# RESTView and RowSet Review in the V7 Version of Pyrrho

Malcolm Crowe, January 2022 (revised April 2023)

## An introduction to RESTViews

The essential idea with RESTView is that the Pyrrho database allows definition of views where the data is held on remote DBMS(s): at present, the only options are Pyrrho and MySQL. The remote DBMS is accessible via SQL statements sent over HTTP with Json responses.

The implementation in the Pyrrho server code is written for the case where the remote DBMS is also Pyrrho. But other DBMS can readily be used, provided a suitable web service mediator is available. Such a service would operate in an ordinary account, would receive requests of the above kind with the ordinary HTTP authentication headers. The service would then use the DBMS’s application library, connecting with the DBMS’s authentication mechanisms, to access the remote DBMS. The Pyrrho distribution includes a version of this sort of client service, for MySQL, called RestifD (source posted on github.com/MalcolmCrowe/restif)

In the use cases considered here, where a query Q references a RESTView V, we assume that (a) materialising V by Extract-transform-load is undesirable for some reason, and (b) we know nothing of the internal details of contributor databases. A single remote select statement defines each RESTView: the agreement with a contributor does not provide any complex protocols, so that for any given Q, we want at most one query to any contributor, compatible with the permissions granted to us by the contributor, namely grant select on the RESTView columns.

Crucially, though, for any given Q, we want to minimise the volume D of data transferred. We can consider how much data Q needs to compute its results, and we rewrite the query to keep D as low as possible. Many such queries (such as the obvious select \* from V) would need all of the data. At the other extreme, if Q only refers to local data (no RESTViews) D is always zero.

For example, a filter on remote columns can be applied on the remote database: and an aggregation of remote data on a single remote source can be carried out on the remote DBMS. But it soon becomes apparent that in more complex cases some transformation of the original query is required. For example, a COUNT of a remote datum supplied separately by several DBMS will need to be implemented as a SUM of the contributions from these, and D will have just one row per contributor. In addition, many SQL functions are differently written in different DBMS, so some details will require a knowledge of the SQL dialect for each contributor. Where ordering and grouping are specified for remote columns, the ordering and grouping can be performed on the remote contributors, and the results merged. In general, some aggregation operations will remain to be performed after the remote data D has been collected.

Although in this document the examples are mostly very simple, we aim to present the analysis in such a way as to demonstrate the applicability of the rules to more complex cases. In other studies in this research, such as the Sierra Leone example, queries can reference multiple stored queries (view definitions) and functions. At present, RESTViews are only found in Pyrrho, but in principle we could have several stages where one RESTView is defined using other RESTViews. We also bear in mind that a query Q might involve joins of RESTViews possibly from the same remote database(s).

There are two types of RESTView corresponding to whether the view has one single contributor or multiple remote databases. In the simple exercises in this document, VV is a RESTview with one contributor, and WW has two. In the implementation, all of the algorithms allow for the general case of a list of contributors (provided by a “using table”, VU here).

This document follows the conventions in the 2022 version of Pyrrho, where we transform rowsets rather than queries. The view definition is precompiled (using framing uids `1,`2 etc), and is instanced (to heap uids %1,%2, etc) for each reference. Instanced column uids are replaced by lexical uids where the given query contains the identifiers explicitly. The select-list of the given query defines the rowType of the result rowSet, which in this version is a property of Domain. The select-list comes first in the syntax, while filters, ordering and grouping come later in the syntax, and are applied to the result rowSet. The Apply method for rowsets passes such tasks to source rowsets wherever possible.

The technical details can be found in the Pyrrho documentation (Pyrrho.pdf and SourceIntro.pdf in the distribution) and were presented at DBKDA 2017 in Barcelona by Fritz Laux and myself. For simple examples, see the definitions of VV, WW, VU and M in the Appendix of this paper. The database RV described there is used for the following illustrations.

The server should be running with the flags +s -H for these demonstrations. We place a break in the debugger just before traversal (line 417 of Start,cs), so that we can see the effect of query processing. For other examples of this sort of analysis, see the later sections of Section 6 of SourceIntro.pdf.

