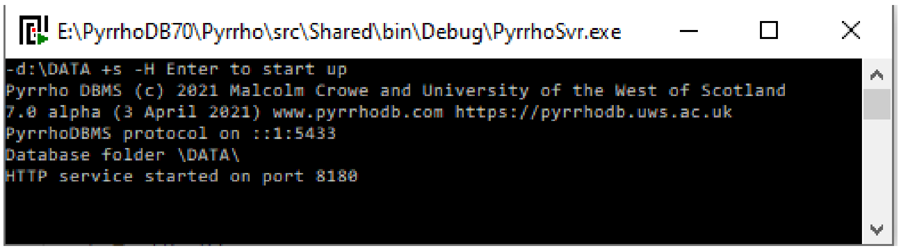
# All the details for Test 12

Malcolm Crowe, 4 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 (3 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 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).

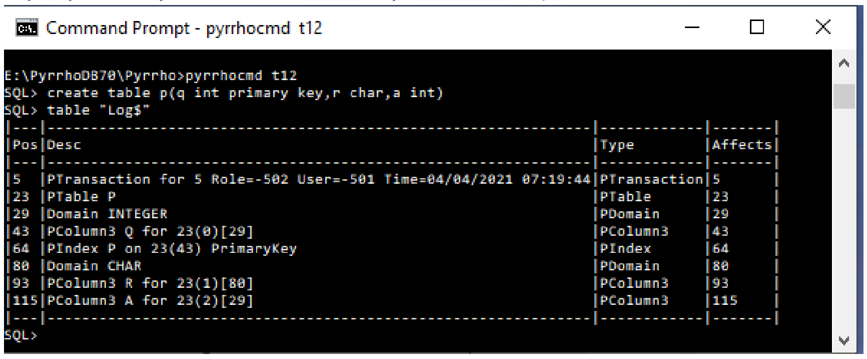
We start with no database file.

## 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 exckusive 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.



Command Prompt - pyrrhocmd t12

```
E:\PyrrhoDB70\Pyrrho>pyrrhocmd t12
SQL> create table p(q int primary key,r char,a int)
SQL> table "Log$"
|---|----------------------------------------------------------------|------------|-------|
|Pos|Desc                                                            |Type        |Affects|
|---|----------------------------------------------------------------|------------|-------|
|5  |PTransaction for 5 Role=-502 User=-501 Time=04/04/2021 07:19:44|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>
```

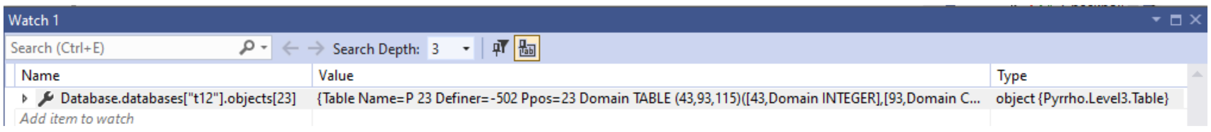
```
create table p(q primary
key,r char,a int)
table "Log$"
```

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.

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.

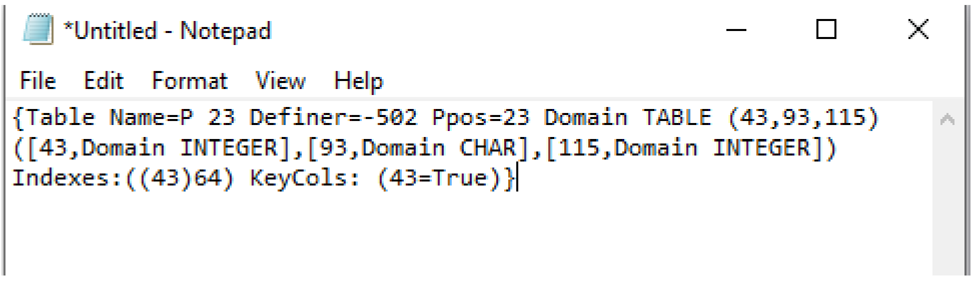
The Watch window allows easy inspection of things, by default displaying the ToString() version and allowing expansion, CopyValue, etc. Pause the server, and in the Watch 1 window, enter

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} |
| Add item to watch | | |

