

All of the details for the RESTView examples

Malcolm Crowe, 10 April 2021



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 9 April 2021, and localhost instead of servA. I have set a debugging -H 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.

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"

```
Command Prompt - pyrrhocmd A
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 RV:

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

Command Prompt - pyrrhocmd RV

```
E:\PyrrhoDB70\Pyrrho>pyrrhocmd B
SQL> ^C
E:\PyrrhoDB70\Pyrrho>pyrrhocmd RV
SQL> table "Log$"
|---|-----|-----|
|Pos|Desc|-----|Typ|
|---|-----|-----|
|5|PTransaction for 6 Role=-502 User=-501 Time=14/12/2020 11:43:59|PTR|
|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. The Framing for the RestView is just the anonymous table definition:

```
{Framing
(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]) KeyCols: ,
66 SqlValue Name=E 66 Domain INTEGER,
100 SqlValue Name=F 100 Domain CHAR,
122 SqlValue Name=G 122 Domain CHAR)
Results: (})Result 0 Results: (})
```

Note SqlValues for the virtual columns here (not SqlCopy). Consider what happens when we request

select * from w

on this database.

When this select statement is parsed, the RestView objects above are added to the Context, and query objects constructed as follows (To see them, set a breakpoint in Start.cs at line 416):

```
{(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]) 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[144],F:100[144],G:122[144])
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).

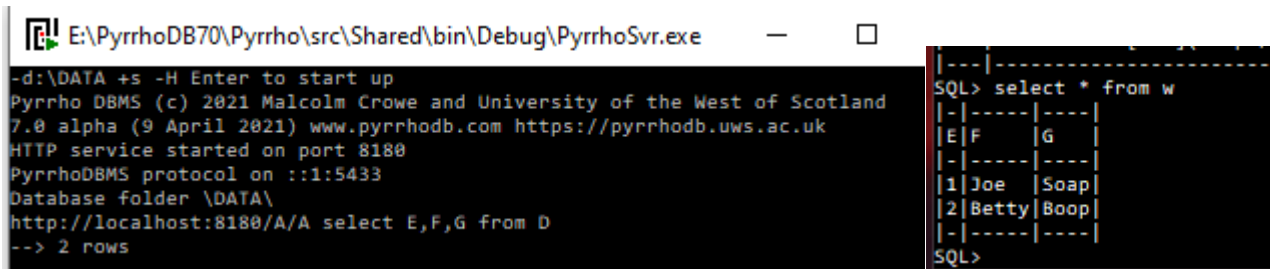
The next step is the creation of rowsets. The RowSets are built starting from the CursorSpecification #1. Views generally are compiled objects whose rowsets were prepared when the view definition was loaded into the

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):

```
{(#10=TableExpRowSet #10(%0,%1,%2) key (%0,%1,%2) targets: 23=#15 Source: #15,  
  #15=RestRowSet #15(%0,%1,%2) targets: 23=#15 http://localhost:8180/A/A/D  
  RemoteCols: (E:66[144],F:100[144],G:122[144]))}
```

We note the 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.



```
E:\PyrrhoDB70\Pyrrho\src\Shared\bin\Debug\PyrrhoSvr.exe  
-d:\DATA +s -H Enter to start up  
Pyrrho DBMS (c) 2021 Malcolm Crowe and University of the West of Scotland  
7.0 alpha (9 April 2021) www.pyrrhodb.com https://pyrrhodb.uws.ac.uk  
HTTP service started on port 8180  
PyrrhoDBMS protocol on ::1:5433  
Database folder \DATA\  
http://localhost:8180/A/A select E,F,G from D  
--> 2 rows  
SQL> select * from w  
[E|F|G|  
[1|Joe|Soap|  
[2|Betty|Boop|  
SQL>
```

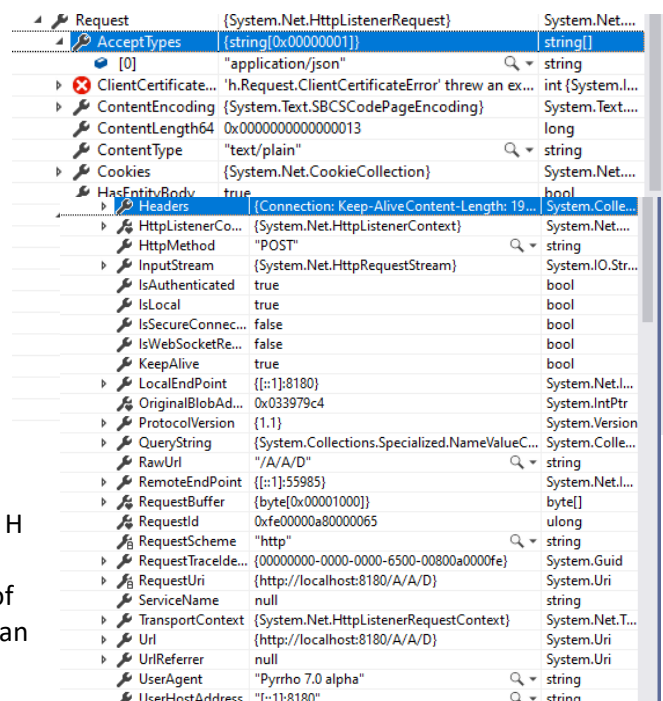
By default in v7, this is a posted query naming the given columns and remote object name as shown. Note that in this feedback, the given URL `http://localhost:8180/A/A/D` has been split to separate the database object name D from the database name and role requested. This is to enable SQL query rewriting (in the requesting server) for the sort of example considered later in this document, and support explicit transactions. 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 handled 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 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.



Request	(System.Net.HttpListenerRequest)	System.Net...
AcceptTypes	{string(0x00000001)}	string[]
[0]	"application/json"	string
ClientCertificate...	'h.Request.ClientCertificateError' threw an ex...	int (System.I...
ContentEncoding	{System.Text.SBCodePageEncoding}	System.Text....
ContentLength64	0x0000000000000013	long
ContentType	"text/plain"	string
Cookies	{System.Net.CookieCollection}	System.Net....
HasEntityBody	true	bool
Headers	{(Connection: Keep-Alive Content-Length: 19...	System.Colle...
HttpListenerCo...	{System.Net.HttpListenerContext}	System.Net....
HttpMethod	"POST"	string
InputStream	{System.Net.HttpRequestStream}	System.IO.Str...
IsAuthenticated	true	bool
IsLocal	true	bool
IsSecureConnec...	false	bool
IsWebSocketRe...	false	bool
KeepAlive	true	bool
LocalEndPoint	{::1}:8180	System.Net.I...
OriginalBlobAd...	0x033979c4	System.IntPtr
ProtocolVersion	{1.1}	System.Version
QueryString	{System.Collections.Specialized.NameValueC...	System.Colle...
RawUrl	"/A/A/D"	string
RemoteEndPoint	{::1}:55985	System.Net.I...
RequestBuffer	{byte(0x00001000)}	byte[]
RequestId	0xfe0000a80000065	ulong
RequestScheme	"http"	string
RequestTracelde...	{00000000-0000-0000-6500-00800a0000fe}	System.Guid
RequestUri	{http://localhost:8180/A/A/D}	System.Uri
ServiceName	null	string
TransportContext	{System.Net.HttpListenerRequestContext}	System.Net.T...
Url	{http://localhost:8180/A/A/D}	System.Uri
UrlReferrer	null	System.Uri
UserAgent	"Pyrrho 7.0 alpha"	string
UserHostAddress	"::1":8180"	string

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.

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 (for Database A)

