# All of the details for the RESTView examples

Malcolm Crowe, 24 March 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 29 March 2021 (some images have different dates), 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.

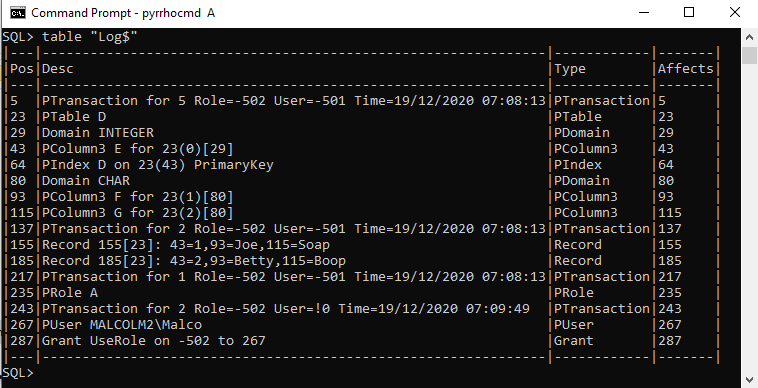
Database A:

**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*"**



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**

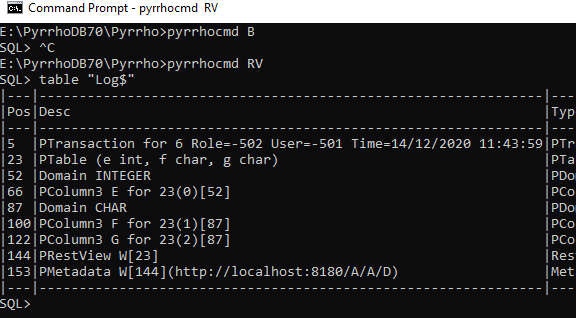
**'http://localhost:8180/A/A/D']**

**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. The Framing for the RestView is just the anonymous table definition:

{Framing Obs:

(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)

Result 0 Results: ()}

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:

{(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,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} 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 0x7000000000000).

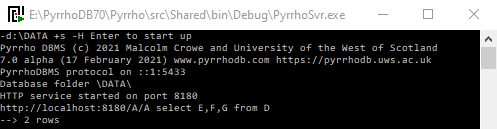
The next step is the creation of rowsets. The RowSets are built starting from the CursorSpecification #1. Views generally are compiled objects whose rowsets were prepared when the view definition was loaded into the database, and need to be “instanced” for the current query as we will see for V later. 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=144 Source: #15,

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

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

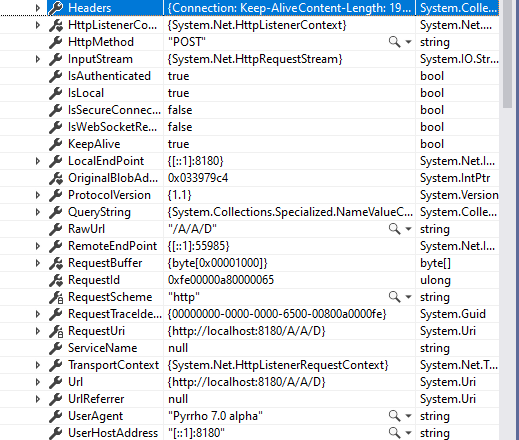
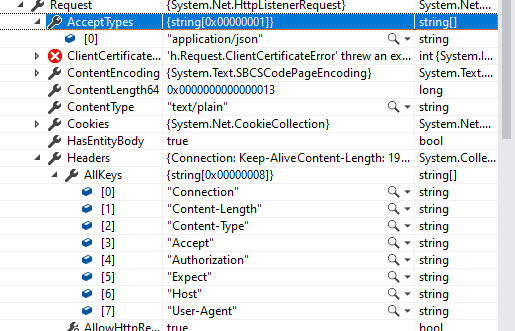
When a rowset is traversed, Pyrrho first checks to see if the RowSet needs to be built. For RestRowSet, the Build method prepares an HTTP POST request to be sent to the given URL, and we can see this because of the -H flag.



