated recursive syntax that allows the specification of index keys, where conditions and even procedure calls. Without these extensions, its operation is rather simpler.

The precise SQL syntax expected in the remote query will depend on the remote DBMS being accessed (as set in the SQLAGENT case for metadata). The default is Pyrrho, and here we have the call

```
Execute(cx, "POST", "H",{"","A","A","D"},"text/plain", "select E,F,G from D", null)
```

The POST method finds SELECT and calls ParseCursorSpecification. Then SendResults send the results as a JSON document.



Back in RestRowSet, the Build method receives the string

"[{\"E\": 1, \"F\":' Joe', \"G\":' Soap},{\"E\": 2, \"F\":' Betty', \"G\":' Boop'}]"

and parses it according to the given Domain

```
{Domain TABLE (66,100,122)([66,Domain INTEGER],[100,Domain
CHAR],[122,Domain CHAR]) }
```

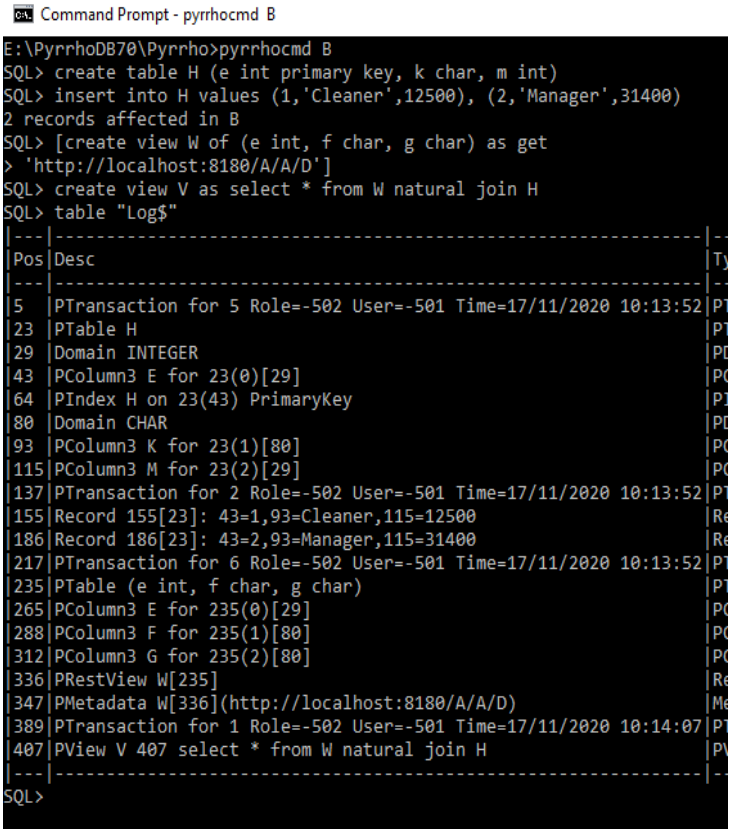
Section 2. Using SQL-style remote syntax, and the example in the paper.

Database B, as mentioned above, was built as follows, and we show the defining positions from the transaction log:

As in section 1, we will look in detail at the parsing of the select statement, which in the paper was

select e,f,m,check from V where e=1

including the instancing of the selected view, in this case V 407. (The view W is at 336 in database B and is instanced using Load.)



select e,f,m,check from W natural join H where e=1

We consider the details for this example first. We will note that the parser replaces all occurrences of E by is lexical position #8 (in general there may be more than one occurrence of an identifier such as E, but the syntax should disambiguate them and parser will keep them separate), of F by #10, etc.

