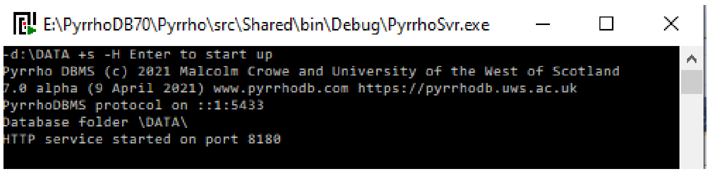
All the details for Test 12

Malcolm Crowe, 9 April 2021

"Test 12" in the PyrrhoTest program includes some simple tests for updatable views. This document traces through the steps of the test using the Visual Studio debugger. We assume throughout that the PyrrhoSvr program is running in another window (or under the Visual Studio debugger). The server flags used for this document are

`PyrrhoSvr -d:\DATA`

(Use \DATA as the folder for the databases. I also used +s and -H for the HTTP service, but that is not needed for Test 12.)



```
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 ::1:5433
Database folder \DATA\
HTTP service started on port 8180
```

The panel shows the command line script for this part of the test with a JSON version of the responses inserted on the lines beginning -> . (Don't input these response lines.)

```
create table p(q int primary key,r char,a int)
create view v as select q,r as s,a from p
insert into v(s) values('Twenty'),('Thirty')
update v set s='Forty two' where q=1
select q,s from v
->[{Q:1,S:'Forty two'},{Q:2,R:'Thirty'}]
select r from p
->[{R:'Forty two'},{R:'Thirty'}]
delete from v where s='Thirty'
select * from p
->[{Q:1,R:'Forty two',A:}]
insert into p(r) values('Fifty')
create table t(s char,u int)
insert into t values('Forty two',42),('Fifty',48)
create view w as select * from t natural join v
update w set u=50,a=21 where q=2
table p
->[{Q:1,R:'Forty two',A:},{Q:2,'R:'Fifty',A:21}]
table t
->[{S:'Forty two',U:42},{S:'Fifty',U:50}]
```

To show the effect of each step in the test, we insert additional commands

`table "Log$"`

(equivalent to **`select * from "Log$"`** note the straight double quotes here).

We start with no database file.

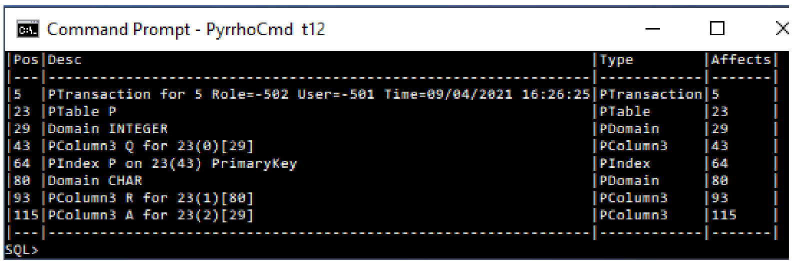
Creating a View

In the client window, start the command line client:

`PyrrhoCmd t12`

This makes the server create an empty database file called t12, 5 bytes long. (It locks it for its exclusive use, so if you want to examine its properties you will need to stop the server.) Unfortunately, stopping and restarting the server will change some of the internal uids used in this test (as would using explicit transactions). For these notes we assume that the server is not restarted and transactions are implicit (so-called autocommit mode). This means the client prompt is always SQL> as shown in the illustrations.

(You may need to change the size of the command window to accommodate the line length of the log entries.)



```
Command Prompt - PyrrhoCmd t12
|Pos|Desc                                                      |Type        |Affects|
|---|----------------------------------------------------------|------------|-------|
|5  |PTransaction for 5 Role=-502 User=-501 Time=09/04/2021 16:26:25|PTransaction|5      |
|23 |PTable P                                                  |PTable      |23     |
|29 |Domain INTEGER                                            |PDomain     |29     |
|43 |PColumn3 Q for 23(0)[29]                                  |PColumn3    |43     |
|64 |PIndex P on 23(43) PrimaryKey                             |PIndex      |64     |
|80 |Domain CHAR                                               |PDomain     |80     |
|93 |PColumn3 R for 23(1)[80]                                  |PColumn3    |93     |
|115|PColumn3 A for 23(2)[29]                                  |PColumn3    |115    |
|---|----------------------------------------------------------|------------|-------|
SQL>
```

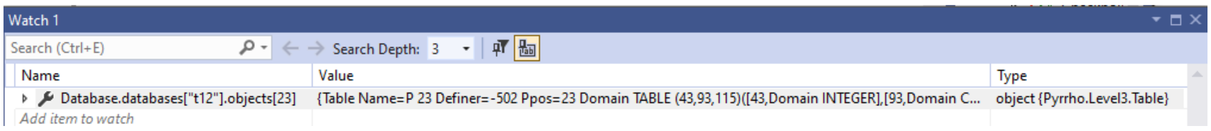
```
create table p(q int primary key,r char,a int)
table "Log$"
```

We see that the effect of the autocommitted transaction is to add 8 “physical records” to the transaction log. We haven’t defined users or roles in this database, so these are system defaults[1].

Following the PTransaction header, we see records defining table P, columns P, Q, R with their domains, and the Index for the table’s primary key. The byte positions of these records in the transaction log are the defining positions (uids) of the database objects created. Many database objects like these define corresponding objects in the server Table, Index, TableColumn etc identified by their uids.

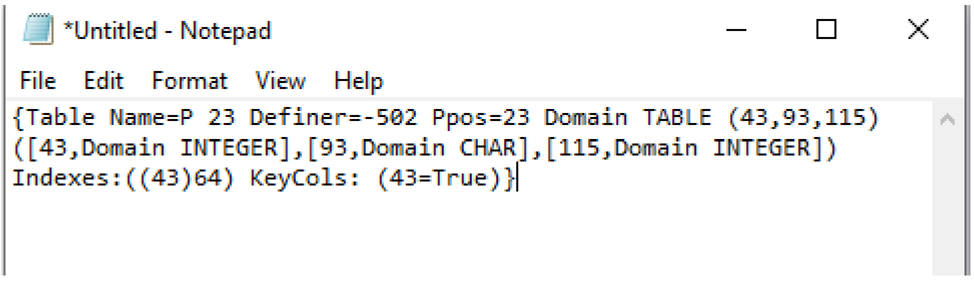
The Watch window allows easy inspection of things, by default displaying the ToString() version and allowing expansion, CopyValue, etc. Pause the server, and in the Watch 1 window, enter

Database.databases["t12"].objects[23]



```
Name                                   Value                                                                                  Type
Database.databases["t12"].objects[23]  {Table Name=P 23 Definer=-502 Ppos=23 Domain TABLE (43,93,115)([43,Domain INTEGER],[93,Domain C...  object {Pyrrho.Level3.Table}
```

Right-clicking the entry and Copy Value allows us to Paste the contents more conveniently in a Notepad window:



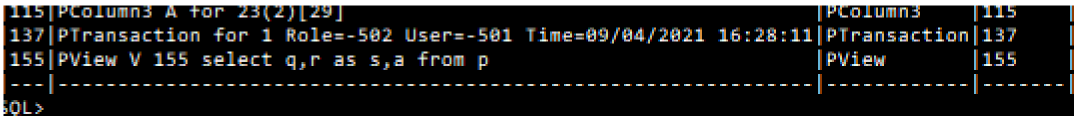
```
{Table Name=P 23 Definer=-502 Ppos=23 Domain TABLE (43,93,115)
([43,Domain INTEGER],[93,Domain CHAR],[115,Domain INTEGER])
Indexes:((43)64) KeyCols: (43=True)}
```

This shows the table with its column uids and domain, and the primary key information. Note the TABLE domain with its columns.

