

All of the details for the RESTView examples

Malcolm Crowe, 1 February 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 accessing the remote database. However, the normal 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 2 February 2021 (some images have different dates), 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"

```
SQL> table "Log$"
```

Pos	Desc	Type	Affects
5	PTransaction for 5 Role=-502 User=-501 Time=19/12/2020 07:08:13	PTransaction	5
23	PTable D	PTable	23
29	Domain INTEGER	PDomain	29
43	PColumn3 E for 23(0)[29]	PColumn3	43
64	PIndex D on 23(43) PrimaryKey	PIndex	64
80	Domain CHAR	PDomain	80
93	PColumn3 F for 23(1)[80]	PColumn3	93
115	PColumn3 G for 23(2)[80]	PColumn3	115
137	PTransaction for 2 Role=-502 User=-501 Time=19/12/2020 07:08:13	PTransaction	137
155	Record 155[23]: 43=1,93=Joe,115=Soap	Record	155
185	Record 185[23]: 43=2,93=Betty,115=Boop	Record	185
217	PTransaction for 1 Role=-502 User=-501 Time=19/12/2020 07:08:13	PTransaction	217
235	PRole A	PRole	235
243	PTransaction for 2 Role=-502 User=!0 Time=19/12/2020 07:09:49	PTransaction	243
267	PUser MALCOLM2\Malco	PUser	267
287	Grant UseRole on -502 to 267	Grant	287

```
SQL>
```

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:

```
CA: Command Prompt - pyrrhocmd RV
E:\PyrrhoDB70\Pyrrho>pyrrhocmd B
SQL> ^C
E:\PyrrhoDB70\Pyrrho>pyrrhocmd RV
SQL> table "Log$"
|---|-----|
|Pos|Desc|
|---|-----|
|5|PTransaction for 6 Role=-502 User=-501 Time=14/12/2020 11:43:59|PTTr
|23|PTable (e int, f char, g char)|PTa
|52|Domain INTEGER|PDO
|66|PColumn3 E for 23(0)[52]|PCo
|87|Domain CHAR|PDO
|100|PColumn3 F for 23(1)[87]|PCo
|122|PColumn3 G for 23(2)[87]|PCo
|144|PRestView W[23]|Res
|153|PMetadata W[144](http://localhost:8180/A/A/D)|Met
|---|-----|
SQL>
```

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, the RestView referred to is fetched from the database, and query objects constructed as follows in the Context:

```
{(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=SqlValue Name=E 66 Domain INTEGER,
  100=SqlValue Name=F 100 Domain CHAR,
  122=SqlValue Name=G 122 Domain CHAR,
  144=RestView Name=W 144 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]) ,
    #1=CursorSpecification #1 RowType:(%0,%1,%2) Source={select * from w} Union: #2,
    #2=QueryExpression #2 RowType:(%0,%1,%2) Left: #7 ,
    #7=QuerySpecification #7 RowType:(%0,%1,%2) TableExp #10,
    #8=SqlStar Name=* #8 CONTENT From:#7 CONTENT,
    #10=TableExpression #10 Nuid=#15 RowType:(%0,%1,%2) Target: #15,
    #15=From Name=W #15 RowType:(%0,%1,%2) Target=144,
    %0=SqlCopy Name=E %0 From:#15 Domain INTEGER copy from 66,
    %1=SqlCopy Name=F %1 From:#15 Domain CHAR copy from 100,
    %2=SqlCopy Name=G %2 From:#15 Domain CHAR copy from 122)}
```