Just before traversal of the result, the context contains the following objects:

```
{(23=Table Name=H 23 Definer=-502 Ppos=23
        Domain TABLE (43,93,115)
                ([43,Domain INTEGER],[93,Domain CHAR],
                [115,Domain INTEGER])
        Enforcement=Select, Insert, Delete, Update
        Indexes:((43)64) KeyCols: (43=True),
  43=TableColumn 43 Definer=-502 Ppos=43 Domain INTEGER
        Table=23 colDefault TypedValue Null,
  93=TableColumn 93 Definer=-502 Ppos=93 Domain CHAR
        Table=23 colDefault TypedValue Null,
```

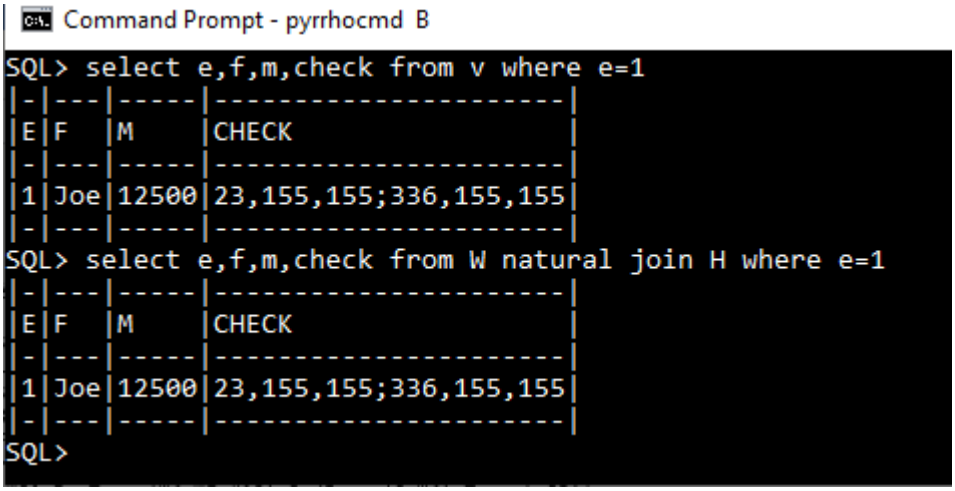
```
  115=TableColumn 115 Definer=-502 Ppos=115 Domain INTEGER Table=23 colDefault TypedValue Null,
  235=Table Name=(e int, f char, g char)  235 Definer=-502 Ppos=235
        Domain TABLE (265,288,312)([265,Domain INTEGER],[288,Domain CHAR],[312,Domain CHAR])
        Enforcement=Select, Insert, Delete, Update KeyCols: ,
  265=TableColumn 265 Definer=-502 Ppos=265 Domain INTEGER Table=235 colDefault TypedValue Null,
  288=TableColumn 288 Definer=-502 Ppos=288 Domain CHAR Table=235 colDefault TypedValue Null,
  312=TableColumn 312 Definer=-502 Ppos=312 Domain CHAR Table=235 colDefault TypedValue Null,
  #0=SelectStatement #0 CS=#1,
  #1=CursorSpecification #1 RowType:(%2,%3,#12,#14) Source={select e,f,m,check from W natural join H where e=1}
        Union: #2,
  #2=QueryExpression #2 RowType:(%2,%3,#12,#14) Left: #7 ,
  #7=QuerySpecification #7 RowType:(%2,%3,#12,#14) TableExp #20,
  #12=SqlCopy Name=M #12 From:#40 Domain INTEGER copy from 115,
  #14=CHECK,
  #20=TableExpression #20 RowType:(%2,%3,%4,#12,%10) Filter:(%9=1) Where:(#49=True) Target: #27,
  #27=JoinPart #27 RowType:(%2,%3,%4,#12,%10) Filter:(%9=1) Matching:(%2=(%9=True)) Where:(#49=True)
        %8 NATURAL INNER join#40 on %11,
  #40=From Name=H #40 RowType:(#12|%9,%10) OrdSpec (0=%9) Target=23,
  #49=SqlValueExpr Name= #49 From:#27 Left:%9 BOOLEAN Right:#50 #49(%9=#50),
  #50=1,
  %0=RestView Name=W %0 Definer=-502 Ppos=336 Query  Ppos: 336 Cols (E=265,F=288,G=312)
        Domain TABLE (%2,%3,%4)([%2,Domain INTEGER],[%3,Domain CHAR],[%4,Domain CHAR])  ViewQry: %8,
  %1=From Name=W %1 RowType:(%2,%3,%4) Target=%0,
  %2=SqlCopy Name=E %2 From:%1 Domain INTEGER copy from 265,
  %3=SqlCopy Name=F %3 From:%1 Domain CHAR copy from 288,
  %4=SqlCopy Name=G %4 From:%1 Domain CHAR copy from 312,
  %5=TableExpression %5 RowType:(%2,%3,%4) Target: %1,
  %6=QuerySpecification %6 RowType:(%2,%3,%4) TableExp %5,
  %7=QueryExpression %7 RowType:(%2,%3,%4) Left: %6 ,
  %8=CursorSpecification %8 RowType:(%2,%3,%4) OrdSpec (0=%2) Source={select * from W} Union: %7,
  %9=SqlCopy Name=E %9 From:#40 Domain INTEGER copy from 43,
  %10=SqlCopy Name=K %10 From:#40 Domain CHAR copy from 93,
  %11=SqlValueExpr Name= %11 Left:%2 BOOLEAN Right:%9 %11(%2=%9))}
```