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 agent[[1]](#footnote-1). (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)

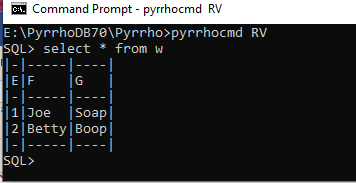


After unpacking the Authorization credentials and database name A, it checks that the URL extension is not .htm, and if not, it creates a connection string and starts a transaction on the requested database. Next, it detects the text/plain content and selects the SqlWebOutput variant of its service. 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 (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 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]) 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,

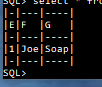
%2=SqlCopy Name=G %2 From:#15 Domain CHAR copy from 122)}

{(#10=TableExpRowSet #10(%0,%1,%2) key (%0,%1,%2) where (#24) matches (%0=1) targets: 23=144 Source: #15,

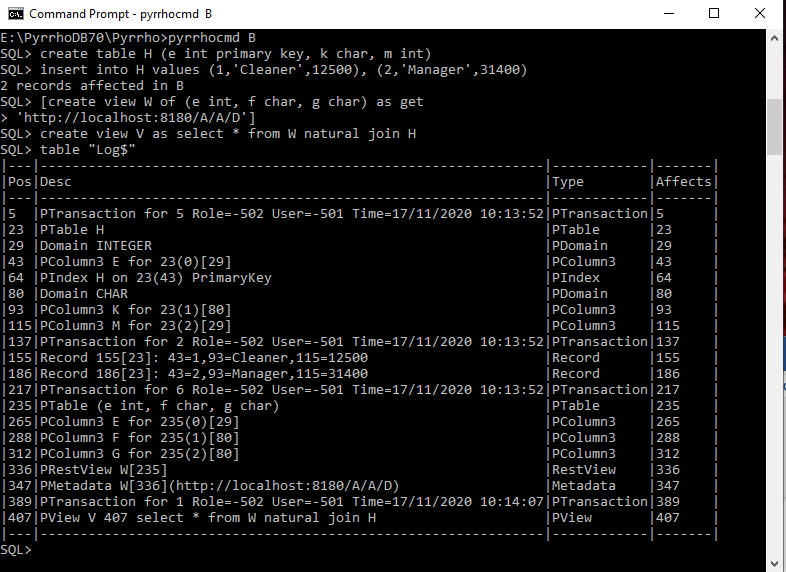
#15=RestRowSet #15(%0,%1,%2) where (#24) matches (%0=1) targets: 23=144 http://localhost:8180/A/A/D

RemoteCols:(E=66,F=100,G=122))}

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”, and the redundant repetition is because for the moment it is both a match and a where)

## 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.

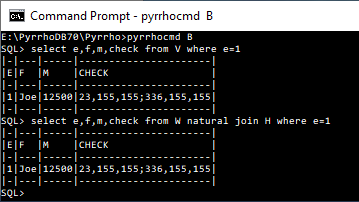
As in section 1, we will look in detail at the parsing of the select statement from the client, which in the paper was

**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.

At the end of parsing the CursorSpecification, the context contains

At 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])

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]) ,

#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|%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) targets: 23=64 Keys: (43,93,115),

#7=SelectRowSet #7(#8,#10,#12,#14) targets: 23=64,235=336 Source: #20,

#20=TableExpRowSet #20(#8,#10|#12,%0,%2,%1) key (#8,#10) where (#49) matches (%1=1) targets: 23=64,235=336

Source: #27,

#25=RestRowSet #25(#8,#10|%0) targets: 235=336 http://localhost:8180/A/A/D RemoteCols:(E=265,F=288,G=312),

#27=JoinRowSet #27(#8,#10|#12,%0,%2,%1) where (#49) targets: 23=64,235=336 JoinCond: (%3) matching #8=%1 %1=#8

First: %4 Second: %5,

#40=SelectedRowSet #40(#12|%1,%2) targets: 23=64 Source: 64,

%4=OrderedRowSet %4(#8,#10|%0) key (#8) order (#8) targets: 235=336 Source: #25,

%5=OrderedRowSet %5(#12|%1,%2) key (%1) order (%1) targets: 23=64 Source: #40)}

During Review, the OrderedRowSet steps are removed, as the matches condition on the JoinPart means that only one row is fetched from either of the operands. So, before traversal the RowSets are:

{(64=IndexRowSet 64(43,93,115) targets: 23=64 Keys: (43,93,115),

#7=SelectRowSet #7(#8,#10,#12,#14) targets: 23=64,235=336 Source: #20,

#20=TableExpRowSet #20(#8,#10|#12,%0,%2,%1) key (#8,#10) where (#49) matches (%1=1) targets: 23=64,235=336

Source: #27,

#25=RestRowSet #25(#8,#10|%0) matches (#8=1) targets: 235=336 http://localhost:8180/A/A/D

RemoteCols:(E=265,F=288,G=312),

#27=JoinRowSet #27(#8,#10|#12,%0,%2,%1) where (#49) matches (%1=1) targets: 23=64,235=336 JoinCond: (%3)

matching #8=%1 %1=#8 First: %4 Second: %5,

#40=SelectedRowSet #40(#12|%1,%2) matches (%1=1) targets: 23=64 Source: 64,

%4=RestRowSet %4(#8,#10|%0) matches (#8=1) targets: 235=336 http://localhost:8180/A/A/D

RemoteCols:(E=265,F=288,G=312),

%5=SelectedRowSet %5(#12|%1,%2) matches (%1=1) targets: 23=64 Source: 64)}

Notice that the entries #25 and #40 here are no longer required, and the matching condition is being passed into the RestRowSet.

We now contrast this with the query that references the RestView via the stored view V. But first, here are the Framing objects (precompiled elements) of View V at positions in the shared region 408 to 438 and @0 upwards (compiled when the database is loaded with generated uids that don’t conflict with uid positions in the tansaction log). It is quite a list. partly because it contains an instance of the RestView, so is worth constructing just once!

{Framing (e int, f char, g char) =(235,);\*=(416,);E=(@3,);F=(@1,);G=(@2,);H=(438,E=(@3,);K=(@4,);M=(@5,););

K=(@4,);M=(@5,);W=(423,E=(@0,);F=(@1,);G=(@2,);); 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,F=288,G=312)

Domain TABLE (265,288,312)([265,Domain INTEGER],[288,Domain CHAR],[312,Domain CHAR]) ,

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: (64 IndexRowSet 64(43,93,115) targets: 23=64

Keys: (43,93,115),

407 JoinRowSet 425(@0,@1,@2,@4,@5|@3) targets: 23=64,235=336 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=64,235=336 Source: 425,

423 RestRowSet 423(@0,@1,@2) targets: 235=336 http://localhost:8180/A/A/D RemoteCols:(E=265,F=288,G=312),

425 JoinRowSet 425(@0,@1,@2,@4,@5|@3) targets: 23=64,235=336 JoinCond: (@6) matching @0=@3 @3=@0 First: @7

Second: @8,

438 SelectedRowSet 438(@3,@4,@5) targets: 23=64 Source: 64,

@7 OrderedRowSet @7(@0,@1,@2) key (@0) order (@0) targets: 235=336 Source: 423,

@8 OrderedRowSet @8(@3,@4,@5) key (@3) order (@3) targets: 23=64 Source: 438)

Result 425 Results: (,410 418,415 418,418 418,425 425,438 438)}

We will see that the same optimisations are carried out and the computation of the result is done the same way as with the expanded version shown above. The result of parsing is as follows (omitting the entries shown above):

{(#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|%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) targets: 23=64 Keys: (43,93,115),

#7=SelectRowSet #7(#8,#10,#12,#14) targets: 23=64,235=336 Source: #20,

#20=TableExpRowSet #20(#8,#10|#12,%0,%2,%1) key (#8,#10) where (#49) matches (%1=1) targets: 23=64,235=336

Source: #27,

#25=RestRowSet #25(#8,#10|%0) matches (#8=1) targets: 235=336 http://localhost:8180/A/A/D

RemoteCols:(E=265,F=288,G=312),

#27=JoinRowSet #27(#8,#10|#12,%0,%2,%1) where (#49) matches (%1=1) targets: 23=64,235=336 JoinCond: (%3)

matching #8=%1 %1=#8 First: %4 Second: %5,

#40=SelectedRowSet #40(#12|%1,%2) matches (%1=1) targets: 23=64 Source: 64,

%4=RestRowSet %4(#8,#10|%0) matches (#8=1) targets: 235=336 http://localhost:8180/A/A/D