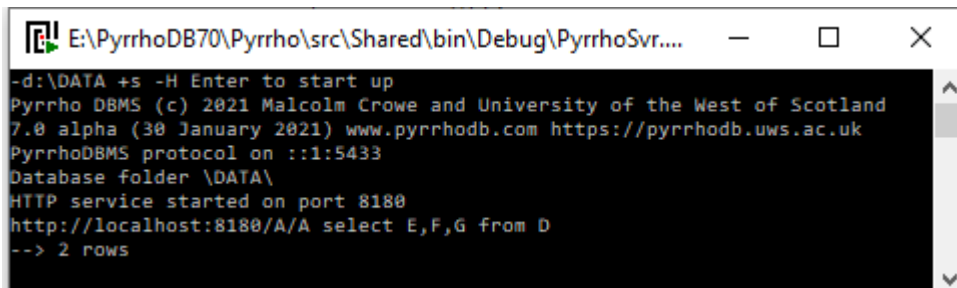
Here the numbers 66,...,144 correspond with log file positions, while #1,...#15 are lexical positions in the parser (they are actually long integers starting at 0x5000000000000000). The numbers %0,...%2 are heap positions for the view columns E,F,G that are implied by the * (they start at 0x7000000000000000).

The next step is the creation of rowsets. The RowSets are built starting from the CursorSpecification #1. Views are compiled objects whose rowsets were prepared when the view definition was loaded into the database, and need to be “instanced” for the current query as we will see for V later. But W is a restview and its rowsets are constructed dynamically. The following rowsets are all we need (in cx.data):

```
{(#1=TableExpRowSet #10(%0,%1,%2) key (%0,%1,%2) target: 144 Finder: (%0=#10[%0],%1=#10[%1],%2=#10[%2]) Source: #15,
#10=TableExpRowSet #10(%0,%1,%2) key (%0,%1,%2) target: 144 Finder: (%0=#10[%0],%1=#10[%1],%2=#10[%2]) Source: #15,
#15=RestRowSet #15(%0,%1,%2) target: 144 Finder: (%0=#15[%0],%1=#15[%1],%2=#15[%2])
144(E=66,F=100,G=122)http://localhost:8180/A/A/D JoinCols: UsingCols:)}
```

We note the RestRowSet here, including the names and uids of the remote columns and the URL of the remote view definition. Its JoinCols and UsingCols fields are empty because this is a GET restview rather than GET USING. Notice that looking up the rowset for the CursorSpecification will takes us directly to the rowset for #10, as the intervening layers of the query analysis do not add anything in this case.

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.



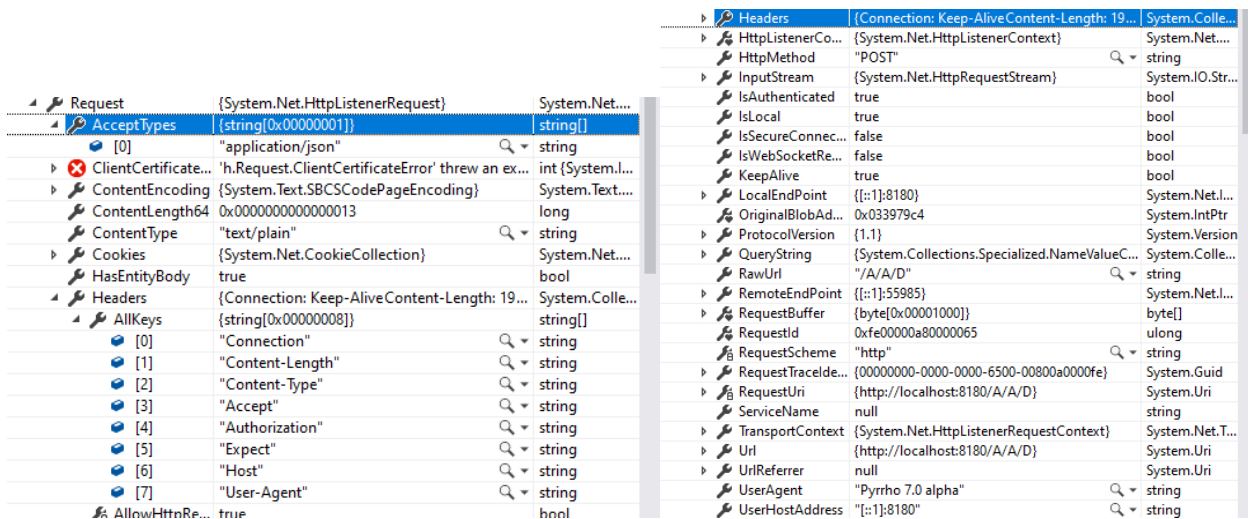
```
E:\PyrrhoDB70\Pyrrho\src\Shared\bin\Debug\PyrrhoSvr....
-d:\DATA +s -H Enter to start up
Pyrrho DBMS (c) 2021 Malcolm Crowe and University of the West of Scotland
7.0 alpha (30 January 2021) www.pyrrhodb.com https://pyrrhodb.uws.ac.uk
PyrrhoDBMS protocol on ::1:5433
Database folder \DATA\
HTTP service started on port 8180
http://localhost:8180/A/A select E,F,G from D
--> 2 rows
```