Some database objects such as Procedures and Views, also have precompiled objects in the server as we will see next. Allow the server to continue:

`create view v as select q,r as s,a from p`

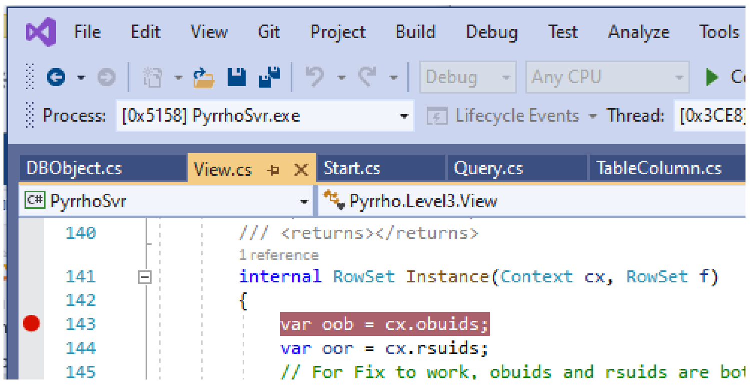
This transaction has two physical records, a PTransaction and a PView. If we try **`table "Log$"`** now we see two further entries:



```
|115|PColumn3 A for 23(2)[29]                                      |PColumn3    |115   |
|137|PTransaction for 1 Role=-502 User=-501 Time=09/04/2021 16:28:11|PTransaction|137   |
|155|PView V 155 select q,r as s,a from p                          |PView       |155   |
|---|--------------------------------------------------------------|------------|------|
SQL>
```

As we can see, the PView simply records the SQL definition of the View. The server compiles this definition and creates a set of shareable “framing” objects, including a table that defines the view’s domain, the query and rowset objects, This will avoid recompiling the View every time it is used.

Let us look at the objects and rowsets defined in the Framing for view V. We could pause the server and access these in the Watch window as above, but instead let us set a breakpoint at the beginning of the View.Instance() method, line 143 of View.cs as shown.



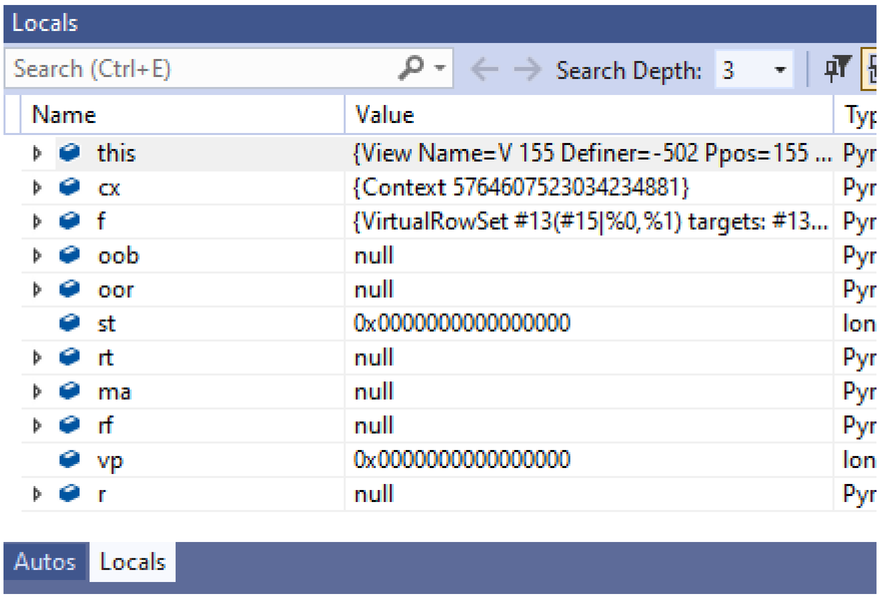
```
140         /// <returns></returns>
            1 reference
141         internal RowSet Instance(Context cx, RowSet f)
142         {
143             var oob = cx.obuids;
144             var oor = cx.rsuids;
145             // For Fix to work, obuids and rsuids are bo
```

[1] This means that the database can only be accessed using the same the same account that started the server.

In order to reach this breakpoint, let us start the next command in the test. This uses the view V to *insert* data in the table P. (Of course this is not how views are normally used!)

```
insert into v(s) values('Twenty'),('Thirty')
```

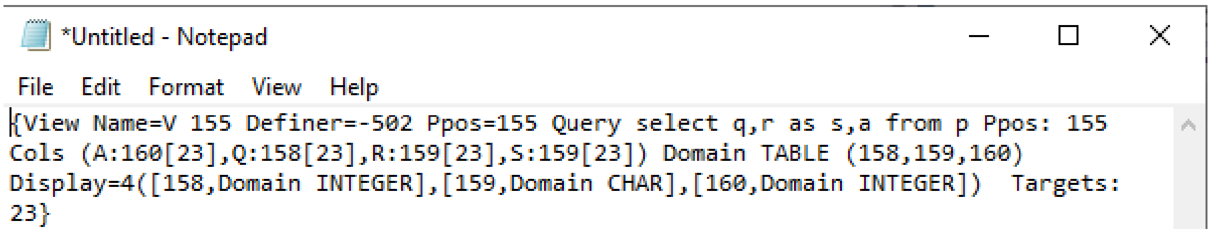
(This command does not set a value for the primary key column Q, leaving it to be supplied by Pyrrho's autokey feature.) But just now we are examining the View's Framing object. If the Locals window of the debugger is empty, single step one instruction. In the Locals window of the debugger we see



Search Depth: 3

Name	Value	Typ
this	{View Name=V 155 Definer=-502 Ppos=155 ...	Pyr
cx	{Context 5764607523034234881}	Pyr
f	{VirtualRowSet #13(#15|%0,%1) targets: #13...	Pyr
oob	null	Pyr
oor	null	Pyr
st	0x0000000000000000	lon
rt	null	Pyr
ma	null	Pyr
rf	null	Pyr
vp	0x0000000000000000	lon
r	null	Pyr

Right-click on the first line here, and copy the value into a Notepad:



```
{View Name=V 155 Definer=-502 Ppos=155 Query select q,r as s,a from p Ppos: 155
Cols (A:160[23],Q:158[23],R:159[23],S:159[23]) Domain TABLE (158,159,160)
Display=4([158,Domain INTEGER],[159,Domain CHAR],[160,Domain INTEGER])  Targets:
23}
```

We see that the View has a different Domain from the table, as the column uids are different. We see a catalogue of the view's columns with the view uids and their home table uid.