```
Execute(cx, "POST", "H",{ "", "A", "A"}, "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 from the ResponseStream:

```
"[{\"E\": 1, \"F\": 'Joe', \"G\": 'Soap', \"$pos\": 155, \"$check\": 155},  
  {\"E\": 2, \"F\": 'Betty', \"G\": 'Boop', \"$pos\": 185, \"$check\": 185}]"
```

(we note the RVV/Etag information) and parses it according to the given Domain (cf #15 From W above):

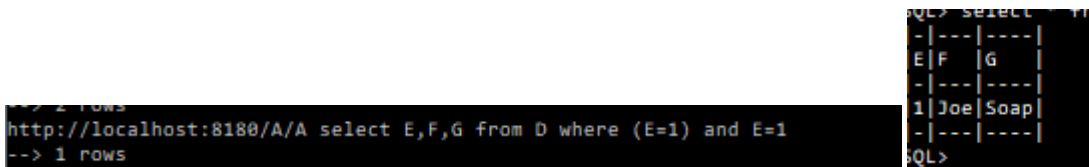
```
{Domain TABLE (%0,%1,%2) Display=3([%0,Domain INTEGER],[%1,Domain CHAR],[%2,Domain CHAR]) }
```

Before leaving this example, let us try a few more things. For example, 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]) 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[144],F:100[144],G:122[144])  
    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 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)}  
{(#10=TableExpRowSet #10(%0,%1,%2) key (%0,%1,%2) where (#24) matches (%0=1) targets: 23=#15 Source: #15,  
  #15=RestRowSet #15(%0,%1,%2) where (#24) matches (%0=1) targets: 23=#15 http://localhost:8180/A/A/D  
  RemoteCols:(E:66[144],F:100[144],G:122[144]))}
```

We see the filter is applied to rowsets 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”, and the redundant repetition is because for the moment it is both a match and a where)



```
SQL> select E,F,G from D where (E=1) and E=1  
--> 2 rows  
http://localhost:8180/A/A select E,F,G from D where (E=1) and E=1  
--> 1 rows
```

For remote views such as this, it is useful (if the remote database has granted permission) to be able to perform updates, inserts and maybe even deletes. This will show the power of the RowSet-first design, and how the REST service defers changes to the Commit point. Let us try

insert into w values(3,'Fred','Bloggs')

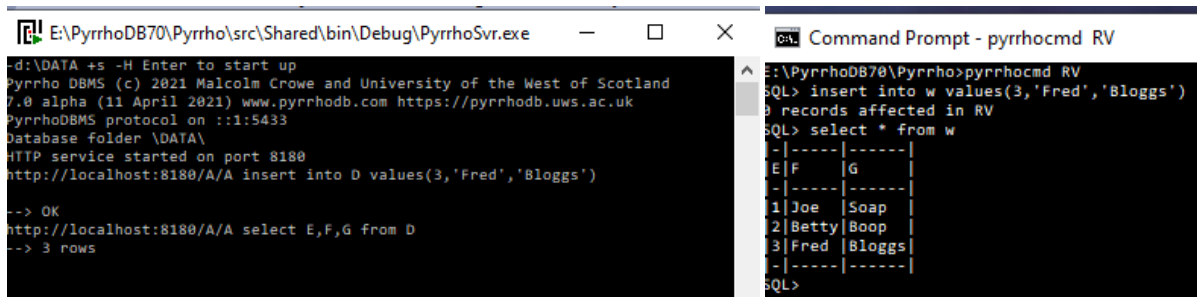
Setting a break point in ParseSqlInsert at line 6606 of Parser.cs, we see cx.obs contains (showing entries only in the lexical and heap ranges # and %):