By default in v7, this is a posted 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 agent¹. (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)

¹ The is a need to tailor the remote SQL to the dialect used by the remote DBMS. This is controlled by metadata flags, discussed elsewhere.



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'}]"
```

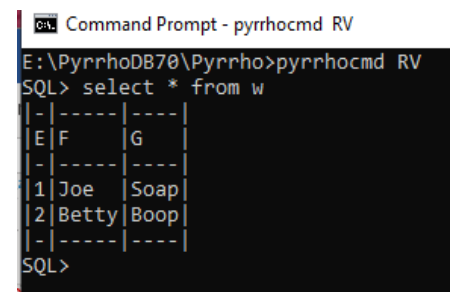
and parses it according to the given Domain

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

Before leaving this example, let us see what happens if a simple filter is added:

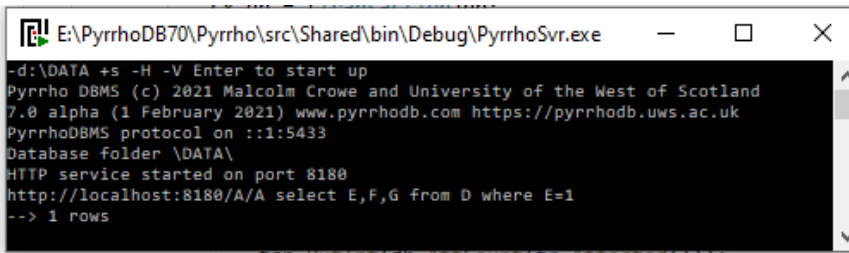
select * from w where e=1

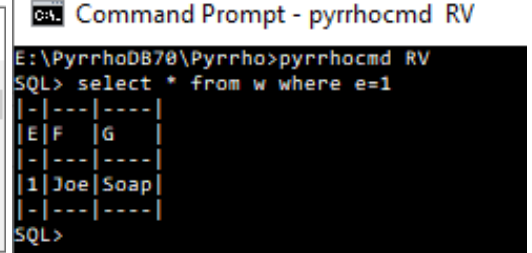
```
{(23=Table Name=(e int, f char, g char) 23 Definer=-502 Ppos=23
  Domain TABLE (66,100,122)([66,Domain INTEGER],[100,Domain CHAR],[122,Domain CHAR])
  Enforcement=Select, Insert, Delete, Update KeyCols: ,
  66=SqlValue Name=E 66 Domain INTEGER,
  100=SqlValue Name=F 100 Domain CHAR,
  122=SqlValue Name=G 122 Domain CHAR,
  144=RestView Name=W 144 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]) ,
  #0=SelectStatement #0 CS=#1,
  #1=CursorSpecification #1 RowType:(%0,%1,%2) Source={select * from w where e=1} Union: #2,
  #2=QueryExpression #2 RowType:(%0,%1,%2) Left: #7 ,
  #7=QuerySpecification #7 RowType:(%0,%1,%2) TableExp #10,
  #8=SqlStar Name=* #8 CONTENT From:#7 CONTENT,
  #10=TableExpression #10 Nuid=#15 RowType:(%0,%1,%2) Filter:(%0=1) Where:(#24=True) Target: #15,
  #15=From Name=W #15 RowType:(%0,%1,%2) Filter:(%0=1) Where:(#24=True) Target=144,
  #24=SqlValueExpr Name= #24 From:#15 Left:%0 BOOLEAN Right:#25 #24(%0=#25),#25=1,
  %0=SqlCopy Name=E %0 From:#15 Domain INTEGER copy from 66,
  %1=SqlCopy Name=F %1 From:#15 Domain CHAR copy from 100,
  %2=SqlCopy Name=G %2 From:#15 Domain CHAR copy from 122)}
{(#1=TableExpRowSet #10(%0,%1,%2) key (%0,%1,%2) where (#24) matches (%0=1) target: 144 Source: #15,
  #10=TableExpRowSet #10(%0,%1,%2) key (%0,%1,%2) where (#24) matches (%0=1) target: 144 Source: #15,
```