Using the view V in addition to the RestView W looks more complex, as we can see the elements of precompiled View V at positions in the shared region 408 to 438, as well as the instantiated versions from %0- to %21. But it is important that the the computation of the result will be done the same way. In what follows we will explore filtering and automatic remote query optimisation.

```
{(23=Table Name=H 23 Definer=-502 Ppos=23
        Domain TABLE (43,93,115) ([43,Domain INTEGER],[93,Domain CHAR],[115,Domain INTEGER])
        Enforcement=Select, Insert, Delete, Update Indexes:((43)64) KeyCols: (43=True),
  43=TableColumn 43 Definer=-502 Ppos=43 Domain INTEGER Table=23 colDefault TypedValue Null,
  93=TableColumn 93 Definer=-502 Ppos=93 Domain CHAR Table=23 colDefault TypedValue Null,
  115=TableColumn 115 Definer=-502 Ppos=115 Domain INTEGER Table=23 colDefault TypedValue Null,
  235=Table Name=(e int, f char, g char)  235 Definer=-502 Ppos=235
        Domain TABLE (265,288,312)([265,Domain INTEGER],[288,Domain CHAR],[312,Domain CHAR])
        Enforcement=Select, Insert, Delete, Update KeyCols: ,
  265=TableColumn 265 Definer=-502 Ppos=265 Domain INTEGER Table=235 colDefault TypedValue Null,
  288=TableColumn 288 Definer=-502 Ppos=288 Domain CHAR Table=235 colDefault TypedValue Null,
  312=TableColumn 312 Definer=-502 Ppos=312 Domain CHAR Table=235 colDefault TypedValue Null,
  408=SelectStatement 408 CS=409,
  409=CursorSpecification 409 RowType:(417,419,420,412,413) Source={select * from W natural join H} Union: 410,
  410=QueryExpression 410 RowType:(417,419,420,412,413) Left: 415 ,
  411=SqlCopy Name=E 411 From:438 Domain INTEGER copy from 43,
  412=SqlCopy Name=K 412 From:438 Domain CHAR copy from 93,
  413=SqlCopy Name=M 413 From:438 Domain INTEGER copy from 115,
  414=RestView Name=W 414 Definer=-502 Ppos=336 Query  Ppos: 336 Cols (E=265,F=288,G=312)
        Domain TABLE (417,419,420)([417,Domain INTEGER],[419,Domain CHAR],[420,Domain CHAR])  ViewQry: 426,
  415=QuerySpecification 415 RowType:(417,419,420,412,413) TableExp 418,
  416=SqlStar Name=* 416 CONTENT From:415 CONTENT,
  417=SqlCopy Name=E 417 From:421 Domain INTEGER copy from 265,
  418=TableExpression 418 RowType:(417,419,420,412,413) Target: 425,
  419=SqlCopy Name=F 419 From:421 Domain CHAR copy from 288,
  420=SqlCopy Name=G 420 From:421 Domain CHAR copy from 312,
  421=From Name=W 421 RowType:(417,419,420) Target=414,
  422=TableExpression 422 RowType:(417,419,420) Target: 421,
  423=QuerySpecification 423 RowType:(417,419,420) TableExp 422,
  424=QueryExpression 424 RowType:(417,419,420) Left: 423 ,