In the Locals window, expand `this`, right-click the framing field and copy its value into a notepad window. After adding some line breaks, this contains the following:

```
{Framing (23 Table Name=P 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,
 157 CursorSpecification 157 RowType:(158,159,160) Where: Source={select q,r as s,a from p}
        Union: 161,
 158 SqlCopy Name=Q 158 From:164 Domain INTEGER copy from 43,
 159 SqlCopy Name=R 159 From:164 Alias=S Domain CHAR copy from 93,
 160 SqlCopy Name=A 160 From:164 Domain INTEGER copy from 115,
 161 QueryExpression 161 RowType:(158,159,160) Where: Left: 162 ,
 162 QuerySpecification 162 RowType:(158,159,160) Where: TableExp 163,
 163 TableExpression 163 Nuid=164 RowType:(158,159,160) Where: Target: 164,
 164 From Name=P 164 RowType:(158,159,160) Where: Target=23)
Data:
(64 IndexRowSet 64(43,93,115) Keys: (43,93,115),
 163 TableExpRowSet 163(158,159,160) key (158,159,160) targets: 23=164 Source: 164,
 164 SelectedRowSet 164(158,159,160) key (158,159,160) targets: 23=164 Source: 64
        SQMap: (158=164[43],159=164[93],160=164[115]))
Result 164 Results: (,161 163,162 163,163 163,164 164)}
```

We will see that the RowSets section (starting at Data:) is the most interesting. RowSets allow selection of the view contents, but they also support insertion, update and deletion. Thus the important information in these two extracts was the View definition 155 (from above) and the Result RowSet object 164:

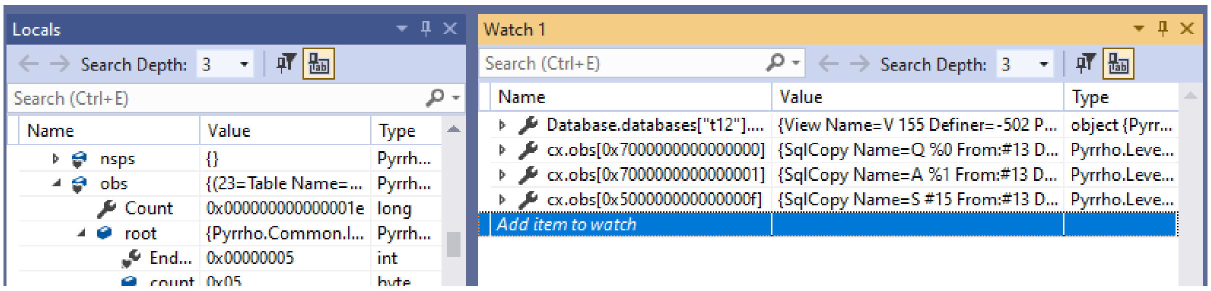
```
{View Name=V 155 Definer=-502 Ppos=155 Query select q,r as s,a from p Ppos: 155 Cols
        (A:160[23],Q:158[23],R:159[23],S:159[23]) Domain TABLE (158,159,160)
        ([158,Domain INTEGER],[159,Domain CHAR],[160,Domain INTEGER])  Targets: 23}
 164 SelectedRowSet 164(158,159,160) key (158,159,160) targets: 23=64 Source: 64
        SQMap: (158=164[43],159=164[93],160=164[115]))
```

Insert into a View

Set another breakpoint at the end of the Instance() method (line 186) and continue to there, to see what happens in the instance. Scrolll down in the Locals window to r and right-click and copy its to a Notepad:

```
{SelectedRowSet %7(#15,%0,%1) key (%0,#15,%1) targets: 23=%7 Source: 64
        SQMap: (#15=%7[93],%0=%7[43],%1=%7[115])}
```

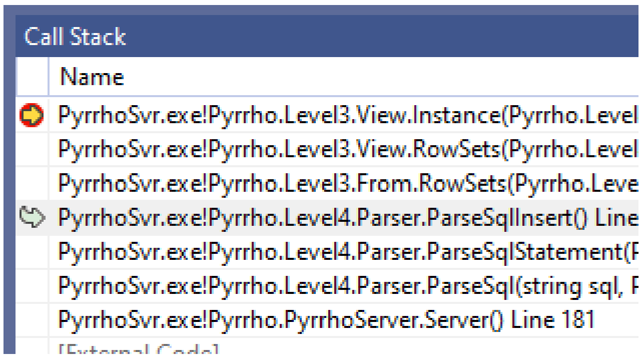
We see that many of the uids have changed to values displayed with # and %. The server uses numbers above 2^{60} as internal uids, in several ranges defined in Transaction.cs. For easy reading these long numbers are notated with prefixes !, #, %, @ for the uncommitted, lexical, heap and prepared ranges respectively (so #15 is 0x500000000000000f). Drilling down in the Locals window into the BTree structure of cx.obs, or using Watch entries such as cx.obs[0x7000000000000000], we can see the following objects:



Name	Value	Type
nsps	{}	Pyrrh...
obs	{(23=Table Name=...	Pyrrh...
Count	0x000000000000001e	long
root	{Pyrrho.Common.I...	Pyrrh...
End...	0x00000005	int
count	0x05	byte

Name	Value	Type
Database.databases["t12"]....	{View Name=V 155 Definer=-502 P...	object {Pyrr...
cx.obs[0x7000000000000000]	{SqlCopy Name=Q %0 From:#13 D...	Pyrrho.Leve...
cx.obs[0x7000000000000001]	{SqlCopy Name=A %1 From:#13 D...	Pyrrho.Leve...
cx.obs[0x500000000000000f]	{SqlCopy Name=S #15 From:#13 D...	Pyrrho.Leve...

```
{SqlCopy Name=S #15 From:#13 Domain CHAR copy from 159}
{SqlCopy Name=Q %0 From:#13 Domain INTEGER copy from 158}
{SqlCopy Name=A %1 From:#13 Domain INTEGER copy from 160}
```



```
PyrrhoSvr.exe!Pyrrho.Level3.View.Instance(Pyrrho.Level
PyrrhoSvr.exe!Pyrrho.Level3.View.RowSets(Pyrrho.Level
PyrrhoSvr.exe!Pyrrho.Level3.From.RowSets(Pyrrho.Leve
PyrrhoSvr.exe!Pyrrho.Level4.Parser.ParseSqlInsert() Line
PyrrhoSvr.exe!Pyrrho.Level4.Parser.ParseSqlStatement(P
PyrrhoSvr.exe!Pyrrho.Level4.Parser.ParseSql(string sql, P
PyrrhoSvr.exe!Pyrrho.PyrrhoServer.Server() Line 181
[External Code]
```

The cx.data tree has the SelectedRowSet shown above.