```
#15=RestRowSet #15(%0,%1,%2) where (#24) target: 144 144(E=66,F=100,G=122)http://localhost:8180/A/A/D JoinCols:
UsingCols:)} }
```

We see the filter is applied at successively lower levels of the query, and at the RestRowSet level, the request made to the remote database takes the filter into account, as shown below (notice “where E=1”):





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

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

We see the RestView W defined as before (but now at position 336), and View V defined at file position 407. When the database is loaded, the stored join defining V is parsed into stored rowsets as we will see below.

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

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

This should end up effectively the same as for

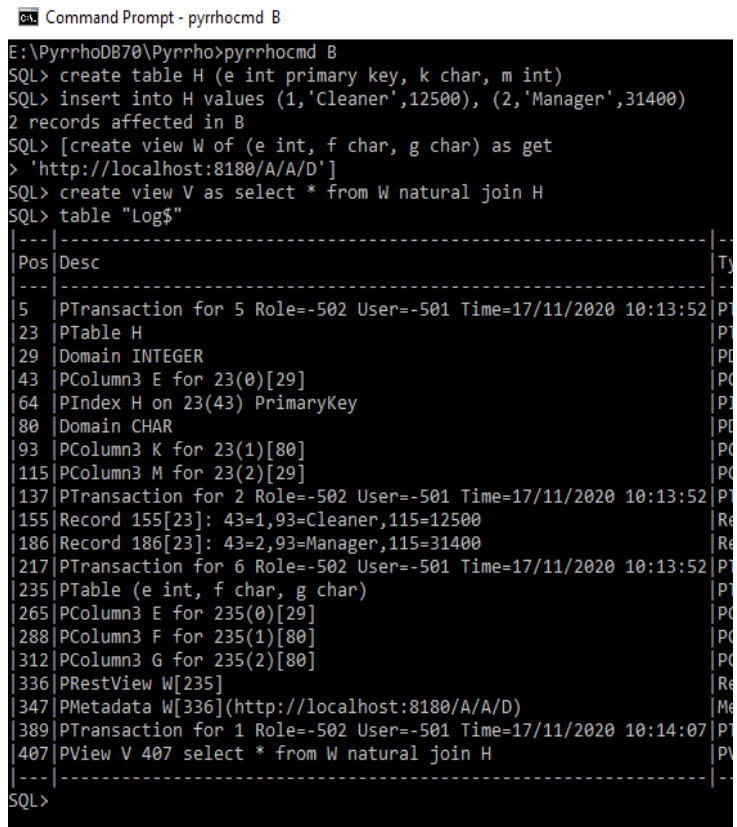
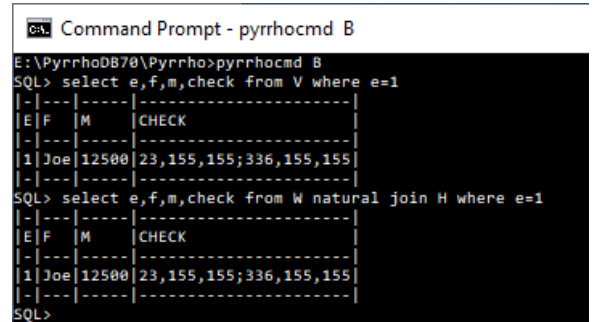
select e,f,m,check from W natural join H where e=1

(In the results screenshot, it is a coincidence that in the CHECK column the same position 155 arises in both databases A and B.)

We consider the details for the second example first: it uses the RestView but not the stored join defining V. The parser replaces all occurrences of identifiers such as E by its defining 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 the parser will keep them separate), of F by #10, etc.

After rowset construction, and 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=SqlValue Name=E 265 Domain INTEGER,288=SqlValue Name=F 288 Domain CHAR,
  312=SqlValue Name=G 312 Domain CHAR,
  336=RestView Name=W 336 Definer=-502 Ppos=336 Query Ppos: 336 Cols (E=265,F=288,G=312)
  Domain TABLE (265,288,312)((265,Domain INTEGER],[288,Domain CHAR],[312,Domain CHAR)) ,
  #0=SelectStatement #0 CS=#1,
  #1=CursorSpecification #1 RowType:(#8,#10,#12,#14) Source={select e,f,m,check from W natural join H where e=1}