  425=JoinPart 425 RowType:(417,419,420,412,413) Matching:(417=(411=True))426 NATURAL INNER join438 on 427,
  426=CursorSpecification 426 RowType:(417,419,420) OrdSpec (0=417) Source={select * from W} Union: 424,
  427=SqlValueExpr Name= 427 Left:417 BOOLEAN Right:411 427(417=411),
  438=From Name=H 438 RowType:(411,412,413) OrdSpec (0=411) Target=23,
  #0=SelectStatement #0 CS=#1,
  #1=CursorSpecification #1 CONTENT  RowType:(#8,#10,#12,#14) Source={select e,f,m,check from v where e=1} Union: #2,
  #2=QueryExpression #2 CONTENT  RowType:(#8,#10,#12,#14) Left: #7 ,
  #7=QuerySpecification #7 CONTENT  RowType:(#8,#10,#12,#14) TableExp #20,
  #8=SqlCopy Name=E #8 From:%14 Domain INTEGER copy from 265,
  #10=SqlCopy Name=F #10 From:%14 Domain CHAR copy from 288,
  #12=SqlCopy Name=M #12 From:%21 Domain INTEGER copy from 115,
  #14=CHECK,
```

```
#20=TableExpression #20 RowType:(#8,#10,%13,%5,#12) Filter:(#8=1) Where:(#34=True) Target: %2,
#34=SqlValueExpr Name= #34 From:%2 Left:#8 BOOLEAN Right:#35 #34(#8=#35),
#35=1,
%0=View Name=V %0 Definer=-502 Ppos=407 Query select * from W natural join H Ppos: 407
      Cols (E=#8,F=#10,G=%13,K=%5,M=#12) Domain ROW (#8,#10,%13,%5,#12) Display=5
      ([#8,Domain INTEGER],[#10,Domain CHAR],[#12,Domain INTEGER],[%5,Domain CHAR],[%13,Domain CHAR])
      ViewQry: %2,
%1=SelectStatement %1 CS=%2,
%2=CursorSpecification %2 RowType:(#8,#10,%13,%5,#12) Filter:(#8=1) Where:(#34=True)
      Source={select * from W natural join H} Union: %3,
%3=QueryExpression %3 RowType:(#8,#10,%13,%5,#12) Left: %8 ,
%4=SqlCopy Name=E %4 From:%21 Domain INTEGER copy from 43,
%5=SqlCopy Name=K %5 From:%21 Domain CHAR copy from 93,
%7=RestView Name=W %7 Definer=-502 Ppos=336 Query  Ppos: 336 Cols (E=265,F=288,G=312)
      Domain TABLE (#8,#10,%13)([#8,Domain INTEGER],[#10,Domain CHAR],[%13,Domain CHAR])  ViewQry: %19,
%8=QuerySpecification %8 RowType:(#8,#10,%13,%5,#12) TableExp %11,
%9=SqlStar Name=* %9 CONTENT From:%8 CONTENT,
%11=TableExpression %11 RowType:(#8,#10,%13,%5,#12) Target: %18,
%13=SqlCopy Name=G %13 From:%14 Domain CHAR copy from 312,
%14=From Name=W %14 RowType:(#8,#10,%13) Target=%7,
%15=TableExpression %15 RowType:(#8,#10,%13) Target: %14,
%16=QuerySpecification %16 RowType:(#8,#10,%13) TableExp %15,
%17=QueryExpression %17 RowType:(#8,#10,%13) Left: %16 ,
%18=JoinPart %18 RowType:(#8,#10,%13,%5,#12) Matching:(#8=(%4=True))%19 NATURAL INNER join%21 on %20,
%19=CursorSpecification %19 RowType:(#8,#10,%13) OrdSpec (0=#8) Source={select * from W} Union: %17,
%20=SqlValueExpr Name= %20 Left:#8 BOOLEAN Right:%4 %20(#8=%4),
%21=From Name=H %21 RowType:(%4,%5,#12) OrdSpec (0=%4) Target=23)}
```