For database DB

**create table T(E int,F char)**

**insert into T values(3,'Three'),(6,'Six'),(4,'Vier'),(6,'Sechs')**

**create role DB**

**grant DB to "MALCOLM1\Malcolm"**

For database DC

**create table U(E int,F char)**

**insert into U values(5,'Five'),(4,'Four'),(8,'Ate')**

**create role DC**

**grant DC to "MALCOLM1\Malcolm"**

For database RV

**create view VV of (E int,F char) as get 'http://localhost:8180/DB/DB/t'**

**create table VU (d char primary key, k int, u char)**

**insert into VU values('B',4,'http://localhost:8180/DB/DB/t')**

**insert into VU values('C',1,'http://localhost:8180/DC/DC/u')**

**create view WW of (E int, D char, K int, F char) as get using VU**

**create table M (e int primary key, n char, unique(n))**

**insert into M values (2,'Deux'),(3,'Trois'),(4,'Quatre')**

**insert into M values (5,'Cinq'),(6,'Six'),(7,'Sept')**

In these details: VV is a single-source RestView, and WW is a multiple-source RESTView whose using table is VU.

Having set up database RV, restart the server for safety.

## Filters

The first aspect of rewriting we consider is filters. If there are some columns of the RESTView that are not used in the given query, there is an obvious reduction, and if a where-condition can be passed to the remote database, this will also reduce the number of rows returned. We begin with such simple filters and return to consider more complex case at the end of this section.

Start with a simple select. In the client **pyrrhocmd RV**, at the SQL> prompt give the command:

**select e,f from vv**

In the debugger, examine cx.obs, and add some white space[[1]](#footnote-1):

{(#1=SelectRowSet #1:Domain TABLE (#8,#10) Display=2[#8, INTEGER],[#10, CHAR] targets: %6=%1 Source: %1,

#8=SqlValue E #8 INTEGER,

#10=SqlValue F #10 CHAR,

%1=RestRowSet %1:Domain TABLE (#8,#10) Display=2[#8, INTEGER],[#10, CHAR] targets: %6=%1 Target=%6 SRow:() http://localhost:8180/DB/DB/t RestView %6 RemoteCols:(%3,%4) RemoteNames:(#8=E,#10=F),

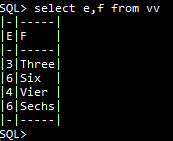
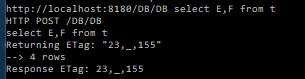
%6=RestView %6 Domain TABLE (%3,%4) Display=2[%3, INTEGER],[%4, CHAR] ViewDef (E int,F char) Ppos: 23,

%8=SelectStatement %8 Union=#1)}

The highlighted object is the instanced version of[[2]](#footnote-2)

RestView 23 Domain (E int,F char) VIEW (`2,`3)[`2,Domain INTEGER],[`3,Domain CHAR] Definer=-502 LastChange=23 ViewDef (E int,F char) Ppos: 23

Continue, and we get the following output in server and client:

  
The first and last lines of server output are from the Pyrrho service thread and the four intervening lines are from the HTTPservice thread.

If we add a filter **where e=4**, we see the following differences:

{(#1=SelectRowSet #1:Domain TABLE (#8,#10) Display=2[#8, INTEGER],[#10, CHAR] where (#27) matches (#8=4) targets: %6=%1 Source: %1,

#8=SqlValue E #8 INTEGER,

#10=SqlValue F #10 CHAR,

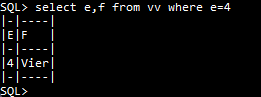
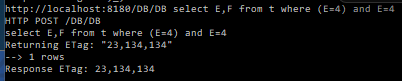
#27=SqlValueExpr #27 BOOLEAN Left:#8 Right:#28 #27(#8=#28),

#28=4,

%1=RestRowSet %1:Domain TABLE (#8,#10) Display=2[#8, INTEGER],[#10, CHAR] where (#27) matches (#8=4) targets: %6=%1 Target=%6 SRow:() http://localhost:8180/DB/DB/t RestView %6 RemoteCols:(%3,%4) RemoteNames:(#8=E,#10=F),

%6=RestView %6 Domain TABLE (%3,%4) Display=2[%3, INTEGER],[%4, CHAR] ViewDef (E int,F char) Ppos: 23,

%8=SelectStatement %8 Union=#1)}

  
And if we have E>=4 we get

{(#1=SelectRowSet #1:Domain TABLE (#8,#10) Display=2[#8, INTEGER],[#10, CHAR] where (#27) targets: %6=%1 Source: %1,