  Union: #2,
```


```

#2=QueryExpression #2 RowType:(#8,#10,#12,#14) Left: #7 ,
#7=QuerySpecification #7 RowType:(#8,#10,#12,#14) TableExp #20,
#8=SqlCopy Name=E #8 From:#25 Domain INTEGER copy from 265,
#10=SqlCopy Name=F #10 From:#25 Domain CHAR copy from 288,
#12=SqlCopy Name=M #12 From:#40 Domain INTEGER copy from 115,
#14=CHECK,
#20=TableExpression #20 Nuid=#27 RowType:(#8,#10|#12,%0,%2,%1) Filter:(%1=1) Where:(#49=True) Target: #27,
#25=From Name=W #25 RowType:(#8,#10|#12,%0) OrdSpec (0=#8) Target=336,
#27=JoinPart #27 RowType:(#8,#10|#12,%0,%2,%1) Filter:(%1=1) Where:(#49=True)#25 NATURAL INNER join#40 on %3
    matching #8=%1 %1=#8,
#40=From Name=H #40 RowType:(#12|#1,%2) OrdSpec (0=#1) Target=23,
#49=SqlValueExpr Name= #49 From:#27 Left:%1 BOOLEAN Right:#50 #49(%1=#50),
#50=1,
%0=SqlCopy Name=G %0 From:#25 Domain CHAR copy from 312,
%1=SqlCopy Name=E %1 From:#40 Domain INTEGER copy from 43,
%2=SqlCopy Name=K %2 From:#40 Domain CHAR copy from 93,
%3=SqlValueExpr Name= %3 Left:#8 BOOLEAN Right:%1 %3(#8=%1)})
{(64=IndexRowSet 64(43,93,115) target: 23,
#1=SelectRowSet #7(#8,#10,#12,#14) Source: #20,
#7=SelectRowSet #7(#8,#10,#12,#14) Source: #20,
#20=TableExpRowSet #20(#8,#10|#12,%0,%2,%1) key (#8,#10) where (#49) matches (%1=1) Source: #27,
#25=RestRowSet #25(#8,#10|#12,%0) order (#8) matches (#8=1) target: 336
    336(E=265,F=288,G=312)http://localhost:8180/A/A/D JoinCols: UsingCols:,
#27=JoinRowSet #27(#8,#10|#12,%0,%2,%1) where (#49) matches (#8=1,%1=1) JoinCond: (%3) matching #8=%1 %1=#8
    First: %4 Second: %5,
#40=SelectedRowSet #40(#12|#1,%2) matches (#8=1,%1=1) target: 23 Source: 64,
%4=RestRowSet %4(#8,#10|#12,%0) order (#8) matches (#8=1) target: 336
    336(E=265,F=288,G=312)http://localhost:8180/A/A/D JoinCols: UsingCols:,
%5=SelectedRowSet %5(#12|#1,%2) matches (#8=1,%1=1) target: 23 Source: 64)}