RemoteCols:(E=265,F=288,G=312),

%5=SelectedRowSet %5(#12|%1,%2) matches (%1=1) targets: 23=64 Source: 64)}

Once again we see that the OrderedRowSets have been removed during review, and RowSets #25 and #40 are not used. 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, and omitting the entries that are the same as before (quite a few of the entries from the immutable View above have been “instantiated” in uids %0 to %22 because of incorporating properties from the command):

{(#1=UpdateSearch #1 Nuid=#8 Target: 407,

#8=From Name=V #8 RowType:(%0,%1,%2,%3,%4) Assigs:(UpdateAssignment Vbl: %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 265,

%1=SqlCopy Name=F %1 From:#8 Domain CHAR copy from 288,

%2=SqlCopy Name=G %2 From:#8 Domain CHAR copy from 312,

%3=SqlCopy Name=K %3 From:#8 Domain CHAR copy from 93,

%4=SqlCopy Name=M %4 From:#8 Domain INTEGER copy from 115,

%5=View Name=V %5 Definer=-502 Ppos=407 Query select \* from W natural join H Ppos: 407

Cols (E=%0,F=%1,G=%2,K=%3,M=%4) 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=JoinPart %6 RowType:(%0,%1,%2,%3,%4|%16)%13 NATURAL INNER join%15 on %20 matching %0=%16 %16=%0,

%7=TableExpression %7 Nuid=%6 RowType:(%0,%1,%2,%3,%4|%16) Target: %6,

%8=QuerySpecification %8 RowType:(%0,%1,%2,%3,%4,%16) TableExp %7,

%9=QueryExpression %9 RowType:(%0,%1,%2,%3,%4,%16) Left: %8 ,

%10=CursorSpecification %10 RowType:(%0,%1,%2,%3,%4,%16) Source={select \* from W natural join H} Union: %9,

%11=SelectStatement %11 CS=%10,

%12=SqlStar Name=\* %12 CONTENT From:%8 CONTENT,

%13=From Name=W %13 RowType:(%0,%1,%2) OrdSpec (0=%0) Target=336,

%14=SqlCopy Name=F %14 From:%13 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:%6 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:%13 Domain CHAR copy from 312)}

{(64=IndexRowSet 64(43,93,115) targets: 23=64 Keys: (43,93,115),

#8=JoinRowSet %6(%0,%1,%2,%3,%4|%16) key (%0,%1,%2,%3,%4,%16) matches (%0=2) targets: 23=64,235=336

Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) JoinCond: (%20) matching %0=%16 %16=%0 First: %21 Second: %18,

%6=JoinRowSet %6(%0,%1,%2,%3,%4|%16) key (%0,%1,%2,%3,%4,%16) matches (%0=2) targets: 23=64,235=336 Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) JoinCond: (%20) matching %0=%16 %16=%0 First: %21 Second: %18,

%7=TableExpRowSet %7(%0,%1,%2,%3,%4|%16) key (%0,%1,%2,%3,%4) targets: 23=64,235=336 Source: %6,

%13=RestRowSet %13(%0,%1,%2) key (%0,%1,%2) matches (%0=2) targets: 235=336

Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) http://localhost:8180/A/A/D RemoteCols:(E=265,F=288,G=312),

%15=SelectedRowSet %15(%16,%3,%4) key (%16,%3,%4) matches (%0=2) targets: 23=64 Source: 64,

%18=OrderedRowSet %18(%16,%3,%4) key (%16) order (%16) matches (%0=2) targets: 23=64 Source: %15,

%21=RestRowSet %21(%0,%1,%2) key (%0,%1,%2) matches (%0=2) targets: 235=336

Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) http://localhost:8180/A/A/D RemoteCols:(E=265,F=288,G=312))}

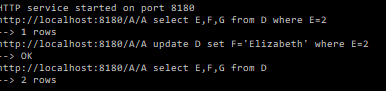
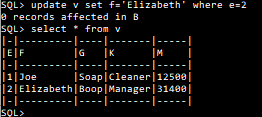
The JoinRowSet %6 receives the Update. Split computes the explicit rows for copies a and b of the join operands:

a.explRows<- {(0=(185, (%0=2,%1=Betty,%2=Boop) %21))}

b.explRows<- {(0=(186, (%16=2,%3=Manager,%4=31400) %18))}

However the second will be ignored since there are no UpdateAssignments for %18. RestRowSet %21.Update(a) accesses the cursor for %21 above,which has a RemoteTableRow. We compute the modified value list for the RemoteTableRow and call \_Update for it. RemoteTableRow.\_Update retrieves the url information from metadata and constructs a POST request {update D set F='Elizabeth' where E=2} which is sent to the HTTP service.