#8=SqlValue E #8 INTEGER,

#10=SqlValue F #10 CHAR,

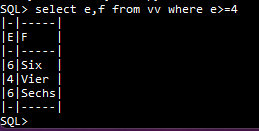
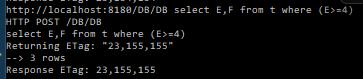
#27=SqlValueExpr #27 BOOLEAN Left:#8 Right:#29 #27(#8>=#29),

#29=4,

%1=RestRowSet %1:Domain TABLE (#8,#10) Display=2[#8, INTEGER],[#10, CHAR] where (#27) targets: %6=%1 Target=%6 SRow:() http://localhost:8180/DB/DB/t RestView %6 RemoteCols:(%3,%4) RemoteNames:(#8=E,#10=F),

%6=RestView %6 Domain TABLE (%3,%4) Display=2[%3, INTEGER],[%4, CHAR] ViewDef (E int,F char) Ppos: 23,

%8=SelectStatement %8 Union=#1)}



We see that fewer rows have transferred from the remote database in both cases.

Before considering WW, let us look at select D,K from VU where k>=4

{(117= Table 117 Domain CONTENT (124,163,186)[124, CHAR],[163, INTEGER],[186, CHAR] Definer=-502 LastChange=186 rows 2 Indexes:((124)146) KeyCols: (124=True),

124=TableColumn 124 CHAR Definer=-502 LastChange=124 -508 Table=117,

163=TableColumn 163 INTEGER Definer=-502 LastChange=163 -511 Table=117,

186=TableColumn 186 CHAR Definer=-502 LastChange=186 -508 Table=117,

#1=SelectRowSet #1:Domain TABLE (#8,#10) Display=2[#8, CHAR],[#10, INTEGER] key (#8) where (#27) targets: 117=#17 From: #17 Source: #17,

#8=SqlCopy #8 CHAR D From:#17 copy from 124,

#10=SqlCopy #10 INTEGER K From:#17 copy from 163,

#17=TableRowSet #17:Domain TABLE (#8,#10,%3) Display=3[#8, CHAR],[#10, INTEGER],[%3, CHAR] Indexes=[(#8)146] key (#8) where (#27) targets: 117=#17 From: #17 Target=117 SRow:(124,163,186) Target:117 VU,

#27=SqlValueExpr #27 BOOLEAN Left:#10 Right:#29 #27(#10>=#29),

#29=4,

%3=SqlCopy %3 CHAR U From:#17 copy from 186,

%5=SelectStatement %5 Union=#1)}

select D,K from VU where D='B'

{(117=Table 117 Domain CONTENT (124,163,186)[124, CHAR],[163, INTEGER],[186, CHAR] Definer=-502 LastChange=186 rows 2 Indexes:((124)146) KeyCols: (124=True),

124=TableColumn 124 CHAR Definer=-502 LastChange=124 -508 Table=117,

163=TableColumn 163 INTEGER Definer=-502 LastChange=163 -511 Table=117,

186=TableColumn 186 CHAR Definer=-502 LastChange=186 -508 Table=117,

#1=SelectRowSet #1:Domain TABLE (#8,#10) Display=2[#8, CHAR],[#10, INTEGER] key (#8) where (#27) matches (#8=B) targets: 117=#17 From: #17 Source: #17,

#8=SqlCopy #8 CHAR D From:#17 copy from 124,

#10=SqlCopy #10 INTEGER K From:#17 copy from 163,

#17=TableRowSet #17:Domain TABLE (#8,#10,%3) Display=3[#8, CHAR],[#10, INTEGER],[%3, CHAR] Indexes=[(#8)146] key (#8) where (#27) matches (#8=B) targets: 117=#17 From: #17 Target=117 SRow:(124,163,186) Target:117 VU,

#27=SqlValueExpr #27 BOOLEAN Left:#8 Right:#28 #27(#8=#28),

#28=B,

%3=SqlCopy %3 CHAR U From:#17 copy from 186,

%5=SelectStatement %5 Union=#1)}

The equality condition is implemented as a low-level match, which is faster than using expression evaluation.[[3]](#footnote-3)

Now consider selections from WW. First consider

Select \* from ww

{(#1=SelectRowSet #1:Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] targets: 373=%15 Source: %15,

#8=SqlStar \* #8 CONTENT,

%1=RestRowSet %1:Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(%3,%6) RemoteNames:(%3=E,%6=F) UsingTableRowSet %10,

%3=SqlValue E %3 INTEGER,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%6=SqlValue F %6 CHAR,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,%5,%13) Display=3[%4, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%16=SelectStatement %16 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

Note that the RestView contains a mixture of local and remote columns. As can be seen here, the values of the local columns are transmitted to the remopte contributor as constants: this enables their use in expressions, so that many aggregations can be performed on the remote servers, and this reduces network traffic.