```

It is noteworthy that %4 and %5 were originally OrderedRowSets, to arrange the rows in the best order for the natural join, and Pyrrho has noticed that no ordering is required. The greyed-out entries are duplicates or bypassed entries where stages of computation have been removed during the review process.

We now contrast this with the query that references the RestView via the stored view V. Below 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 same optimisations are carried out and the computation of the result is done the same way.

```

{(23=Table Name=H 23 Definer=-502 Ppos=23
    Domain TABLE (43,93,115)([43,Domain INTEGER],[93,Domain CHAR],[115,Domain INTEGER])
    Indexes:((43)64) KeyCols: (43=True),
43=TableColumn 43 Definer=-502 Ppos=43 Domain INTEGER Table=23,
93=TableColumn 93 Definer=-502 Ppos=93 Domain CHAR Table=23,
115=TableColumn 115 Definer=-502 Ppos=115 Domain INTEGER Table=23,
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]) KeyCols: ,
265=SqlValue Name=E 265 Domain INTEGER,
288=SqlValue Name=F 288 Domain CHAR,
312=SqlValue Name=G 312 Domain CHAR,
336=RestView Name=W 336 Definer=-502 Ppos=336 Query Ppos: 336 Cols (E=265,F=288,G=312)
    Domain TABLE (265,288,312)([265,Domain INTEGER],[288,Domain CHAR],[312,Domain CHAR]) ,
407=View Name=V 407 Definer=-502 Ppos=407 Query select * from W natural join H Ppos: 407
    Cols (E=419,F=412,G=413,K=414,M=417) Domain ROW (419,412,413,414,417|411) Display=5
    ([411,Domain INTEGER],[412,Domain CHAR],[413,Domain CHAR],[414,Domain CHAR],[417,Domain INTEGER],
    [419,Domain INTEGER]) ,
408=SelectStatement 408 CS=409,
409=CursorSpecification 409 RowType:(419,412,413,414,417,411) Source={select * from W natural join H} Union: 410,
410=QueryExpression 410 RowType:(419,412,413,414,417,411) Left: 415 ,
411=SqlCopy Name=E 411 From:438 Domain INTEGER copy from 43,
412=SqlCopy Name=F 412 From:423 Domain CHAR copy from 288,
413=SqlCopy Name=G 413 From:423 Domain CHAR copy from 312,
414=SqlCopy Name=K 414 From:438 Domain CHAR copy from 93,
415=QuerySpecification 415 RowType:(419,412,413,414,417,411) TableExp 418,
416=SqlStar Name=* 416 CONTENT From:415 CONTENT,
417=SqlCopy Name=M 417 From:438 Domain INTEGER copy from 115,
418=TableExpression 418 Nuid=425 RowType:(419,412,413,414,417|411) Target: 425,
419=SqlCopy Name=E 419 From:423 Domain INTEGER copy from 265,
420=SqlValueExpr Name= 420 Left:419 BOOLEAN Right:411 420(419=411),
423=From Name=W 423 RowType:(419,412,413) OrdSpec (0=419) Target=336,
425=JoinPart 425 RowType:(419,412,413,414,417|411)423 NATURAL INNER join438 on 420 matching 411=419 419=411,
438=From Name=H 438 RowType:(411,414,417) OrdSpec (0=411) Target=23,
#0=SelectStatement #0 CS=#1,
#1=CursorSpecification #1 RowType:(#8,#10,#12,#14) Source={select e,f,m,check from v where e=1} Union: #2,
#2=QueryExpression #2 RowType:(#8,#10,#12,#14) Left: #7 ,
#7=QuerySpecification #7 RowType:(#8,#10,#12,#14) TableExp #20,