Notice that the SqlCopy objects take their values from the View columns, and the SelectedRowSets contain the information (in SQMap) about the corresponding TableColumns.

Remember, though, that the command is INSERT, so let us step the debugger (Step Over) till we reach the Transaction.Execute() method in ParseSqlInsert(), called at line 6607 in Parser.cs. The set of RowSets has just been reviewed, and this.cx.data now contains:

```
{(64=IndexRowSet 64(43,93,115) Keys: (43,93,115),
  163=TableExpRowSet 163(158,159,160) key (158,159,160) targets: 23=164 Source: 164,
  164=SelectedRowSet 164(158,159,160) key (158,159,160) targets: 23=164 Source: 64
        SQMap: (158=164[43],159=164[93],160=164[115]),
  #13=SelectedRowSet %7(#15,%0,%1) key (%0,#15,%1) targets: 23=%7 Source: 64
        SQMap: (#15=%7[93],%0=%7[43],%1=%7[115]),
  #18=SqlRowSet #18(#15),
  %6=TableExpRowSet %6(%0,#15,%1) key (%0,#15,%1) targets: 23=%7 Source: %7,
  %7=SelectedRowSet %7(#15,%0,%1) key (%0,#15,%1) targets: 23=%7 Source: #18
        SQMap: (#15=%7[93],%0=%7[43],%1=%7[115]))}
```

In the illustration above, the greyed lines are things we have seen before, or no longer longer referenced following the review, We see that the instanced version %7 of the View result 164 has been connected to the lexical position of the INSERT at character 13 of the command, and contains enough information to direct the data to the base table 23. #18 is just the list of rows from the VALUES part of the command, and these can be seen in the value for this.cx.obs:

```
#18=#24,#35,
#24=SqlRow #24 Domain ROW (#25)([#25,CHAR])  [#25],
#25=Twenty,
#35=SqlRow #35 Domain ROW (#36)([#36,CHAR])  [#36],
#36=Thirty,
```

Now tr.Execute(s,cx) is about to be called. Wec see that s is the SqlInsert statement

```
#1=SqlInsert #1 Nuid=#13 Target: 155 Value: %7,
#13=From Name=V #13 RowType:(#15|%0,%1) Target=%2
```

(The #13 here refers both to a From object in cx.obs and the the corresponding rowset #13 which is now redirected to %7[2].)

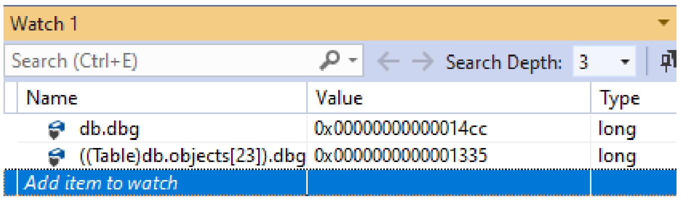
Use the Step Into icon in the debugger, and then follow (using Step Over) how tr.Execute(#1) leads to #1.Obey() (Step Into), %7.Insert() (Step Into), and to 23.Insert(cx,%7), at line 403 of Table.cs.

Insert on a base table (like Update and Delete) may involve triggers, and SQL trigger machinery is quite complex. Pyrrho handles such operations in a TableActivation with a TransitionRowSet. Step over the return statement back into %7.Insert, and look at the TableActivation ta. Its _trs field is a TransitionRowSet:

```
{TransitionRowSet %10(#15,%0,%1) targets: 23=64 From: 23 Data: #18}
```

There are no triggers in this example. Traversal of the Data rowset #18 starts. At line 1238 of RowSet.cs, step into ta.EachRow(). In Activation.cs at line 361 we see we have a TargetCursor `{(43=1,93=Twenty,115= Null) %10}` that has filled in a suitable value for the primary key column 43. There is also and a proposed TableRow rc. In the next few lines we generate a Physical record r (stop at line 387 of Activation.cs)

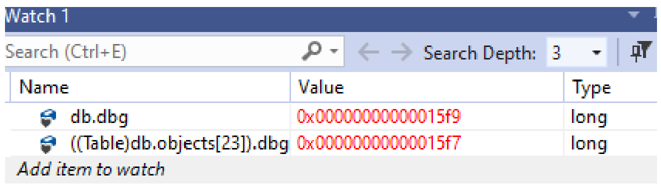
```
{Record !0[23]: 43=1,93=Twenty,115=}
```


Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
db.dbg	0x00000000000014cc	long
((Table)db.objects[23]).dbg	0x0000000000001335	long
Add item to watch		

Let us take a little time to see what happens at line 387 in cx.Add(r). Use the debugger to note the value[3] of db.dbg is 0x14cc, and ((Table)db.objects[23]).dbg is 0x1335.


Watch 1

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
db.dbg	0x00000000000015f9	long
((Table)db.objects[23]).dbg	0x00000000000015f7	long
Add item to watch		

When we add physical Record r is added to the TableActivation, it is installed the Transaction object. Adding a new record to the table makes a new Table object, and therefore a new Transaction object, so when we step over line 387 the Watch window changes to showthe changed objects. Both before and after the change, nodes the nodes of the structure such as Table than the root of the structure. These aspects are typical of shareable structures.

[2] The "unique" identifier refers to two different things! At least the From is in cx.obs and the rowset is in cx.dataAs a reminder, this property is called Nuid.

[3] Your mileage may vary (as they say in car maintenance books), depending on any extra steps you have made. Your numbers should match each other where mine do, and be in the same order.

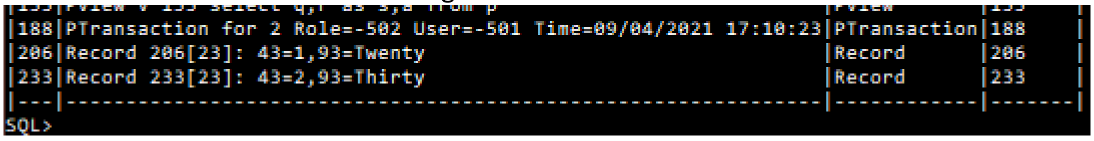
Anyway, at this point the new row has been added to the table in the transaction. At line 393 the Context's copy of the transaction is updated (in a single 64-bit assignment).

We are not done though, as there is a second row in the TransitionRowSet traversal, which gives

```
{Record !1[23]: 43=2,93=Thirty,115=}
```

Once this is also added, the Transaction will commit, and as discussed before this will lock the database file while the physical records are added, and the new records will be loaded into the cx.db.parent database. The Database.databases list is then (atomically) replaced, and the next command will start with the new version of the Database.