When we start to consider inserts and updates of restviews, we will see the RestRowSetUsing structure helping to sort out what changes locally and what changes are posted to the remote databases.

We start by considering filters, some of which are handled as above. If the filter is on the usingTable:

Select \* from ww where k=1

{(#1=SelectRowSet #1:Domain TABLE (%3,%4,#24,%6) Display=4[%3, INTEGER],[%4, CHAR],[#24, INTEGER],[%6, CHAR] where (#25) matches (#24=1) targets: 373=%15 Source: %15,

#8=SqlStar \* #8 CONTENT,

#24=SqlCopy #24 INTEGER K From:%10 copy from 163,

#25=SqlValueExpr #25 BOOLEAN Left:#24 Right:#26 #25(#24=#26),

#26=1,

%1=RestRowSet %1:Domain TABLE (%3,%4,#24,%6) Display=4[%3, INTEGER],[%4, CHAR],[#24, INTEGER],[%6, CHAR] targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(%3,%6) RemoteNames:(%3=E,%6=F) UsingTableRowSet %10,

%3=SqlValue E %3 INTEGER,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%6=SqlValue F %6 CHAR,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,#24,%13) Display=3[%4, CHAR],[#24, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) where (#25) matches (#24=1) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (%3,%4,#24,%6) Display=4[%3, INTEGER],[%4, CHAR],[#24, INTEGER],[%6, CHAR] where (#25) matches (#24=1) targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%16=SelectStatement %16 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

When we allow the server to continue, we see that no request is made to the first contributor.

## Aggregations

Next, we look at how aggregations are handled with RESTViews. Previous versions of Pyrrho transformed queries so that results could be aggregated on the local server. Such aggregation is obviously still required, but now the process does not require query transformation. In the next example, we see that queries including aggregations are passed unchanged to the remote servers. The local aggregation of the results is facilitated instead by a small extension to the JSON structures that are returned from the remote servers. The same mechanism also helps with grouped data.

**select count(e) from ww**

{(#1=SelectRowSet #1:Domain TABLE (#8|#14,%4,%5,%6) Display=1[#8, INTEGER],[#14, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] Aggs (#8) targets: 373=%15 Source: %15,

#8=SqlFunction #8 INTEGER COUNT From:#1 COUNT(#14),

#14=SqlValue E #14 INTEGER,

%1=RestRowSet %1:Domain TABLE (#8|#14,%4,%5,%6) Display=1[#8, INTEGER],[#14, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] Aggs (#8) targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(#14,%6) RemoteNames:(#14=E,%6=F) UsingTableRowSet %10,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%6=SqlValue F %6 CHAR,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,%5,%13) Display=3[%4, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (#8) Display=1[#8, INTEGER] targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%18=SelectStatement %18 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

Of course, only one row needs to be returned from the remote servers.

select max(f) from ww where e>4

{(#1=SelectRowSet #1:Domain TABLE (#8|#29,%4,%5,#12) Display=1[#8, CHAR],[#29, INTEGER],[%4, CHAR],[%5, INTEGER],[#12, CHAR] Aggs (#8) where (#30) targets: 373=%15 Source: %15,

#8=SqlFunction #8 CHAR MAX From:#1 MAX(#12),

#12=SqlValue F #12 CHAR,

#29=SqlValue E #29 INTEGER,

#30=SqlValueExpr #30 BOOLEAN Left:#29 Right:#31 #30(#29>#31),

#31=4,

%1=RestRowSet %1:Domain TABLE (#8|#29,%4,%5,#12) Display=1[#8, CHAR],[#29, INTEGER],[%4, CHAR],[%5, INTEGER],[#12, CHAR] Aggs (#8) where (#30) targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(#29,#12) RemoteNames:(#12=F,#29=E) UsingTableRowSet %10,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,%5,%13) Display=3[%4, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (#8) Display=1[#8, CHAR] where (#30) targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%18=SelectStatement %18 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