```
{(.. #1=SqlInsert #1 Nuid=#13 Target: 144 Value: #13,  
  #13=From Name=W #13 RowType:(%0,%1,%2) Target=144,  
  #15=#21,  
  #21=SqlRow #21 Domain ROW (#22,#24,#31)([#22,INTEGER],[#24,CHAR],[#31,CHAR]) [#22,#24,#31],  
  #22=3,  
  #24=Fred,  
  #31=Bloggs,  
  %0=SqlCopy Name=E %0 From:#13 Domain INTEGER copy from 66,  
  %1=SqlCopy Name=F %1 From:#13 Domain CHAR copy from 100,  
  %2=SqlCopy Name=G %2 From:#13 Domain CHAR copy from 122)}
```

and cx.data has:

```
{(#13=RestRowSet #13(%0,%1,%2) targets: 23=#13 Source: #15 http://localhost:8180/A/A/D
  RemoteCols: (E:66[144],F:100[144],G:122[144]),
  #15=SqlRowSet #15(%0,%1,%2))}
```

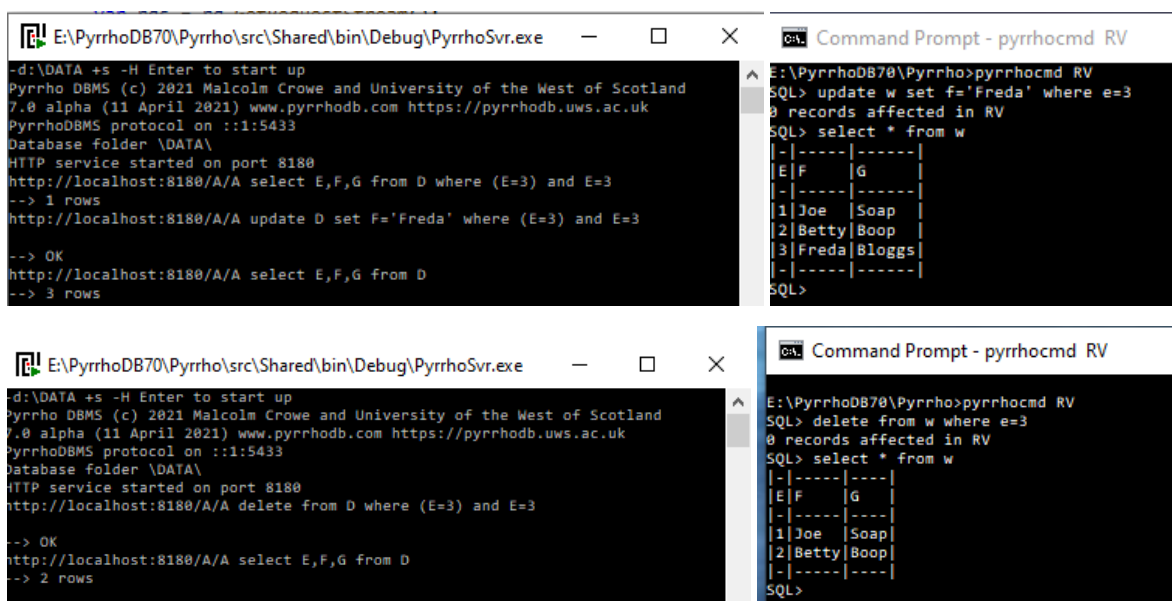
Now be careful not to delay execution (Http operations time out), but place breakpoints in Post.Commit() (line 551 of Physical.cs) to see the RoundTrip() method is called:



```
E:\PyrrhoDB70\Pyrrho\src\Shared\bin\Debug\PyrrhoSvr.exe
-d:\DATA +s -H Enter to start up
Pyrrho DBMS (c) 2021 Malcolm Crowe and University of the West of Scotland
7.0 alpha (11 April 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 insert into D values(3,'Fred','Bloggs')
--> OK
http://localhost:8180/A/A select E,F,G from D
--> 3 rows

E:\PyrrhoDB70\Pyrrho>pyrrhocmd RV
SQL> insert into w values(3,'Fred','Bloggs')
0 records affected in RV
SQL> select * from w
|-|-----|-----|
|E|F|G|
|-|-----|-----|
|1|Joe|Soap|
|2|Betty|Boop|
|3|Fred|Bloggs|
|-|-----|-----|
SQL>
```

Update and Delete are handled similarly:



```
E:\PyrrhoDB70\Pyrrho\src\Shared\bin\Debug\PyrrhoSvr.exe
-d:\DATA +s -H Enter to start up
Pyrrho DBMS (c) 2021 Malcolm Crowe and University of the West of Scotland
7.0 alpha (11 April 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 where (E=3) and E=3
--> 1 rows
http://localhost:8180/A/A update D set F='Freda' where (E=3) and E=3
--> OK
http://localhost:8180/A/A select E,F,G from D
--> 3 rows