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



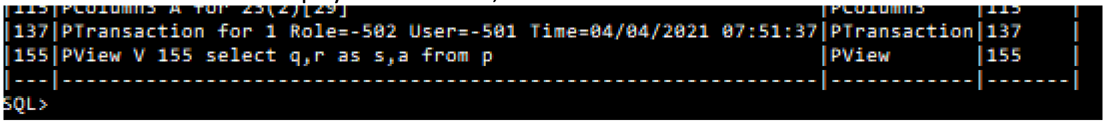
```
{Table Name=P 23 Definer=-502 Ppos=23 Domain TABLE (43,93,115)
([43,Domain INTEGER],[93,Domain CHAR],[115,Domain INTEGER])
Indexes:((43)64) KeyCols: (43=True)}
```

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.

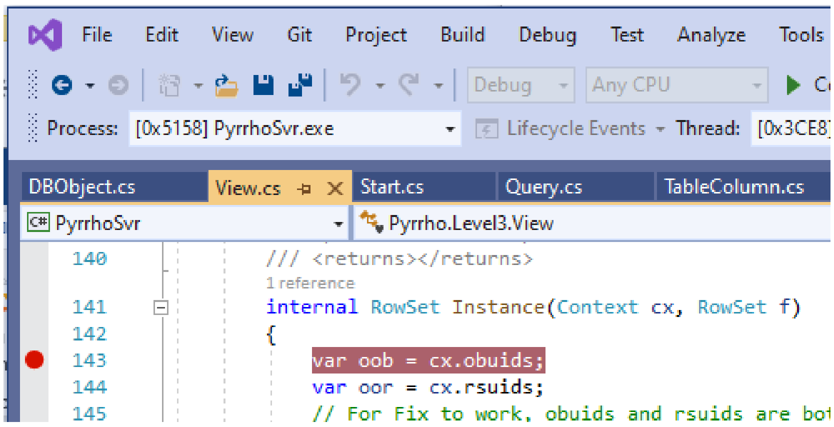


```
|115|PColumns A for 23(2)[29]                                           |PColumns    |115    |
|137|PTransaction for 1 Role=-502 User=-501 Time=04/04/2021 07:51:37|PTransaction|137    |
|155|PView V 155 select q,r as s,a from p                              |PView       |155    |
|---|------------------------------------------------------------------|------------|-------|
SQL>
```

As we can see, the PView simple 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.

The Framing object also includes an analysis of the dependencies between the numerous objects in the Framing. This is because when a view is referenced, we would like to take advantage of ambient information to add filters, review rowsets etc, and for this we need to create instances of the view.

Let is 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.

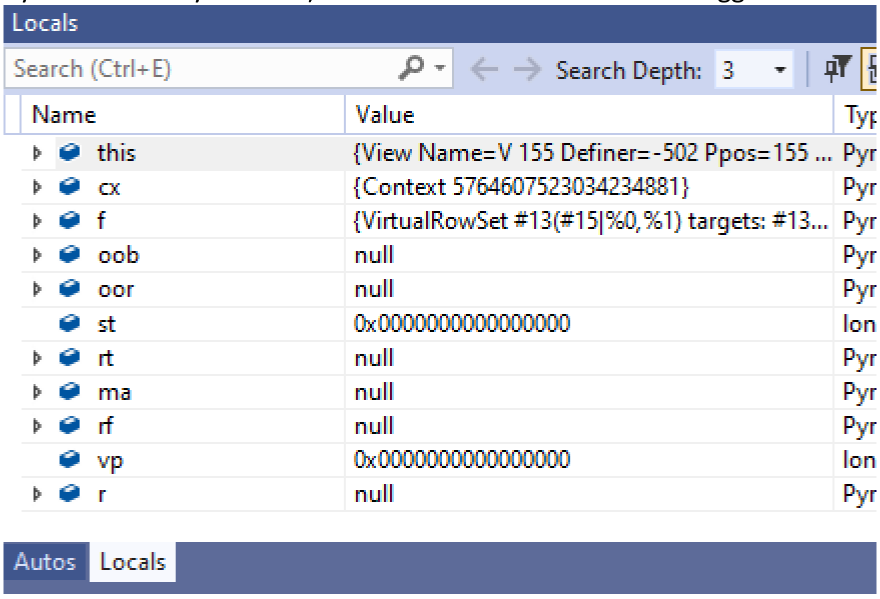


```
140         /// <returns></returns>
            1 reference
141         internal RowSet Instance(Context cx, RowSet f)
142         {
143             var oob = cx.obuids;
144             var oor = cx.rsuids;
145             // For Fix to work, obuids and rsuids are bot
```

Now in the client window letr us use the view V to *insert* data in the table P. (Of course this is not how views are normally used!)