We see that each contributor sent their maximum, and then we finally get the overall maximum.

select count(\*) from ww where k>2

{(#1=SelectRowSet #1:Domain TABLE (#8|%3,%4,#31,%6) Display=1[#8, INTEGER],[%3, INTEGER],[%4, CHAR],[#31, INTEGER],[%6, CHAR] Aggs (#8) where (#32) targets: 373=%15 Source: %15,

#8=SqlFunction #8 INTEGER COUNT From:#1 COUNT(#14),

#14=1,

#31=SqlCopy #31 INTEGER K From:%10 copy from 163,

#32=SqlValueExpr #32 BOOLEAN Left:#31 Right:#33 #32(#31>#33),

#33=2,

%1=RestRowSet %1:Domain TABLE (#8|%3,%4,#31,%6) Display=2[#8, INTEGER],[#31, INTEGER],[%3, INTEGER],[%4, CHAR],[%6, CHAR] Aggs (#8) where (#32) targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(%3,%6) RemoteNames:(%3=E,%6=F),

%3=SqlValue E %3 INTEGER,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%6=SqlValue F %6 CHAR,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,#31,%13) Display=3[%4, CHAR],[#31, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) where (#32) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (#8,#8) Display=2[#8, INTEGER] targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%18=SelectStatement %18 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

We see the request just goes to one contributor. (A small error is highlighted.)

The following examples have trivial results here, but the strategy trace shows how the method will work in more complex cases. Just as in the above example, no query rewriting is needed if all of the columns in the query are known to the remote view: values from the using table are inserted as constants when required by the syntax.

select sum(e),char\_length(f) as x from ww group by x

{(#1=SelectRowSet #1:Domain TABLE (#8,#15|#12,%4,%5,#27) Display=2[#8, INTEGER],[#15, INTEGER],[#12, INTEGER],[%4, CHAR],[%5, INTEGER],[#27, CHAR] Aggs (#8) groupSpec: #49 groupings (%16) GroupCols(#15) targets: 373=%15 Source: %15 Ambient(#15),

#8=SqlFunction #8 INTEGER SUM From:#1 SUM(#12),

#12=SqlValue E #12 INTEGER,

#15=SqlFunction #15 INTEGER CHAR\_LENGTH From:#1 Alias=X CHAR\_LENGTH(#27) as X,

#27=SqlValue F #27 CHAR,

#49=GroupSpecification #49 Null(%16),

%1=RestRowSet %1:Domain TABLE (#8,#15|#12,%4,%5,#27) Display=2[#8, INTEGER],[#15, INTEGER],[#12, INTEGER],[%4, CHAR],[%5, INTEGER],[#27, CHAR] Aggs (#8) groupSpec: %21 GroupCols(#15) targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(#12,#27) RemoteNames:(#12=E,#27=F),

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,%5,%13) Display=3[%4, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (#8,#15) Display=2[#8, INTEGER],[#15, INTEGER] targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%16=Grouping %16 Domain ROW (#15)[#15, INTEGER] GROUP (#15=0),

%21=GroupSpecification %21 Null(%16),

%24=SelectStatement %24 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

Grouping by a non-remote formula:

select count(\*),k/2 as k2 from ww group by k2

{(#1=SelectRowSet #1:Domain TABLE (#8,#18|%3,%4,#17,%6) Display=2[#8, INTEGER],[#18, INTEGER],[%3, INTEGER],[%4, CHAR],[#17, INTEGER],[%6, CHAR] Aggs (#8) groupSpec: #41 groupings (%16) GroupCols(#18) targets: 373=%15 Source: %15 Ambient(#18),

#8=SqlFunction #8 INTEGER COUNT From:#1 COUNT(#14),

#14=1,

#17=SqlCopy #17 INTEGER K From:%10 copy from 163,

#18=SqlValueExpr #18 INTEGER K2 From:#1 Left:#17 Right:#19 #18(#17/#19),

#19=2,

#41=GroupSpecification #41 Null(%16),

%1=RestRowSet %1:Domain TABLE (#8,#18|%3,%4,#17,%6) Display=2[#8, INTEGER],[#18, INTEGER],[%3, INTEGER],[%4, CHAR],[#17, INTEGER],[%6, CHAR] Aggs (#8) groupSpec: %21 GroupCols(#18) targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(%3,%6) RemoteNames:(%3=E,%6=F),