The new entries in the transaction log are:



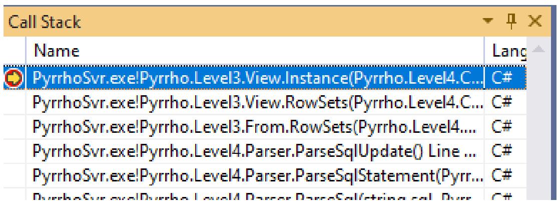
```
|188|PTransaction for 2 Role=-502 User=-501 Time=09/04/2021 17:10:23|PTransaction|188    |
|206|Record 206[23]: 43=1,93=Twenty                                   |Record      |206    |
|233|Record 233[23]: 43=2,93=Thirty                                   |Record      |233    |
|---|------------------------------------------------------------------|------------|-------|
SQL>
```

The important aspect in the above is that the changes are made to the Table, not the View. The View contains no data.

Updating a View

Ensure the breakpoint at the end of View.Instance() is still in place. The next step in the test is

`update v set s='Forty two' where q=1`



```
Call Stack
Name                                                              Lang
PyrrhoSvr.exe!Pyrrho.Level3.View.Instance(Pyrrho.Level4.C...     C#
PyrrhoSvr.exe!Pyrrho.Level3.View.RowSets(Pyrrho.Level4.C...      C#
PyrrhoSvr.exe!Pyrrho.Level3.From.RowSets(Pyrrho.Level4....       C#
PyrrhoSvr.exe!Pyrrho.Level4.Parser.ParseSqlUpdate() Line ...     C#
PyrrhoSvr.exe!Pyrrho.Level4.Parser.ParseSqlStatement(Pyrr...     C#
```

This time the VirtualRowSet is

```
{VirtualRowSet #8(%0,%1,%2) where (#35) matches (%0=1)
        targets: 155=#8
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True)}
```

and the Result rowset r is

```
{SelectedRowSet %8(%0,%1,%2) key (%0,%1,%2) where (#35) matches (%0=1) targets: 23=%8 Source: 64
        SQMap: (%0=%8[43],%1=%8[93],%2=%8[115])}
```

Single-stepping back to SqlUpdate, after Review of the rowsets, at line 6708 of Parser.cs in ParseSqlUpdate() we have the following in this.cx.data:

```
{(64=IndexRowSet 64(43,93,115) Keys: (43,93,115),
  163=TableExpRowSet 163(158,159,160) key (158,159,160) targets: 23=164 Source: 164,
  164=SelectedRowSet 164(158,159,160) key (158,159,160) targets: 23=164 Source: 64
        SQMap: (158=164[43],159=164[93],160=164[115]),
  #8=SelectedRowSet %8(%0,%1,%2) key (%0,%1,%2) where (#35) matches (%0=1) targets: 23=%8
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) Source: 64
        SQMap: (%0=%8[43],%1=%8[93],%2=%8[115]),
  %7=TableExpRowSet %7(%0,%1,%2) key (%0,%1,%2) targets: 23=%8 Source: %8,
  %8=SelectedRowSet %8(%0,%1,%2) key (%0,%1,%2) where (#35) matches (%0=1) targets: 23=%8
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) Source: 64
        SQMap: (%0=%8[43],%1=%8[93],%2=%8[115]))}
```

Note that entries 64-164 are (of course) the same as before. The review process has added the where condition, a filter %0=1 and the update assignments (the information for these was alread available during Instance but not used at that point).

Once again the update operation is made to take place on the table instead of the view. tr.Execute(#1) leads to #1.Obey(), and %8.Update(), which leads to 23.Update(). Stepping back to the SelectedRowSet's %8.Update(), at line 1244 we again have a TableActivation, 44 this time, whose TransitionRowSet %11 is

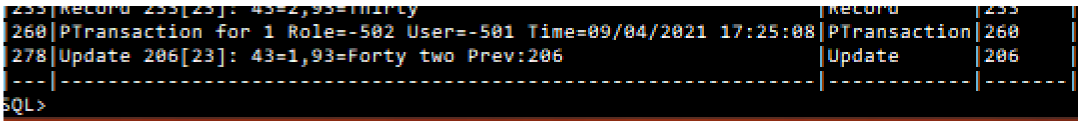
```
{TransitionRowSet %11(%0,%1,%2) where (#35) targets: 23=%8 From: 23 Data: %8}
```

At line 1246, we start traversal, and stepping into ta.EachRow() we see at line 362 of Avctivation.cs that we have a TargetCursor `{(43=1,93=Twenty,115= Null) %11}`. Execution skips down to line 400, where by line 407 the updated values vs are computed as `{(43=1,93=Forty two,115= Null)}` At this point there is some complicated code caused by the SQL stanadrd's requirements for trigger operation, but by line 450 we have constructed a physical record u

```
{(!0=Update 206[23]: 43=1,93=Forty two,115= Prev:206)}
```

which is then added to the TableActivation.

Transaction Commit() leads to new committed entries, and we can check these in the transaction log:



```
|233|Record 233[23]: 43=2,93=Thirty                                |Record      |233    |
|260|PTransaction for 1 Role=-502 User=-501 Time=09/04/2021 17:25:08|PTransaction|260    |
|278|Update 206[23]: 43=1,93=Forty two Prev:206                     |Update      |206    |
|---|-------------------------------------------------------------|------------|-------|
SQL>
```

Deletion from a View

`delete from v where s='Thirty'`

This time we get

```
{VirtualRowSet #13(%0,%1,%2) where (#22) matches (%1=Thirty) targets: 155=#13}
```

and Instance() result is

```
{SelectedRowSet %9(%0,%1,%2) key (%0,%1,%2) where (#22) matches (%1=Thirty) targets: 23=%9 Source: 64
        SQMap: (%0=%9[43],%1=%9[93],%2=%9[115])}
```

The review of rowsets proceeds similarly to the above in ParseSqlDelete, at line 6656 of Parser.cs, cx.data contains:

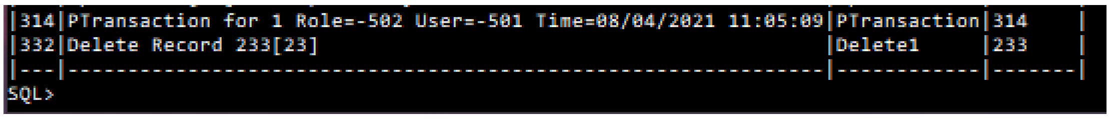
```
{(64=IndexRowSet 64(43,93,115) targets: 23=164 Keys: (43,93,115),
  163=TableExpRowSet 163(158,159,160) key (158,159,160) targets: 23=164 Source: 164,
  164=SelectedRowSet 164(158,159,160) key (158,159,160) targets: 23=164 Source: 64
        SQMap: (158=164[43],159=164[93],160=164[115]),
  #13=SelectedRowSet %9(%0,%1,%2) key (%0,%1,%2) where (#22) matches (%1=Thirty) targets: 23=%9
        Source: 64 SQMap: (%0=%9[43],%1=%9[93],%2=%9[115]),
  %8=TableExpRowSet %8(%0,%1,%2) key (%0,%1,%2) targets: 23=%9 Source: %9,
  %9=SelectedRowSet %9(%0,%1,%2) key (%0,%1,%2) matches (%1=Thirty) targets: 23=%9 Source: 64
        SQMap: (%0=%9[43],%1=%9[93],%2=%9[115]))}
```

Tracing the Execution sequence as before, tr.Execute(#1) leads to #1.Obey(), leads to %10.Delete() (and 23.Delete(cx,%10)), so that at line 1253 of RowSet.cs, we haveTableActivation ta so that when we step into EachRow we see that tgc is `{(43=2,93=Thirty,115= Null) %13}`.