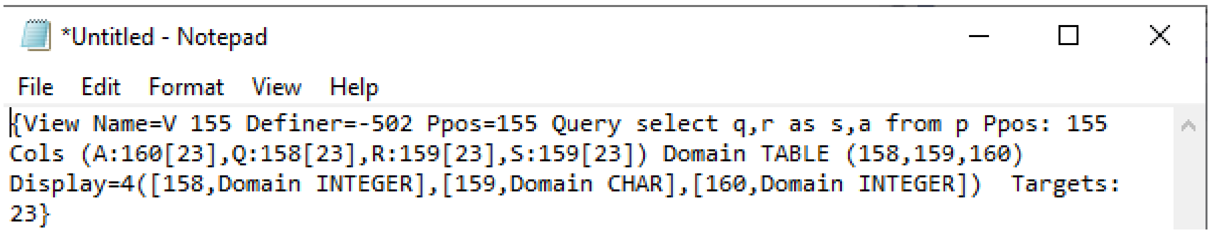
```
insert into v(s) values('Twenty'),('Thirty')
```

(This command does not set a value for the primary key column Q, leaving it to be supplied by Pyrrho's autokey feature.) In the Locals window of the debugger we see



| 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 |

Copy the value from the first line here 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 and copy the framing field into a notepad window. After adding some line breaks, and ignoring the obresfs section, 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) targets: 23=64 Keys: (43,93,115),
 163 TableExpRowSet 163(158,159,160) key (158,159,160) targets: 23=64 Source: 164,
 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]))
ObRefs: ..
Result 164 Results: (,161 163,162 163,163 163,164 164)}
```

We will see that the RowSets section is the most interesting. RowSets allow selection of the view contents, but they also support insertion, update and deletion. Thus the important information here is the View definition 155 and the Result RowSet object 164:

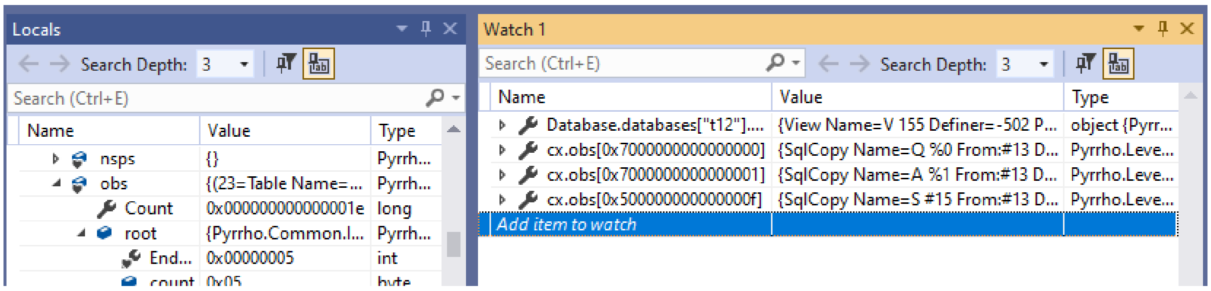
```
{View Name=V 155 Definer=-502 Ppos=155 Query select q,r as s,a from p Ppos: 155 Cols
        (A:160[23],Q:158[23],R:159[23],S:159[23]) Domain TABLE (158,159,160)
        ([158,Domain INTEGER],[159,Domain CHAR],[160,Domain INTEGER])  Targets: 23}
 164 SelectedRowSet 164(158,159,160) key (158,159,160)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

## Insert into a View

Set a breakpoint at the end of the Instance() method (line 186) and continue to there. Scrolll down in the Locals window and copy r to a Notepad:

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

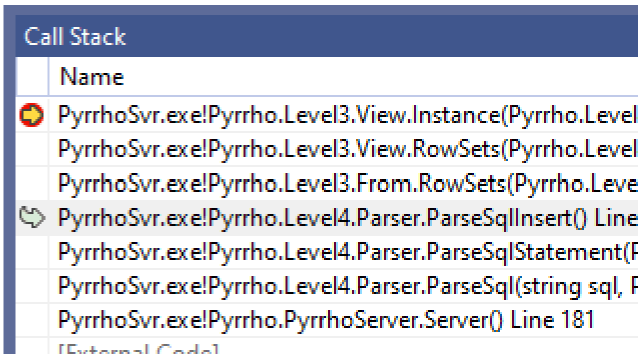
The server uses numbers above $2^{60}$ as internal uids. These are in several ranges defined in Transaction.cs, and for easy reading these long numbers are notated with prefixes !, #, %, @ for the uncommitted, lexical, heap and prepared ranges respectively. Drilling down in the Locals window into the 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... |
| *Add item to watch* | | |

```
{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}
```



```
Name
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(F
PyrrhoSvr.exe!Pyrrho.Level4.Parser.ParseSql(string sql, F
PyrrhoSvr.exe!Pyrrho.PyrrhoServer.Server() Line 181
```

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 till we reach the Transaction.Execute() method, called at line 6607 in Parser.cs. The set of RowSets has just been reviewed, and cx.data now contains:

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

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:

```
  #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 called: 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 correspinding rowset ~13 which is now redirected to %7.)

So tr.Execute(#1) leads to #1.Obey(), %7.Insert(), and to 23.Insert(cx,%7), at line 403 of Table.cs.

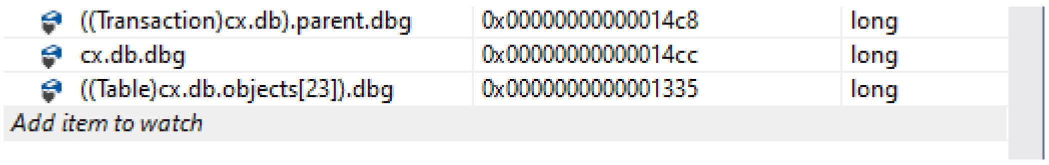
Now Insert on a base table (like Update and Delete) may involved triggers, and SQL trigger machiner is quite complex. Pyrrho handles such operations in a TableActivation with a TransitionRowSet.  At line 423 of Table.cs, we have TableActivation 39 with the TransitionRowSet

```
{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 429, we have the TransitionCursor `{(#15=Twenty,%0= Null,%1= Null) %10}`, containing a TargetCursor `{(43=1,93=Twenty,115= Null) %10}`, with the autokey Q=1 already filled in. At line 441, we call _Insert(cx) for the corresponding TableRow, and this generates the uncommitted Physical record

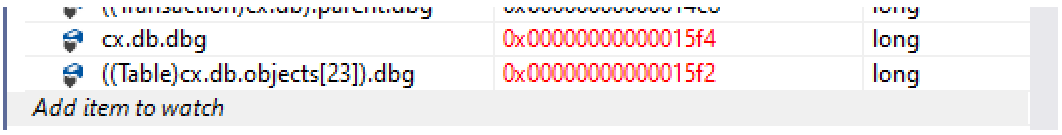
```
{Record !0[23]: 43=1,93=Twenty,115=}
```

Just before the next step please note that cx.db.dbg is 0x14cc, and cx.db.objects[23].dbg is 0x1335.



```
((Transaction)cx.db).parent.dbg    0x00000000000014c8    long
cx.db.dbg                          0x00000000000014cc    long
((Table)cx.db.objects[23]).dbg     0x0000000000001335    long
Add item to watch
```

At line 505 of TableColumn.cs, this is added to the Context (not the TableActivation), which installs it in the Transaction object. Adding the new record to the table made a new Table object, and a new Transaction object:

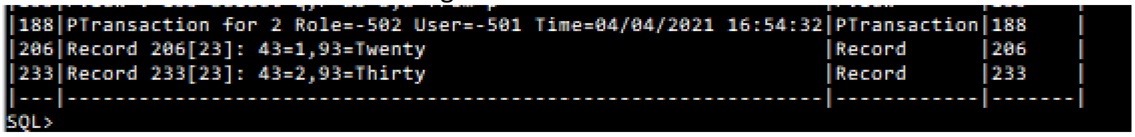


```
cx.db.dbg                          0x00000000000015f4    long
((Table)cx.db.objects[23]).dbg     0x00000000000015f2    long
Add item to watch
```

While of course the parent (Database) is unchanged. Note also that before and after the change, nodes further down the structure are newer than the parent of the structure. Theses aspects are typical of shareable structures.