Traversal of either query gives a single row



This row is a join, obtained from two different databases, and it so happens that that the defpos and ppos of the rows used have the same values 155 in the two databases.

Section 3. Using URL-style remote syntax, and the example in the paper

To recover the use of URL-style remote syntax, we need to request this in the metadata for the RESTView W , so that the database B needs to be constructed as follows:

create table H (e int primary key, k char, m int)

insert into H values (1,'Cleaner',12500), (2,'Manager',31400)

create view W of (e int, f char, g char) as get URL 'http://localhost:8180/A/A/D'

create view V as select * from W natural join H

In the paper we now have the following operations on database B:

select e,f,m,check from V where e=1

```
SQL> select e,f,m,check from V where e=1
|-|---|-----|------------------|
|E|F  |M    |CHECK             |
|-|---|-----|------------------|
|1|Joe|12500|A:209:193,B:208:192|
|-|---|-----|------------------|
SQL>
```

Note that in normal use there is no need to request the check pseudocolumn: it is here so can show what is happening within the two databases. The database API uses it to implement the Versioned feature in client side “database model” classes.

Here, the check value was requested explicitly in the SELECT statement, and shows that this row of the join uses a row from A with defining position 209 placed there in transaction 193 (see the log for A) and a row from B with defining position 208 arising from transaction 192 (see the log for B).

The debug information for A shows the REST request from B to A, and the ETag it constructed.

```
311858
HTTP GET /A/A/D/E=1
Returning ETag: A:209:193;A|275|[69--(1)]
```

The ETag consists of an RVV for the first row of the result (mentioned above), and a readCheck for the read operation carried out by A. This was a specific row in table D (position 69) with key (1) .

The next operation is

update v set f='Elizabeth' where e=2

```
SQL> update v set f='Elizabeth' where e=2
1 records affected
SQL>
```

```
TTP GET /A/A/D/E=2
eturning ETag: A:241:193;A|275|[69--(2)]
TTP PUT /A/A/D/E=2/A:241:193
"_id": "A:241:193", "F": "Elizabeth"}
eturning ETag: A:241:275;A|275|[69--(2)]
```

We see there is now an updated ETag supplied by A showing the new transaction that has updated the record defined at 241.

Also check B’s view using the join:

```
SQL> select e,f,m,check from V
|-|---------|-----|------------------|
|E|F        |M    |CHECK             |
|-|---------|-----|------------------|
|1|Joe      |12500|A:209:193,B:208:192|
|2|Elizabeth|31400|A:241:275,B:241:192|
|-|---------|-----|------------------|
SQL>
```

The next operation is an Insert into the View/RestView/Join combination:

[insert into v(e,f,g,k,m)

values(3,'Fred','Smith','Janitor',22160)]

```
SQL> [insert into v(e,f,g,k,m)
> values(3,'Fred','Smith','Janitor',22160)]
2 records affected
SQL>
```

```
TTP GET /A/A/D
eturning ETag: A:209:193;A|335|[69-0]
TTP POST /A/A/D
{"E": 3, "F": "Fred", "G": "Smith"}
eturning ETag: ;A|335|[69-0]
```

And again verifying the view from B:

```
SQL> select e,f,m,check from V
|-|---------|-----|------------------|
|E|F        |M    |CHECK             |
|-|---------|-----|------------------|
|1|Joe      |12500|A:209:193,B:208:192|
|2|Elizabeth|31400|A:241:275,B:241:192|
|3|Fred     |22160|A:351:335,B:501:485|
|-|---------|-----|------------------|
SQL>
```

Finally, we try a deletion from the View/RestView/Join combination:

delete from v where g='Soap'

```
Returning ETag: A:209:193;A|386|[69-0]
HTTP GET /A/A/D/G='Soap'
Returning ETag: A:209:193;A|386|[69-0]
HTTP DELETE /A/A/D/G='Soap'/A:209:193,B:208:192
HTTP GET /A/A/D
Returning ETag: A:241:275;A|409|[69-0]
```

```
|-|---------|-----|------------------|
SQL> delete from v where g='Soap'
1 records affected
SQL> select e,f,m,check from V
|-|---------|-----|------------------|
|E|F        |M    |CHECK             |
|-|---------|-----|------------------|
|2|Elizabeth|31400|A:241:275,B:241:192|
|3|Fred     |22160|A:351:335,B:501:485|
|-|---------|-----|------------------|
```