```

```

#8=SqlCopy Name=E #8 From:#25 Domain INTEGER copy from 419,
#10=SqlCopy Name=F #10 From:#25 Domain CHAR copy from 412,
#12=SqlCopy Name=M #12 From:#25 Domain INTEGER copy from 417,
#14=CHECK,
#20=TableExpression #20 Nuid=#25 RowType:(#8,#10,#12|%0,%1) Filter:(#8=1) Where:(#34=True) Target: #25,
#25=From Name=V #25 RowType:(#8,#10,#12|%0,%1) Filter:(#8=1) Where:(#34=True) Target=407,
#34=SqlValueExpr Name= #34 From:#25 Left:#8 BOOLEAN Right:#35 #34(#8=#35),
#35=1,
%0=SqlCopy Name=G %0 From:#25 Domain CHAR copy from 413,
%1=SqlCopy Name=K %1 From:#25 Domain CHAR copy from 414)}}
{64=IndexRowSet 64(43,93,115) target: 23,
407=TableExpRowSet 418(419,412,413,414,417|411) key (419,412,413,414,417) Source: 425,
409=TableExpRowSet 418(419,412,413,414,417|411) key (419,412,413,414,417) Source: 425,
418=TableExpRowSet 418(419,412,413,414,417|411) key (419,412,413,414,417) Source: 425,
421=OrderedRowSet 421(419,412,413) key (419) order (419) target: 336 Source: 423,
422=OrderedRowSet 422(411,414,417) key (411) order (411) target: 23 Source: 438,
423=RestRowSet 423(419,412,413) key (419,412,413) order (419) target: 336
336(E=265,F=288,G=312)http://localhost:8180/A/A/D JoinCols: UsingCols:,
425=JoinRowSet 425(419,412,413,414,417|411) key (419,412,413,414,417,411) JoinCond: (420) matching 411=419 419=411
First: 421 Second: 422,
438=SelectedRowSet 438(411,414,417) key (411,414,417) target: 23 Source: 64,
#1=SelectRowSet #7(#8,#10,#12,#14) Source: #20,
#7=SelectRowSet #7(#8,#10,#12,#14) Source: #20,
#20=TableExpRowSet #20(#8,#10,#12|%0,%1) key (#8,#10,#12) where (#34) matches (#8=1) Source: #25,
#25=TableExpRowSet #25(#8,#10,%0,%1,#12|411) key (#8,#10,%0,%1,#12) where (#34) matches (#8=1) Source: %2,
%2=JoinRowSet %2(#8,#10,%0,%1,#12|411) key (#8,#10,%0,%1,#12,411) where (#34) matches (#8=1) JoinCond: (%6)
matching 411=419 419=411 First: %3 Second: 422,
%3=RestRowSet %3(#8,#10,%0) key (#8,#10,%0) order (#8) where (#34) matches (#8=1) target: 336
336(E=265,F=288,G=312)http://localhost:8180/A/A/D JoinCols: UsingCols:,
%4=RestRowSet %4(#8,#10,%0) key (#8,#10,%0) order (#8) where (#34) matches (#8=1) target: 336
336(E=265,F=288,G=312)http://localhost:8180/A/A/D JoinCols: UsingCols:)}

```

The shareable rowsets at positions 421 to 438 have been instantiated for the non-shared View #25, resulting in rowsets %2 and %3 following optimisation as before. The instantiation process enables the same view to be referenced in several places in a query, by generating unique uids for each instance.

Traversal of either query gives a single row as shown in the screenshot above. In both cases only one row is fetched from database A. (For this particular row there is a coincidence between the separate databases A and B at position 155, and this makes the value in the check column look strange.)

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)
```

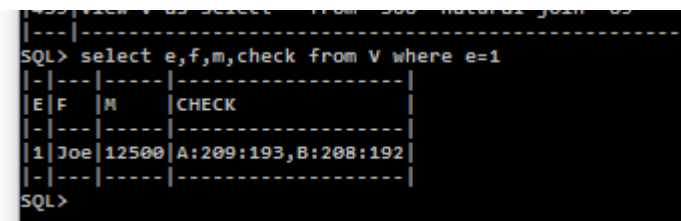
```
insert into H values (1,'Cleaner',12500), (2,'Manager',31400)
```

```
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>
```

```
HTTP GET /A/A/D/E=2
Returning ETag: A:241:193;A|275|[69--(2)]
HTTP PUT /A/A/D/E=2/A:241:193
{"_id": "A:241:193", "F": "Elizabeth"}
Returning 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)
C> values(3,'Fred','Smith','Janitor',22160)]
2 records affected
SQL>
```

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

```
SQL>
```