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

Note that pyrrhocmd reports 0 records affected in B. This is currently correct, since the updated record is in database A. I plan to change the command-line feedback to make it more useful.

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

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

To recover the use of URL-style remote syntax described in the paper, 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)**

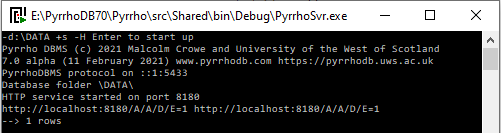
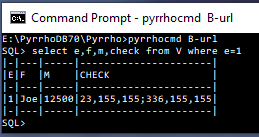
**insert into H values (1,'Cleaner',12500), (2,'Manager',31400)**

**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**

This time we can see that the where condition E=1 has been encoded into the URL request to database A, rather than being sent as SQL in posted data.

The context of this request on database B is very similar to the one above, though the defining positions of V and its compiled components have moved a little:

{(64=IndexRowSet 64(43,93,115) target: 23,

410=TableExpRowSet 421(422,415,416,417,420|414) key (422,415,416,417,420) Source: 428,

412=TableExpRowSet 421(422,415,416,417,420|414) key (422,415,416,417,420) Source: 428,

421=TableExpRowSet 421(422,415,416,417,420|414) key (422,415,416,417,420) Source: 428,

424=OrderedRowSet 424(422,415,416) key (422) order (422) target: 336 Source: 426,

425=OrderedRowSet 425(414,417,420) key (414) order (414) target: 23 Source: 441,

426=RestRowSet 426(422,415,416) key (422,415,416) target: 336 RemoteCols:(E=265,F=288,G=312),

428=JoinRowSet 428(422,415,416,417,420|414) key (422,415,416,417,420,414) JoinCond: (423) matching 414=422 422=414

First: 424 Second: 425,

441=SelectedRowSet 441(414,417,420) key (414,417,420) 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|414) key (#8,#10,%0,%1,#12) where (#34) matches (#8=1) Source: %2,

%2=JoinRowSet %2(#8,#10,%0,%1,#12|414) key (#8,#10,%0,%1,#12,414) where (#34) matches (414=1,#8=1) JoinCond: (%8)

matching 414=#8 #8=414 First: %3 Second: %4,

%3=RestRowSet %3(#8,#10,%0) key (#8,#10,%0) where (#34) matches (#8=1) target: 336 RemoteCols:(E=265,F=288,G=312),

%4=SelectedRowSet %4(414,%1,#12) key (414,%1,#12) matches (414=1,#8=1) target: 23 Source: 64,

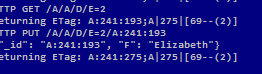
%5=RestRowSet %5(#8,#10,%0) key (#8,#10,%0) where (#34) matches (#8=1) target: 336 RemoteCols:(E=265,F=288,G=312),

%6=SelectedRowSet %6(414,%1,#12) key (414,%1,#12) matches (414=1,#8=1) target: 23 Source: 64)}

The next operation is

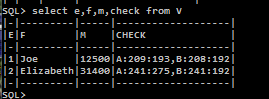
**update v set f='Elizabeth' where e=2**





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:

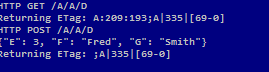


The next operation is an Insert into the View/RestView/Join combination:

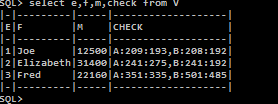
**[insert into v(e,f,g,k,m)**

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



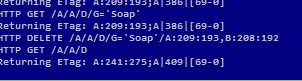


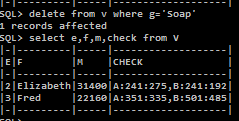
And again verifying the view from B:



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

**delete from v where g='Soap'**





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. [↑](#footnote-ref-1)