%3=SqlValue E %3 INTEGER,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%6=SqlValue F %6 CHAR,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,#17,%13) Display=3[%4, CHAR],[#17, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (#8,#18) Display=2[#8, INTEGER],[#18, INTEGER] targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%16=Grouping %16 Domain ROW (#18)[#18, INTEGER] GROUP (#18=0),

%21=GroupSpecification %21 Null(%16),

%24=SelectStatement %24 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

Select avg(e) from ww

{(#1=SelectRowSet #1:Domain TABLE (#8|#12,%4,%5,%6) Display=1[#8, NUMERIC],[#12, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] Aggs (#8) targets: 373=%15 Source: %15,

#8=SqlFunction #8 NUMERIC AVG From:#1 AVG(#12),

#12=SqlValue E #12 INTEGER,

%1=RestRowSet %1:Domain TABLE (#8,#12|%4,%5,%6) Display=2[#8, NUMERIC],[#12, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] Aggs (#8) targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(#12,%6) RemoteNames:(#12=E,%6=F),

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%6=SqlValue F %6 CHAR,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,%5,%13) Display=3[%4, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (#8,#12) Display=2[#8, NUMERIC],[#12, INTEGER] targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%18=SelectStatement %18 Union=#1)}

Text

Description automatically generated

select sum(e)\*sum(e),d from ww group by d

{(#1=SelectRowSet #1:Domain TABLE (#14,#22|#12,%5,%6) Display=2[#14,Domain INTEGER Aggs (#8,#15)],[#22, CHAR],[#12, INTEGER],[%5, INTEGER],[%6, CHAR] Aggs (#8,#15) groupSpec: #38 groupings (%16) GroupCols(#22) targets: 373=%15 Source: %15 Ambient(#22),

#8=SqlFunction #8 INTEGER SUM From:#1 SUM(#12),

#12=SqlValue E #12 INTEGER,

#14=SqlValueExpr #14 Domain INTEGER Aggs (#8,#15) From:#1 Left:#8 Right:#15 #14(#8\*#15),

#15=SqlFunction #15 INTEGER SUM From:#1 SUM(#12),

#22=SqlCopy #22 CHAR D From:%10 copy from 124,

#38=GroupSpecification #38 Null(%16),

%1=RestRowSet %1:Domain TABLE (#14,#22|#12,%5,%6) Display=2[#14,Domain INTEGER Aggs (#8,#15)],[#22, CHAR],[#12, INTEGER],[%5, INTEGER],[%6, CHAR] Aggs (#8,#15) groupSpec: %21 GroupCols(#22) targets: %8=%1 Target=373 SRow:() RestView %8 RemoteCols:(#12,%6) RemoteNames:(#12=E,%6=F),

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%6=SqlValue F %6 CHAR,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

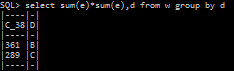
%10=TableRowSet %10:Domain TABLE (#22,%5,%13) Display=3[#22, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(#22)146] key (#22) targets: 117=%10 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (#22,#14,#22) Display=3[#14,Domain INTEGER Aggs (#8,#15)],[#22, CHAR] targets: 373=%15 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%16=Grouping %16 Domain ROW (#22)[#22, CHAR] GROUP (#22=0),

%21=GroupSpecification %21 Null(%16),

 %24=SelectStatement %24 Union=#1)} Text

Description automatically generated

## RESTView and Join

One of the steps in constructing a RESTView is to reduce the view to the columns needed for the given query. When the RESTView is used in a join, we need to ensure that columns needed for the joinCondition are added to the list of needed columns.