E:\PyrrhoDB70\Pyrrho>pyrrhocmd RV
SQL> update w set f='Freda' where e=3
0 records affected in RV
SQL> select * from w
|-|-----|-----|
|E|F|G|
|-|-----|-----|
|1|Joe|Soap|
|2|Betty|Boop|
|3|Freda|Bloggs|
|-|-----|-----|
SQL>
```

```
E:\PyrrhoDB70\Pyrrho\src\Shared\bin\Debug\PyrrhoSvr.exe
-d:\DATA +s -H Enter to start up
Pyrrho DBMS (c) 2021 Malcolm Crowe and University of the West of Scotland
7.0 alpha (11 April 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 delete from D where (E=3) and E=3
--> OK
http://localhost:8180/A/A select E,F,G from D
--> 2 rows

E:\PyrrhoDB70\Pyrrho>pyrrhocmd RV
SQL> delete from w where e=3
0 records affected in RV
SQL> select * from w
|-|-----|-----|
|E|F|G|
|-|-----|-----|
|1|Joe|Soap|
|2|Betty|Boop|
|-|-----|-----|
SQL>
```

Section 2. Remote views and Joins

In the paper, database B has the following:

Database B:

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

As explained in the paper, normal use does not require the CHECK column: it is here because the paper was discussion ETags and RVV.

The defining positions from the transaction log are:


```

C:\ Command Prompt - pyrrhocmd B
E:\PyrrhoDB70\Pyrrho>pyrrhocmd B
SQL> create table H (e int primary key, k char, m int)
SQL> insert into H values (1,'Cleaner',12500), (2,'Manager',31400)
2 records affected in B
SQL> [create view W of (e int, f char, g char) as get
> 'http://localhost:8180/A/A/D']
SQL> create view V as select * from W natural join H
SQL> table "Log$"
|---|-----|-----|-----|-----|-----|
|Pos|Desc|-----|-----|-----|-----|
|---|-----|-----|-----|-----|-----|
|5|PTransaction for 5 Role=-502 User=-501 Time=17/11/2020 10:13:52|PT|
|23|PTable H|-----|-----|-----|-----|
|29|Domain INTEGER|-----|-----|-----|-----|
|43|PColumn3 E for 23(0)[29]|-----|-----|-----|-----|
|64|PIndex H on 23(43) PrimaryKey|-----|-----|-----|-----|
|80|Domain CHAR|-----|-----|-----|-----|
|93|PColumn3 K for 23(1)[80]|-----|-----|-----|-----|
|115|PColumn3 M for 23(2)[29]|-----|-----|-----|-----|
|137|PTransaction for 2 Role=-502 User=-501 Time=17/11/2020 10:13:52|PT|
|155|Record 155[23]: 43=1,93=Cleaner,115=12500|Re|
|186|Record 186[23]: 43=2,93=Manager,115=31400|Re|
|217|PTransaction for 6 Role=-502 User=-501 Time=17/11/2020 10:13:52|PT|
|235|PTable (e int, f char, g char)|-----|-----|-----|-----|
|265|PColumn3 E for 235(0)[29]|-----|-----|-----|-----|
|288|PColumn3 F for 235(1)[80]|-----|-----|-----|-----|
|312|PColumn3 G for 235(2)[80]|-----|-----|-----|-----|
|336|PRestView W[235]|-----|-----|-----|-----|
|347|PMetadata W[336](http://localhost:8180/A/A/D)|Me|
|389|PTransaction for 1 Role=-502 User=-501 Time=17/11/2020 10:14:07|PT|
|407|PView V 407 select * from W natural join H|PV|
|---|-----|-----|-----|-----|-----|
SQL>

```

We see the RestView W defined as before (but now at position 336), and View V is 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.

The select statement in the paper was

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

We will show this is implemented effectively as

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.

At the end of parsing the CursorSpecification, but before RowSet construction (line 5377 of Parser.cs) the context contains (cx.obs)

```

{(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[336],F:288[336],G:312[336])
}

```

```

C:\ Command Prompt - pyrrhocmd B
E:\PyrrhoDB70\Pyrrho>pyrrhocmd B
SQL> select e,f,m,check from V where e=1
|---|-----|-----|-----|
|E|F|M|CHECK|
|---|-----|-----|-----|
|1|Joe|12500|23,155,155;336,155,155|
|---|-----|-----|-----|
SQL> select e,f,m,check from W natural join H where e=1
|---|-----|-----|-----|
|E|F|M|CHECK|
|---|-----|-----|-----|
|1|Joe|12500|23,155,155;336,155,155|
|---|-----|-----|-----|
SQL>

```

```

E:\PyrrhoDB70\Pyrrho\src\Shared\bin\Debug\PyrrhoSvr.exe
-d:\DATA +s -H Enter to start up
Pyrrho DBMS (c) 2021 Malcolm Crowe and University of the West of Scotland
7.0 alpha (9 April 2021) www.pyrrhodb.com https://pyrrhodb.uws.ac.uk
PyrrhoDBMS protocol on :15433
Database folder \DATA\
HTTP service started on port 8180
http://localhost:8180/A/A select E,F,G from D where E=1
--> 1 rows

```

```

Domain TABLE (265,288,312)([265,Domain INTEGER],[288,Domain CHAR],[312,Domain CHAR]) ,
#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,%2,%1}) 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,%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))}

```

Step Over line 5377, and look at the RowSets that have been constructed:

```

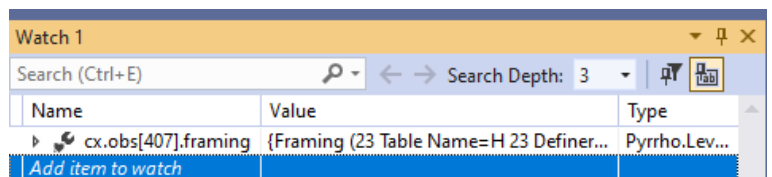
{(64=IndexRowSet 64(43,93,115) Keys: (43,93,115),
#7=SelectRowSet #7(#8,#10,#12,#14) targets: 23=%5,235=%4 Source: #20,
#20=TableExpRowSet #20(#8,#10|#12,%0,%2,%1) key (#8,#10) where (#49) matches (%1=1) targets: 23=%5,235=%4
Source: #27,
#25=RestRowSet #25(#8,#10|%0) matches (#8=1) targets: 235=#25 http://localhost:8180/A/A/D
RemoteCols: (E:265[336],F:288[336],G:312[336]),
#27=JoinRowSet #27(#8,#10|#12,%0,%2,%1) where (#49) matches (#8=1,%1=1) targets: 23=%5,235=%4 JoinCond: (%3)
matching #8=%1 %1=#8 First: %4 Second: %5,
#40=SelectedRowSet #40(#12|#1,%1,%2) matches (%1=1) targets: 23=#40 Source: 64
SMap: (#12=#40[115],%1=#40[43],%2=#40[93]),
%4=RestRowSet #4(#8,#10|%0) matches (#8=1) targets: 235=%4 http://localhost:8180/A/A/D
RemoteCols: (E:265[336],F:288[336],G:312[336]),
%5=SelectedRowSet #5(#12|#1,%1,%2) matches (%1=1) targets: 23=%5 Source: 64 SMap: (#12=%5[115],%1=%5[43],%2=%5[93]))}

```

Notice the OrderedRowSets usually present in computing a join have already been removed, and the greyed-out entries are not required. Because of the RestRowSet, the Build step may be triggered at various points. Notice that the matching condition has been passed into the RestRowSet.

We will now contrast this with the query that references the RestView via the stored view V. To avoid confusion, we have restarted the server (the compiled elements end up in slightly different places if they have been freshly defined).

We begin by examining the Framing objects (precompiled elements) of View V at positions in the shared region 407 to 438 and @0 upwards. Objects such Views, Procedures etc are compiled on definition and when the database is loaded. The generated objects are given uids that don't conflict with file positions in the transaction log. It is quite a list. partly because it contains an instance of the RestView, so is worth constructing just once! You can view the Framing any time after database load as a field of object V, which is at cx.db.objects[407], or if V is referenced in the current Context, cx.obs[407]. Right-click the value and copy it to a Notepad window. Add some white space for readability as below. We omit lines that match the Framing for W:



Name	Value	Type
cx.obs[407].framing	{Framing (23 Table Name=H 23 Definer... Pyrrho.Lev...	Pyrrho.Lev...

```

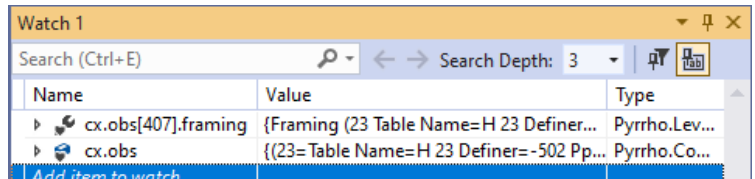
{Framing
(...408 SelectStatement 408 CS=409,
409 CursorSpecification 409 RowType:(@0,@1,@2,@4,@5,@3) Source={select * from W natural join H} Union: 410,
410 QueryExpression 410 RowType:(@0,@1,@2,@4,@5,@3) Left: 415 ,
415 QuerySpecification 415 RowType:(@0,@1,@2,@4,@5,@3) TableExp 418,
416 SqlStar Name=* 416 CONTENT From:415 CONTENT,
418 TableExpression 418 Nuid=425 RowType:(@0,@1,@2,@4,@5|@3) Target: 425,
423 From Name=W 423 RowType:(@0,@1,@2) OrdSpec (0=@0) Target=336,
425 JoinPart 425 RowType:(@0,@1,@2,@4,@5|@3)423 NATURAL INNER join438 on @6 matching @0=@3 @3=@0,
438 From Name=H 438 RowType:(@3,@4,@5) OrdSpec (0=@3) Target=23,
@0 SqlCopy Name=E @0 From:423 Domain INTEGER copy from 265,
@1 SqlCopy Name=F @1 From:423 Domain CHAR copy from 288,
@2 SqlCopy Name=G @2 From:423 Domain CHAR copy from 312,
@3 SqlCopy Name=E @3 From:438 Domain INTEGER copy from 43,
@4 SqlCopy Name=K @4 From:438 Domain CHAR copy from 93,

```