The TableRow rc is available. Instead tgc.Rec() returns a list of the (single) item to be deleted, and line 478 of Activation.cs creates the physical record

```
  {Delete Record 233[23]}
```

and it is added and committed as before:



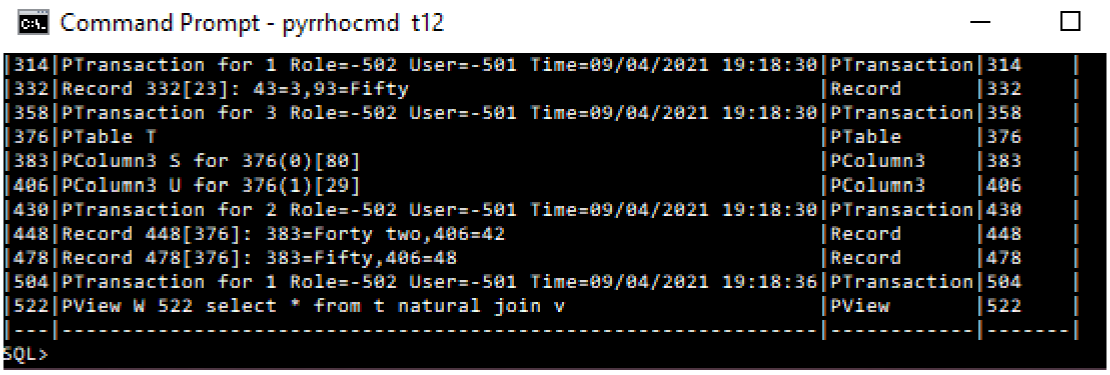
```
|314|PTransaction for 1 Role=-502 User=-501 Time=08/04/2021 11:05:09|PTransaction|314    |
|332|Delete Record 233[23]                                          |Delete1     |233    |
|---|-------------------------------------------------------------|------------|-------|
SQL>
```

A View of a Join

At this point it is clear that updating views of joins is within reach. (There isn't a need to have updates to ordinary joins, but they could work too.) The code for this is in Join.cs as we will see.

We first add some more records and a new view definition:

```
insert into p(r) values('Fifty')
create table t(s char,u int)
insert into t values('Forty two',42),('Fifty',48)
create view w as select * from t natural join v
```


Command Prompt - pyrrhocmd t12

```
|314|PTransaction for 1 Role=-502 User=-501 Time=09/04/2021 19:18:30|PTransaction|314    |
|332|Record 332[23]: 43=3,93=Fifty                                  |Record      |332    |
|358|PTransaction for 3 Role=-502 User=-501 Time=09/04/2021 19:18:30|PTransaction|358    |
|376|PTable T                                                       |PTable      |376    |
|383|PColumn3 S for 376(0)[80]                                      |PColumn3    |383    |
|406|PColumn3 U for 376(1)[29]                                      |PColumn3    |406    |
|430|PTransaction for 2 Role=-502 User=-501 Time=09/04/2021 19:18:30|PTransaction|430    |
|448|Record 448[376]: 383=Forty two,406=42                          |Record      |448    |
|478|Record 478[376]: 383=Fifty,406=48                              |Record      |478    |
|504|PTransaction for 1 Role=-502 User=-501 Time=09/04/2021 19:18:36|PTransaction|504    |
|522|PView W 522 select * from t natural join v                     |PView       |522    |
|---|---------------------------------------------------------------|------------|-------|
SQL>
```

The next step will be

`update w set u=50,a=21 where q=2`

At the start of View.Instance() we see **`this`** (View 522) is

```
{View Name=W 522 Definer=-502 Ppos=522 Query select * from t natural join v Ppos: 522
        Cols (A:@4[@7],Q:@2[@7],S:@0[376]@3[@7],U:@1[376])
        Domain ROW (@0,@1,@2,@4,@3) Display=5([@0,Domain CHAR],[@1,Domain INTEGER],
                [@2,Domain INTEGER],[@3,Domain CHAR],[@4,Domain INTEGER])  Targets: 23,376}
```

Note the ambiguous reference S is resolved to table T and view V in the Cols property (@7 is an instance of view V, as we will see below).

Here is the Framing for W (View 522) but omitting the entries we had above:

```
{Framing (..
 376 Table Name=T 376 Definer=-502 Ppos=376
        Domain TABLE (383,406)([383,Domain CHAR],[406,Domain INTEGER])  KeyCols: ,
 383 TableColumn 383 Definer=-502 Ppos=383 Domain CHAR Table=376,
 406 TableColumn 406 Definer=-502 Ppos=406 Domain INTEGER Table=376,
 524 CursorSpecification 524 RowType:(@0,@1,@2,@4,@3) Where: Source={select * from t natural join v}
        Union: 525,
 525 QueryExpression 525 RowType:(@0,@1,@2,@4,@3) Where: Left: 526 ,
 526 QuerySpecification 526 RowType:(@0,@1,@2,@4,@3) Where: TableExp 528,
 527 SqlStar Name=* 527 CONTENT From:526 CONTENT,
 528 TableExpression 528 Nuid=529 RowType:(@0,@1,@2,@4|@3) Where: Target: 529,
 529 JoinPart 529 RowType:(@0,@1,@2,@4|@3) Where:530 NATURAL INNER join531 on @5
        matching @0=@3 @3=@0,
 530 From Name=T 530 RowType:(@0,@1) OrdSpec (0=@0) Where: Target=376,
 531 From Name=V 531 RowType:(@2,@3,@4) OrdSpec (0=@3) Where: Target=@7,
 @0 SqlCopy Name=S @0 From:530 Domain CHAR copy from 383,
 @1 SqlCopy Name=U @1 From:530 Domain INTEGER copy from 406,
 @2 SqlCopy Name=Q @2 From:531 Domain INTEGER copy from 164,
 @3 SqlCopy Name=S @3 From:531 Domain CHAR copy from 166,
 @4 SqlCopy Name=A @4 From:531 Domain INTEGER copy from 173,
 @5 SqlValueExpr Name= @5 From:_ Left:@0 BOOLEAN Right:@3 @5(@0=@3),
 @7 View Name=V @7 Definer=-502 Ppos=155 Query select q,r as s,a from p Ppos: 155
        Cols (A:@4[23],Q:@2[23],R:@3[23],S:@3[23]) Domain TABLE (@2,@3,@4) Display=3
        ([@2,Domain INTEGER],[@3,Domain CHAR],[@4,Domain INTEGER])  Targets: 23,
 @8 CursorSpecification @8 RowType:(@2,@3,@4) Where: Source={select q,r as s,a from p} Union: @10,
 @9 SelectStatement @9 CS=@8,
 @10 QueryExpression @10 RowType:(@2,@3,@4) Where: Left: @11 ,
 @11 QuerySpecification @11 RowType:(@2,@3,@4) Where: TableExp @12,
 @12 TableExpression @12 Nuid=@13 RowType:(@2,@3,@4) Where: Target: @13,
 @13 From Name=P @13 RowType:(@2,@3,@4) Where: Target=23,
 @14 SqlCopy Name=R @14 From:@13 Alias=S Domain CHAR copy from 93,
 @15 SqlCopy Name=A @15 From:@13 Domain INTEGER copy from 115)
Data: (..
 376 TableRowSet 376(383,406) targets: 376=376,
 528 TableExpRowSet 528(@0,@1,@2,@4|@3) key (@0,@1,@2,@4) targets: 23=@13,376=530 Source: 529,
 529 JoinRowSet 529(@0,@1,@2,@4|@3) targets: 23=@13,376=530 JoinCond: (@5) matching @0=@3 @3=@0
        First: @6 Second: @16,
 530 SelectedRowSet 530(@0,@1) targets: 376=530 Source: 376
        SQMap: (@0=530[383],@1=530[406]),
 531 SelectedRowSet @13(@2,@3,@4) key (@2,@3,@4) targets: 23=@13 Source: 64
        SQMap: (@2=@13[43],@3=@13[93],@4=@13[115]),
 @6 OrderedRowSet @6(@0,@1) key (@0) order (@0) targets: 376=530 Source: 530,
 @12 TableExpRowSet @12(@2,@3,@4) key (@2,@3,@4) targets: 23=@13 Source: @13,
 @13 SelectedRowSet @13(@2,@3,@4) key (@2,@3,@4) targets: 23=@13 Source: 64
        SQMap: (@2=@13[43],@3=@13[93],@4=@13[115]),
 @16 OrderedRowSet @16(@2,@3,@4) key (@3) order (@3) targets: 23=@13 Source: @13)
Result 529 Results: (,158 175,163 175,175 175,180 180,525 528,526 528,528 528,529 529)}
```