Select f,n from ww natural join m

{(436=Table 436 Domain CONTENT (443,486)[443, INTEGER],[486, CHAR] Definer=-502 LastChange=511 rows 6 Indexes:((443)467;(486)511) KeyCols: (443=True,486=True),

443=TableColumn 443 INTEGER Definer=-502 LastChange=443 -511 Table=436,

486=TableColumn 486 CHAR Definer=-502 LastChange=486 -508 Table=436,

#1=SelectRowSet #1:Domain TABLE (#8,#10) Display=2[#8, CHAR],[#10, CHAR] Indexes=[(#10)511] matching (%3=(%16),%16=(%3)) targets: 373=%15,436=#33 Source: #20,

#8=SqlValue F #8 CHAR,

#10=SqlCopy #10 CHAR N From:#33 copy from 486,

#20=JoinRowSet #20:Domain TABLE (%3,%4,%5,#8,#10|%16) Display=5[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR],[#10, CHAR],[%16, INTEGER] matching (%3=(%16),%16=(%3)) targets: 373=%15,436=#33 INNER First: %20 Second: #33 on %3=%16,

#33=TableRowSet #33:Domain TABLE (%16,#10) Display=2[%16, INTEGER],[#10, CHAR] Indexes=[(#10)511,(%16)467] key (%16) order (%16) targets: 436=#33 From: #33 Target=436 SRow:(443,486) Target:436 M,

%1=RestRowSet %1:Domain TABLE (%3,%4,%5,#8) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR] targets: %8=%1 From: #20 Target=373 SRow:() RestView %8 RemoteCols:(%3,#8) RemoteNames:(#8=F,%3=E),

%3=SqlValue E %3 INTEGER,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,%5,%13) Display=3[%4, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 From: #20 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (%3,%4,%5,#8) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR] targets: 373=%15 From: #20 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%16=SqlCopy %16 INTEGER E From:#33 copy from 443,

%20=OrderedRowSet %20:Domain TABLE (%3,%4,%5,#8) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR] key (%3) order (%3) targets: 373=%15 Source: %15,

%21=SelectStatement %21 Union=#1)}

Text

Description automatically generated Text

Description automatically generated

We see that 7 rows have been returned from the RESTView, but only 6 rows in the join. This is correct, as the join has been done on the local table m.

## Filters and Join

A filter can be moved to a factor of a join provided that factor provides all of the columns needed to compute the filter. A nice optimisation here would be to take advantage of available indexes and the join condition. Here a filter on n would select a single row of the join.

Select f,n from ww natural join m where n='Cinq'

{(436=Table 436 Domain CONTENT (443,486)[443, INTEGER],[486, CHAR] Definer=-502 LastChange=511 rows 6 Indexes:((443)467;(486)511) KeyCols: (443=True,486=True),

443=TableColumn 443 INTEGER Definer=-502 LastChange=443 -511 Table=436,

486=TableColumn 486 CHAR Definer=-502 LastChange=486 -508 Table=436,

#1=SelectRowSet #1:Domain TABLE (#8,#10) Display=2[#8, CHAR],[#10, CHAR] Indexes=[(#10)511] where (#42) matches (#10=Cinq) matching (%3=(%16),%16=(%3)) targets: 373=%15,436=#33 Source: #20,

#8=SqlValue F #8 CHAR,

#10=SqlCopy #10 CHAR N From:#33 copy from 486,

#20=JoinRowSet #20:Domain TABLE (%3,%4,%5,#8,#10|%16) Display=5[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR],[#10, CHAR],[%16, INTEGER] where (#42) matches (#10=Cinq) matching (%3=(%16),%16=(%3)) targets: 373=%15,436=#33 INNER First: %20 Second: #33 on %3=%16,

#33=TableRowSet #33:Domain TABLE (%16,#10) Display=2[%16, INTEGER],[#10, CHAR] Indexes=[(#10)511,(%16)467] key (%16) order (%16) where (#42) matches (#10=Cinq) targets: 436=#33 From: #33 Target=436 SRow:(443,486) Target:436 M,

#42=SqlValueExpr #42 BOOLEAN Left:#10 Right:#43 #42(#10=#43),

#43=Cinq,

%1=RestRowSet %1:Domain TABLE (%3,%4,%5,#8) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR] targets: %8=%1 From: #20 Target=373 SRow:() RestView %8 RemoteCols:(%3,#8) RemoteNames:(#8=F,%3=E),

%3=SqlValue E %3 INTEGER,

%4=SqlCopy %4 CHAR D From:%10 copy from 124,

%5=SqlCopy %5 INTEGER K From:%10 copy from 163,

%8=RestView %8 Domain TABLE (%3,%4,%5,%6) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[%6, CHAR] ViewDef (E int, D char, K int, F char) Ppos: 373 UsingTable: 117,