```
@5 SqlCopy Name=M @5 From:438 Domain INTEGER copy from 115,
@6 SqlValueExpr Name= @6 Left:@0 BOOLEAN Right:@3 @6(@0=@3))
Data: (..
407 JoinRowSet 425(@0,@1,@2,@4,@5|@3) targets: 23=438,235=423 JoinCond: (@6) matching @0=@3 @3=@0
First: @7 Second: @8,
418 TableExpRowSet 418(@0,@1,@2,@4,@5|@3) key (@0,@1,@2,@4,@5) targets: 23=438,235=423 Source: 425,
423 RestRowSet 423(@0,@1,@2) targets: 235=423 http://localhost:8180/A/A/D
RemoteCols: (E:265[336],F:288[336],G:312[336]),
425 JoinRowSet 425(@0,@1,@2,@4,@5|@3) targets: 23=438,235=423 JoinCond: (@6) matching @0=@3 @3=@0
First: @7 Second: @8,
438 SelectedRowSet 438(@3,@4,@5) targets: 23=438 Source: 64 SMap: (@3=438[43],@4=438[93],@5=438[115]),
@7 OrderedRowSet @7(@0,@1,@2) key (@0) order (@0) targets: 235=423 Source: 423,
@8 OrderedRowSet @8(@3,@4,@5) key (@3) order (@3) targets: 23=438 Source: 438)
Result 425 Results: (,410 418,415 418,418 418,425 425,438 438)}
```

During parsing, the view definition V is instantiated.


This happens in View.Instance(), so to see it, place a breakpoint at line 143 of View.cs. The parse has resulted in the following new entries in the Context (it is always guaranteed that objects whose uids are file positions or in the executable range do not get modified):



Name	Value	Type
cx.obs[407].framing	{Framing (23 Table Name=H 23 Definer...	Pyrrho.Lev...
cx.obs	{(23= Table Name=H 23 Definer=-502 Pp...	Pyrrho.Co...

```
#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 @0,
#10=SqlCopy Name=F #10 From:#25 Domain CHAR copy from @1,
#12=SqlCopy Name=M #12 From:#25 Domain INTEGER copy from @5,
#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 @2,
%1=SqlCopy Name=K %1 From:#25 Domain CHAR copy from @4)
```

At the end of View.Instance(), the Context contains the following new entries:

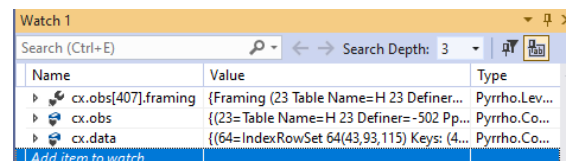


Name	Value	Type
cx.obs[407].framing	{Framing (23 Table Name=H 23 Definer...	Pyrrho.Lev...
cx.obs	{(23= Table Name=H 23 Definer=-502 Pp...	Pyrrho.Co...
cx.data	{(64= IndexRowSet 64(43,93,115) Keys: (4...	Pyrrho.Co...

```
{(.. %2=View Name=V %2 Definer=-502 Ppos=407 Query select * from W natural join H Ppos: 407
Cols (E:#8[336],F:#10[336],G:%0[336],K:%1[23],M:#12[23]) Domain ROW (#8,#10,%0,%1,#12|%13) Display=5
([#8,Domain INTEGER],[#10,Domain CHAR],[#12,Domain INTEGER],[%0,Domain CHAR],[%1,Domain CHAR],
[%13,Domain INTEGER]) Targets: 23,235,
%3=CursorSpecification %3 RowType:(#8,#10,%0,%1,#12,%13) Source={select * from W natural join H} Union: %5,
%4=SelectStatement %4 CS=%3,
%5=QueryExpression %5 RowType:(#8,#10,%0,%1,#12,%13) Left: %6 ,
%6=QuerySpecification %6 RowType:(#8,#10,%0,%1,#12,%13) TableExp %8,
%7=SqlStar Name=* %7 CONTENT From:%6 CONTENT,
%8=TableExpression %8 Nuid=%10 RowType:(#8,#10,%0,%1,#12|%13) Target: %10,
%9=From Name=W %9 RowType:(#8,#10,%0) OrdSpec (0=#8) Target=336,
%10=JoinPart %10 RowType:(#8,#10,%0,%1,#12|%13)%9 NATURAL INNER join%12 on %17 matching #8=%13 %13=#8,
%11=SqlCopy Name=F %11 From:%9 Domain CHAR copy from 288,
%12=From Name=H %12 RowType:(%13,%1,#12) OrdSpec (0=%13) Target=23,
%13=SqlCopy Name=E %13 From:%12 Domain INTEGER copy from 43,
%14=SqlCopy Name=K %14 From:%12 Domain CHAR copy from 93,
%16=SqlCopy Name=M %16 From:%12 Domain INTEGER copy from 115,
%17=SqlValueExpr Name= %17 Left:#8 BOOLEAN Right:%13 %17(#8=%13),
%19=SqlCopy Name=G %19 From:%9 Domain CHAR copy from 312)}
```

And cx.data's entries in the # and % ranges are

```
{(..#25=JoinRowSet %10(#8,#10,#12,%0,%1|%13) key (#8,#10,%0,%1,#12,%13)
where (#34) matches (#8=1)
targets: 23=%12,235=%9 JoinCond: (%17) matching #8=%13 %13=#8
First: %18 Second: %15,..
%8=TableExpRowSet %8(#8,#10,%0,%1,#12|%13) key (#8,#10,%0,%1,#12) targets: 23=%12,235=%9 Source: %10,
%9=RestRowSet %9(#8,#10,%0) key (#8,#10,%0) targets: 235=%9 http://localhost:8180/A/A/D
RemoteCols: (E:265[336],F:288[336],G:312[336]),
%10=JoinRowSet %10(#8,#10,%0,%1,#12|%13) key (#8,#10,%0,%1,#12,%13) targets: 23=%12,235=%9 JoinCond: (%17)
matching #8=%13 %13=#8 First: %18 Second: %15,
%12=SelectedRowSet %12(%13,%1,#12) key (%13,%1,#12) targets: 23=%12 Source: 64
SMap: (%12=%12[115],%1=%12[93],%13=%12[43]),
%15=OrderedRowSet %15(%13,%1,#12) key (%13) order (%13) targets: 23=%12 Source: %12,
```



Name	Value	Type
cx.obs[407].framing	{Framing (23 Table Name=H 23 Definer...	Pyrrho.Lev...
cx.obs	{(23= Table Name=H 23 Definer=-502 Pp...	Pyrrho.Co...
cx.data	{(64= IndexRowSet 64(43,93,115) Keys: (4...	Pyrrho.Co...