There are quite a few complicating factors here. First, note that the reference to view V in object 524 resulted in a precompiled instancing of V 155 to the new uid[4] @7. Second, the join is natural, so that the column @3 that will not be shown in the results is placed at the end of the rowType for 528 and marked hidden. Third, the join process requires the two operands of the join to be ordered (see rowSets @6 and @16). Fourth, note that in 529 we are tracking equivalent expressions @0 and @3. But all of these aspects are standard implementation details.

Updating a Viewed Join

The parameter for instancing is the VirtualRowSet

```
{VirtualRowSet #8(%0,%1,%2,%3,%4) where (#31) matches (%2=2) targets: 522=#8
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True,UpdateAssignment Vbl: %3 Val: #21=True)}
```

and Instancing gives the following objects and rowsets (again omitting items that are the same as in the above steps) in cx.obs and cx.data:

```
{(#1=UpdateSearch #1 Nuid=#8 Target: 522,
  #8=From Name=W #8 RowType:(%0,%1,%2,%3,%4)
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True,UpdateAssignment Vbl: %3 Val: #21=True)
        Filter:(%2=2) Where:(#31=True) Target=%5,
  #16=50,
  #21=21,
  #31=SqlValueExpr Name= #31 From:#8 Left:%2 BOOLEAN Right:#32 #31(%2=#32),
  #32=2,
  %0=SqlCopy Name=S %0 From:#8 Domain CHAR copy from @0,
  %1=SqlCopy Name=U %1 From:#8 Domain INTEGER copy from @1,
  %2=SqlCopy Name=Q %2 From:#8 Domain INTEGER copy from @2,
  %3=SqlCopy Name=A %3 From:#8 Domain INTEGER copy from @4,
  %4=SqlCopy Name=S %4 From:#8 Domain CHAR copy from @3,
  %5=View Name=W %5 Definer=-502 Ppos=522 Query select * from t natural join v Ppos: 522
        Cols (A:%3[%18],Q:%2[%18],S:%0[376]%4[%18],U:%1[376])
        Domain ROW (%0,%1,%2,%3,%4) Display=5([%0,Domain CHAR],[%1,Domain INTEGER],
                [%2,Domain INTEGER],[%3,Domain INTEGER],[%4,Domain CHAR])  Targets: 23,376,
  %6=CursorSpecification %6 RowType:(%0,%1,%2,%3,%4) Where: Source={select * from t natural join v}
        Union: %7,
  %7=QueryExpression %7 RowType:(%0,%1,%2,%3,%4) Where: Left: %8 ,
  %8=QuerySpecification %8 RowType:(%0,%1,%2,%3,%4) Where: TableExp %10,
  %9=SqlStar Name=* %9 CONTENT From:%8 CONTENT,
  %10=TableExpression %10 Nuid=%11 RowType:(%0,%1,%2,%3|%4) Where: Target: %11,
  %11=JoinPart %11 RowType:(%0,%1,%2,%3|%4) Where:%12 NATURAL INNER join%16 on %15
        matching %0=%4 %4=%0,
  %12=From Name=T %12 RowType:(%0,%1) OrdSpec (0=%0) Where: Target=376,
  %13=SqlCopy Name=U %13 From:%12 Domain INTEGER copy from 406,
  %15=SqlValueExpr Name= %15 From:_ Left:%0 BOOLEAN Right:%4 %15(%0=%4),
  %16=From Name=V %16 RowType:(%2,%4,%3) OrdSpec (0=%4) Where: Target=%18,
  %17=SqlCopy Name=S %17 From:%16 Domain CHAR copy from 166,
  %18=View Name=V %18 Definer=-502 Ppos=155 Query select q,r as s,a from p Ppos: 155
        Cols (A:%3[23],Q:%2[23],R:%4[23],S:%4[23]) Domain TABLE (%2,%4,%3) Display=3
        ([%2,Domain INTEGER],[%3,Domain INTEGER],[%4,Domain CHAR])  Targets: 23,
  %19=CursorSpecification %19 RowType:(%2,%4,%3) Where: Source={select q,r as s,a from p} Union: %21,
  %20=SelectStatement %20 CS=%19,
  %21=QueryExpression %21 RowType:(%2,%4,%3) Where: Left: %22 ,
```

[4] Uids such as @7 are used for precompiled things: this is in the start of the prepared range, which is used for this purpose. Such uids persist in the database, unlike the lexical #, heap %, and transaction ! uids.

```
  %22=QuerySpecification %22 RowType:(%2,%4,%3) Where: TableExp %23,
  %23=TableExpression %23 Nuid=%24 RowType:(%2,%4,%3) Where: Target: %24,
  %24=From Name=P %24 RowType:(%2,%4,%3) Where: Target=23,
  %25=SqlCopy Name=R %25 From:%24 Alias=S Domain CHAR copy from 93,
  %26=SqlCopy Name=A %26 From:%24 Domain INTEGER copy from 115,
  %28=SqlCopy Name=A %28 From:%16 Domain INTEGER copy from 173)}
```