%10=TableRowSet %10:Domain TABLE (%4,%5,%13) Display=3[%4, CHAR],[%5, INTEGER],[%13, CHAR] Indexes=[(%4)146] key (%4) targets: 117=%10 From: #20 Target=117 SRow:(124,163,186) Target:117 VU,

%13=SqlCopy %13 CHAR U From:%10 copy from 186,

%15=RestRowSetUsing %15:Domain TABLE (%3,%4,%5,#8) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR] targets: 373=%15 From: #20 Target=373 SRow:(124,163,186) Template: %1 UsingTableRowSet:%10 UrlCol:%13,

%16=SqlCopy %16 INTEGER E From:#33 copy from 443,

%20=OrderedRowSet %20:Domain TABLE (%3,%4,%5,#8) Display=4[%3, INTEGER],[%4, CHAR],[%5, INTEGER],[#8, CHAR] key (%3) order (%3) targets: 373=%15 Source: %15,

%21=SelectStatement %21 Union=#1)}

This is a sort of functional dependency not special to RESTViews (and not yet in Pyrrho), so we skip the details here.

## Aggregation and Join

If a grouped query uses a join whose non-remote factor has key J, an aggregation operation grouped by G can be shared with the factors of the join by applying a similar aggregation grouped by G∪J. In this example, the join column is the primary key of both factors. If there is no primary key defined, then all columns are needed to form the join.

Select count(\*) from ww natural join m

We could group on the join column to reduce the volume.

## Appendix

### Setup for tests

For database DB

**create table T(E int,F char)**

**insert into T values(3,'Three'),(6,'Six'),(4,'Vier'),(6,'Sechs')**

**create role DB**

**grant DB to "MALCOLM1\Malcolm"**

For database DC

**create table U(E int,F char)**

**insert into U values(5,'Five'),(4,'Four'),(8,'Ate')**

**create role DC**

**grant DC to "MALCOLM1\Malcolm"**

For database RV

**create view VV of (E int,F char) as get 'http://localhost:8180/DB/DB/t'**

**create table VU (d char primary key, k int, u char)**

**insert into VU values('B',4,'http://localhost:8180/DB/DB/t')**

**insert into VU values('C',1,'http://localhost:8180/DC/DC/u')**

**create view WW of (E int, D char, K int, F char) as get using VU**

**create table M (e int primary key, n char, unique(n))**

**insert into M values (2,'Deux'),(3,'Trois'),(4,'Quatre')**

**insert into M values (5,'Cinq'),(6,'Six'),(7,'Sept')**

### Retrieval tests

select \* from vv

select \* from vv where e=6

select ww

select \* from ww where e<6

select \* from ww where k=1

select sum(e) from vv

select count(e) from vv

select count(e) from ww

select count(\*) from ww

select max(f) from ww

select max(f) from ww where e>4

select count(\*) from ww where k>2

select min(f) from ww

select sum(e)\*sum(e),d from ww group by d

select count(\*),k/2 as k2 from ww group by k2

select avg(e) from ww

select f,n from ww natural join m

select e+char\_length(f) as x,n from ww natural join m

select char\_length(f)+char\_length(n) from ww natural join m

select sum(e)+char\_length(max(f)) from ww

select count(\*),e+char\_length(f) as x from ww group by x

select count(\*),e+char\_length(n) as x from ww natural join m group by x

select sum(e)+char\_length(f),f from ww natural join m group by f

select sum(char\_length(f))+char\_length(n) as x,n from w natural join m group by n

Select count(\*) from w natural join m

### Tests on updatability

table v

update v set F='Tri' where E=3

insert into V values (9,'Nine')

table V

delete from V where E=9

update v set F='Tri' where E=3

table V

update w set f='Eight' where e=8

insert into w(D,E,F) values('B',7,'Seven')

table v

table w

delete from w where E=7

update v set f='Ate' where e=8

update w

table v

table w

table "Log$"

1. In these illustrations, domain details are not separately listed. [↑](#footnote-ref-1)
2. RestView framing is always empty, unlike View and other Compiled objects. [↑](#footnote-ref-2)
3. At some future time, the redundant where (#27) should be removed. [↑](#footnote-ref-3)