```
%18=OrderedRowSet %18(#8,#10,%0) key (#8) order (#8) targets:
235=%9 Source: %9))
```

Oddly, the two versions of the JoinRowSet at #25 and %10 do not match. But after rowset construction and Review, e.g. when traversal is about to start (line 5377 of Start.cs again) we have that cx.results is #7, and (again looking at the same ranges):

Watch 1		
Search (Ctrl+E)		
Search Depth: 3		
Name	Value	Type
cx.obs[407].framing	{Framing (23 Table Name=H 23 Definer...	Pyrrho.Lev...
cx.obs	{(23=Table Name=H 23 Definer=-502 Pp...	Pyrrho.Co...
cx.data	{{(64=IndexRowSet 64(43,93,115) Keys: (4...	Pyrrho.Co...
cx.result	0x5000000000000007	long
Add item to watch		

```
{(#7=SelectRowSet #7(#8,#10,#12,#14) targets: 23=%15,235=%18 Source: #20,
#20=TableExpRowSet #20(#8,#10,#12|%0,%1) key (#8,#10) where (#34) matches (#8=1) targets: 23=%15,235=%18
Source: %10,
#25=JoinRowSet %10(#8,#10,#12,%0,%1|%13) key (#8,#10,%0,%1,#12,%13) where (#34) matches (#8=1)
targets: 23=%15,235=%18 JoinCond: (%17) matching #8=%13 %13=#8 First: %18 Second: %15,
%8=TableExpRowSet %8(#8,#10,%0,%1,#12|%13) key (#8,#10,%0,%1,#12) targets: 23=%15,235=%18 Source: %10,
%9=RestRowSet %9(#8,#10,%0) key (#8,#10,%0) matches (#8=1) targets: 235=%9 http://localhost:8180/A/A/D
RemoteCols: (E:265[336],F:288[336],G:312[336])),
%10=JoinRowSet %10(#8,#10,%0,%1,#12|%13) key (#8,#10,%0,%1,#12,%13) matches (#8=1,%13=1) targets: 23=%15,235=%18
JoinCond: (%17) matching #8=%13 %13=#8 First: %18 Second: %15,
%12=SelectedRowSet %12(%13,%1,#12) key (%13,%1,#12) matches (%13=1) targets: 23=%12 Source: 64
SMap: (%12=%12[115],%1=%12[93],%13=%12[43]),
%15=SelectedRowSet %15(%13,%1,#12) key (%13,%1,#12) matches (%13=1) targets: 23=%15 Source: 64
SMap: (%12=%15[115],%1=%15[93],%13=%15[43]),
%18=RestRowSet %18(#8,#10,%0) key (#8,#10,%0) matches (#8=1) targets: 235=%18 http://localhost:8180/A/A/D
RemoteCols: (E:265[336],F:288[336],G:312[336]))}
```

This matches the rowset collection for the ad-hoc join query above. Traversal of either query then obviously gives a single row as shown in the screenshot above. In both cases only one row is fetched from database A.

The next operation in the paper is the update of the view V (we recall that V is a join of a local table with a remote table).

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

Looking at the context in B just as the Execution is about to start (break at line 6709 of Parser.cs), and omitting the entries that are guaranteed to be the same as before:

```
{(#1=UpdateSearch #1 Nuid=#8 Target: 407,#8=From Name=V #8 RowType:(%0,%1,%2,%3,%4)
Assigs:(UpdateAssignment Vb1: %1 Val: #16=True) Filter:(%0=2) Where:(#35=True) Target=%5,
#16=Elizabeth,
#35=SqlValueExpr Name= #35 From:#8 Left:%0 BOOLEAN Right:#36 #35(%0=#36),
#36=2,
%0=SqlCopy Name=E %0 From:#8 Domain INTEGER copy from @0,
%1=SqlCopy Name=F %1 From:#8 Domain CHAR copy from @1,
%2=SqlCopy Name=G %2 From:#8 Domain CHAR copy from @2,
%3=SqlCopy Name=K %3 From:#8 Domain CHAR copy from @4,
%4=SqlCopy Name=M %4 From:#8 Domain INTEGER copy from @5,
%5=View Name=V %5 Definer=-502 Ppos=407 Query select * from W natural join H Ppos: 407
Cols (E:%0[336],F:%1[336],G:%2[336],K:%3[23],M:%4[23]) Domain ROW (%0,%1,%2,%3,%4|%16) Display=5
([%0,Domain INTEGER],[%1,Domain CHAR],[%2,Domain CHAR],[%3,Domain CHAR],[%4,Domain INTEGER],
[%16,Domain INTEGER]) Targets: 23,235,
%6=CursorSpecification %6 RowType:(%0,%1,%2,%3,%4,%16) Source={select * from W natural join H} Union: %8,
%7=SelectStatement %7 CS=%6,
%8=QueryExpression %8 RowType:(%0,%1,%2,%3,%4,%16) Left: %9 ,
%9=QuerySpecification %9 RowType:(%0,%1,%2,%3,%4,%16) TableExp %11,
%10=SqlStar Name=* %10 CONTENT From:%9 CONTENT,
%11=TableExpression %11 Nuid=%13 RowType:(%0,%1,%2,%3,%4,%16) Target: %13,
%12=From Name=W %12 RowType:(%0,%1,%2) OrdSpec (0=%0) Target=336,
%13=JoinPart %13 RowType:(%0,%1,%2,%3,%4,%16)%12 NATURAL INNER join%15 on %20 matching %0=%16 %16=%0,
%14=SqlCopy Name=F %14 From:%12 Domain CHAR copy from 288,
%15=From Name=H %15 RowType:(%16,%3,%4) OrdSpec (0=%16) Target=23,
%16=SqlCopy Name=E %16 From:%15 Domain INTEGER copy from 43,
%17=SqlCopy Name=K %17 From:%15 Domain CHAR copy from 93,
%19=SqlCopy Name=M %19 From:%15 Domain INTEGER copy from 115,
%20=SqlValueExpr Name= %20 Left:%0 BOOLEAN Right:%16 %20(%0=%16),
%22=SqlCopy Name=G %22 From:%12 Domain CHAR copy from 312)}
{(#8=JoinRowSet %13(%0,%1,%2,%3,%4,%16) key (%0,%1,%2,%3,%4,%16) where (#35) matches (%0=2,%16=2)
targets: 23=%18,235=%21 Assigs:(UpdateAssignment Vb1: %1 Val: #16=True) JoinCond: (%20)
matching %0=%16 %16=%0 First: %21 Second: %18
%11=TableExpRowSet %11(%0,%1,%2,%3,%4,%16) key (%0,%1,%2,%3,%4) targets: 23=%18,235=%21 Source: %13,
%12=RestRowSet %12(%0,%1,%2) key (%0,%1,%2) matches (%0=2) targets: 235=%12
Assigs:(UpdateAssignment Vb1: %1 Val: #16=True) http://localhost:8180/A/A/D
RemoteCols: (E:265[336],F:288[336],G:312[336])),
%13=JoinRowSet %13(%0,%1,%2,%3,%4,%16) key (%0,%1,%2,%3,%4,%16) where (#35) matches (%0=2,%16=2)
targets: 23=%18,235=%21 Assigs:(UpdateAssignment Vb1: %1 Val: #16=True) JoinCond: (%20) matching %0=%16 %16=%0
First: %21 Second: %18,
%15=SelectedRowSet %15(%16,%3,%4) key (%16,%3,%4) matches (%16=2) targets: 23=%15 Source: 64
```

```
SQMap: (%3=%15[93],%4=%15[115],%16=%15[43]),
%18=SelectedRowSet %18(%16,%3,%4) key (%16,%3,%4) matches (%16=2) targets: 23=%18 Source: 64
SQMap: (%3=%18[93],%4=%18[115],%16=%18[43]),
%21=RestRowSet %21(%0,%1,%2) key (%0,%1,%2) matches (%0=2) targets: 235=%21
Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) http://localhost:8180/A/A/D
RemoteCols:(E:265[336],F:288[336],G:312[336]))}
```

The JoinRowSet.Update() method discussed in the Test 12 document traverses the set of targets and their rowsets. Note that in this case there are no UpdateAssignments for %18. so this operand will have no changes. %21 is a RestRowSet, which has its own Update() method. Place a breakpoint at line 5254 of RowSet.cs. Since the metadata does not contain the URL keyword, we create a RESTActivation and return it to JoinRowSet.Update, where we examine the returned value ta. The JoinRowSet.Update method starts a traversal at line 386, and for the RestRowSet %21 performs a WebRequest as before: it retrieves a single row and constructs a RestCursor {(%0=2,%1=Betty,%2=Boop) %21}.

When we get to line 393 of Join.cs the second time (ignoring the no-op for the TableActivation for %18), we can step into ta.EachRow() to see what it does.

At line 566 of Activation.cs, we have the cursor shown above it retrieves the url information from metadata and constructs a POST request at lines starting at line 599. Place a break at line 635, where we see that sql is {update D set F='Elizabeth' where E=2} which is sent to the HTTP service by the RoundTrip method.

The PyrrhoCmd program reports that nothing has changed in B (that is true, since the change is to database A).

We perform a select * from V to check that everything has worked.

```
HTTP service started on port 8180
http://localhost:8180/A/A select E,F,G from D where E=2
-> 1 rows
http://localhost:8180/A/A update D set F='Elizabeth' where E=2
-> OK
http://localhost:8180/A/A select E,F,G from D
-> 2 rows
```

```
SQL> update v set f='Elizabeth' where e=2
0 records affected in B
SQL> select * from v
|-|-----|----|-----|-----|
|E|F      |G   |K      |M      |
|-|-----|----|-----|-----|
|1|Joe     |Soap|Cleaner|12500  |
|2|Elizabeth|Boop|Manager|31400  |
|-|-----|----|-----|-----|
SQL>
```

Delete from remote views and joins works in a similar way.