After Review the rowsets in this.cx.data are (line 6708 of Parser.cs), again omitting rows we have had above:

```
{(376=TableRowSet 376(383,406) targets: 376=376,
  #8=JoinRowSet %11(%0,%1,%2,%3|%4) key (%0,%1,%2,%3,%4) where (#31) matches (%2=2)
        targets: 23=%24,376=%12
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True,UpdateAssignment Vbl: %3 Val: #21=True)
        JoinCond: (%15) matching %0=%4 %4=%0 First: %14 Second: %27,
  %10=TableExpRowSet %10(%0,%1,%2,%3|%4) key (%0,%1,%2,%3) targets: 23=%24,376=%12 Source: %11,
  %11=JoinRowSet %11(%0,%1,%2,%3|%4) key (%0,%1,%2,%3,%4) where (#31) matches (%2=2)
        targets: 23=%24,376=%12
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True,UpdateAssignment Vbl: %3 Val: #21=True)
        JoinCond: (%15) matching %0=%4 %4=%0 First: %14 Second: %27,
  %12=SelectedRowSet %12(%0,%1) key (%0,%1) targets: 376=%12 Source: 376
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) SQMap: (%0=%12[383],%1=%12[406]),
  %14=OrderedRowSet %14(%0,%1) key (%0) order (%0) targets: 376=%12 Source: %12
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True),
  %23=TableExpRowSet %23(%2,%4,%3) key (%2,%4,%3) targets: 23=%24 Source: %24,
  %24=SelectedRowSet %24(%2,%4,%3) key (%2,%4,%3) matches (%2=2) targets: 23=%24 Source: 64
        Assigs:(UpdateAssignment Vbl: %3 Val: #21=True) SQMap: (%2=%24[43],%3=%24[115],%4=%24[93]),
  %27=OrderedRowSet %27(%2,%4,%3) key (%4) order (%4) matches (%2=2) targets: 23=%24 Source: %24
        Assigs:(UpdateAssignment Vbl: %3 Val: #21=True))}
```

As before, tr.Execute(#1) leads to #1.Obey(), then to %11.Update().

We see that JoinRowSet has an override for Update (also Insert and Delete). This method traverses the list of targets, with a TableActivation[5] for each base table target, and, for base tables calls any triggers at statement and row level for each. From the listing above, we can see that the targets of the JoinRowSet are SelectedRowSets %24 and %12, so the machinery we had for the single table case works here too.

At each point we synchronise the transitioncursor and the database state of the tableactivation so that we pick up all physical records created for the transaction, e.g. (Join.cs, line 388)

```
for (var b = rsTargets.First(); b != null; b = b.Next())
{
   var ta = ts[b.key()];
   ta.db = db;
   ((TransitionRowSet)ta._trs).At(ta, ((JoinBookmark)teb)._ts[b.value()]._defpos);
   ta.EachRow();
   db = ta.db;
}
```

An interesting point is that the RTree used to sort the rows for the orderedrowsets keeps track of the cursors used to create the set of rows. The field rows is not just a list of TRow but a list of cursor lists (RTree.cs, line 32):

```
        /// <summary>
        /// rows is a set of snapshots of cx.cursors, taken during Build.
        /// We do this so that we can implement updatable joins:
        /// it allows us to accommodate sorted operands of the join.
        /// In previous versions it was just the sorted BList of the rows,
        /// to get the current row, simply subscript by defpos.
        /// </summary>
        internal readonly BList<BTree<long,Cursor>> rows; // RowSet
```

These snapshots are made visible in the RTreeBookmark (RTree.cs, line 106)

[5] Targets of joins can also be RestViews, so that the more general class TargetActivation is named in the code.

```
    internal class RTreeBookmark : Cursor
    {
        internal readonly RTree _rt;
        internal readonly MTreeBookmark _mb;
        internal readonly BTree<long, Cursor> _cs; // for updatable joins
        internal readonly TRow _key;
```

The OrderedCursor carefully places these cursors back in the Context during traversal of the ordered rowset (RowSet.cs, line 2767):

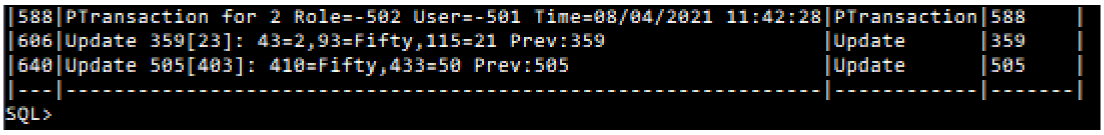
```
        internal OrderedCursor(Context cx,OrderedRowSet ors,RTreeBookmark rb)
            :base(cx,ors,rb._pos,rb._defpos,rb._ppos,rb)
        {
            cx.cursors += rb._cs; // for updatabale joins
            _ors = ors; _rb = rb;
        }
```

It is because these structures are shareable that this sort of manipulation is both safe and cost-free. So in JoinRowSet.Update we have no difficulty constructing the targetcursors and associated new records:

```
{Update 505[403]: 410=Fifty,433=50 Prev:505}
```

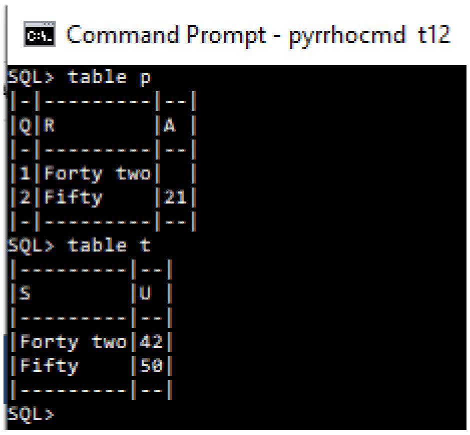
```
{Update 359[23]: 43=2,93=Fifty,115=21 Prev:359}
```

and after Transaction.Commit we have the new transaction log entries:



```
|588|PTransaction for 2 Role=-502 User=-501 Time=08/04/2021 11:42:28|PTransaction|588    |
|606|Update 359[23]: 43=2,93=Fifty,115=21 Prev:359                    |Update      |359    |
|640|Update 505[403]: 410=Fifty,433=50 Prev:505                       |Update      |505    |
|---|-----------------------------------------------------------------|------------|-------|
SQL>
```

All the above actions used commands acting on the views V and W. But, as we have seen, all of the changes were to the base tables. Their final state is



Command Prompt - pyrrhocmd t12

```
SQL> table p
|-|---------|--|
|Q|R        |A |
|-|---------|--|
|1|Forty two|  |
|2|Fifty    |21|
|-|---------|--|
SQL> table t
|---------|--|
|S        |U |
|---------|--|
|Forty two|42|
|Fifty    |50|
|---------|--|
SQL>
```