Anyway, at this point the new row has been added to the table in the transaction. We are not done though, as there is a second row in the TransitionRowSet traversal. 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:



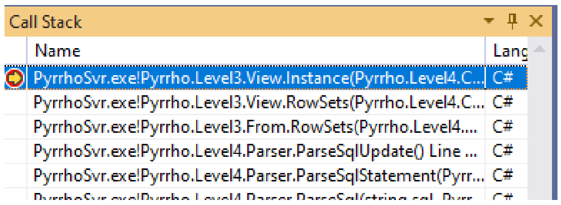
```
|188|PTransaction for 2 Role=-502 User=-501 Time=04/04/2021 16:54:32|PTransaction|188    |
|206|Record 206[23]: 43=1,93=Twenty                                 |Record      |206    |
|233|Record 233[23]: 43=2,93=Thirty                                 |Record      |233    |
|---|---------------------------------------------------------------|------------|-------|
SQL>
```

The important aspect in the above is that no operation was done on the View itself. 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: #8=155
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True)}
```

and the Result rowset is

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

Single-stepping back to SqlUpdate, after Review of the rowsets, we have the following in cx.data:

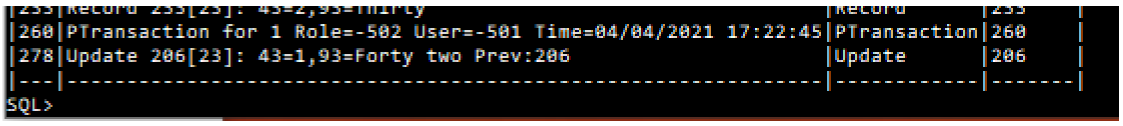
```
{(64=IndexRowSet 64(43,93,115) targets: 23=64 Keys: (43,93,115),
  163=TableExpRowSet 163(158,159,160) key (158,159,160) targets: 23=64 Source: 164,
  164=SelectedRowSet 164(158,159,160) key (158,159,160) targets: 23=64 Source: 64
        SQMap: (158=164[43],159=164[93],160=164[115]),
  #8=SelectedRowSet %8(%0,%1,%2) key (%0,%1,%2) where (#35) matches (%0=1) targets: 23=64
        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=64 Source: %8,
  %8=SelectedRowSet %8(%0,%1,%2) key (%0,%1,%2) where (#35) matches (%0=1) targets: 23=64
        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 filter %0=1 and the update assignments (but in this case these could have been added during Instance).

Once again the update operation is made to take place on the table instead of the view. tr.Execute(%8) leads to %.Update, which leads to 23.Update(). Once again there is a TableActivation, a TransitionRowSet, and a new TableRow, on which _Update() constructs a physical record

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

Transaction Commit leads to new committed entries in the transaction log:



```
|260|PTransaction for 1 Role=-502 User=-501 Time=04/04/2021 17:22:45|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: #13=155}
```

and Instance() result is

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

The review of rowsets proceeds similarly to the above:

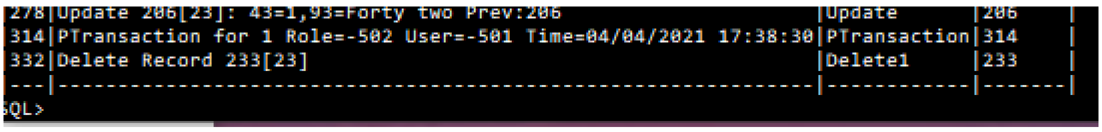
```
{(64=IndexRowSet 64(43,93,115) targets: 23=64 Keys: (43,93,115),
  163=TableExpRowSet 163(158,159,160) key (158,159,160) targets: 23=64 Source: 164,
  164=SelectedRowSet 164(158,159,160) key (158,159,160) targets: 23=64 Source: 64
        SQMap: (158=164[43],159=164[93],160=164[115]),
  #13=SelectedRowSet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

Obviously a new TableRow is not constructed this time. Instead tgc.Rec() returns a list of the single item to be deleted, and rec._Delete() creates the physical record

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

and Transaction.Commit adds it to the transaction log:



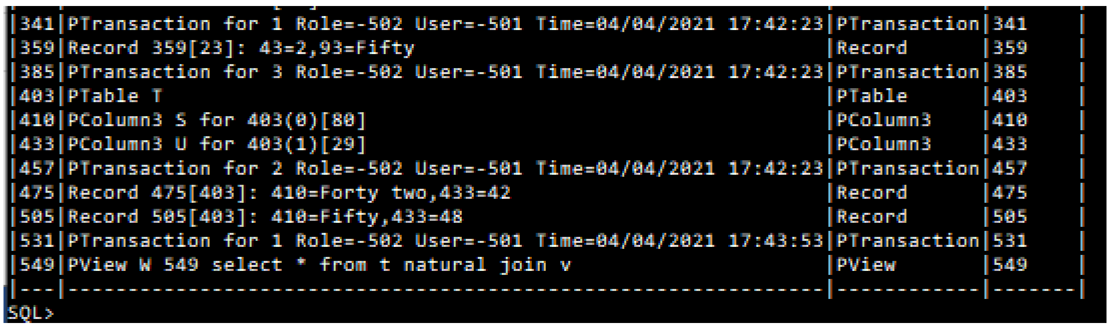
```
|278|Update 206[23]: 43=1,93=Forty two Prev:206                    |Update      |206    |
|314|PTransaction for 1 Role=-502 User=-501 Time=04/04/2021 17:38:30|PTransaction|314    |
|332|Delete Record 233[23]                                          |Delete1     |233    |
|---|---------------------------------------------------------------|------------|-------|
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.)

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
```



```
|341|PTransaction for 1 Role=-502 User=-501 Time=04/04/2021 17:42:23|PTransaction|341    |
|359|Record 359[23]: 43=2,93=Fifty                                  |Record      |359    |
|385|PTransaction for 3 Role=-502 User=-501 Time=04/04/2021 17:42:23|PTransaction|385    |
|403|PTable T                                                       |PTable      |403    |
|410|PColumn3 S for 403(0)[80]                                      |PColumn3    |410    |
|433|PColumn3 U for 403(1)[29]                                      |PColumn3    |433    |
|457|PTransaction for 2 Role=-502 User=-501 Time=04/04/2021 17:42:23|PTransaction|457    |
|475|Record 475[403]: 410=Forty two,433=42                          |Record      |475    |
|505|Record 505[403]: 410=Fifty,433=48                              |Record      |505    |
|531|PTransaction for 1 Role=-502 User=-501 Time=04/04/2021 17:43:53|PTransaction|531    |
|549|PView W 549 select * from t natural join v                     |PView       |549    |
|---|---------------------------------------------------------------|------------|-------|
SQL>
```

As before we will examine the framing of the new view at the start of the next Instance(). The next step will be

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

We see View 549 is

```
{View Name=W 549 Definer=-502 Ppos=549 Query select * from t natural join v Ppos: 549
        Cols (A:@4[@7],Q:@2[@7],S:@0[403]@3[@7],U:@1[403]) 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: 23,403}
```

Note the ambiguous reference S is resolved to both tables in the Cols property.

We show the Framing for View 549 but omitting the entries repeated from V's framing and the obrefs:

```
{Framing (..
 403 Table Name=T 403 Definer=-502 Ppos=403
        Domain TABLE (410,433)([410,Domain CHAR],[433,Domain INTEGER])  KeyCols: ,
 410 TableColumn 410 Definer=-502 Ppos=410 Domain CHAR Table=403,
 433 TableColumn 433 Definer=-502 Ppos=433 Domain INTEGER Table=403,
 551 CursorSpecification 551 RowType:(@0,@1,@2,@4,@3) Where: Source={select * from t natural join v}
        Union: 552,
 552 QueryExpression 552 RowType:(@0,@1,@2,@4,@3) Where: Left: 553 ,
 553 QuerySpecification 553 RowType:(@0,@1,@2,@4,@3) Where: TableExp 555,
 554 SqlStar Name=* 554 CONTENT From:553 CONTENT,
 555 TableExpression 555 Nuid=556 RowType:(@0,@1,@2,@4|@3) Where: Target: 556,
 556 JoinPart 556 RowType:(@0,@1,@2,@4|@3) Where:557 NATURAL INNER join558 on @5
        matching @0=@3 @3=@0,
 557 From Name=T 557 RowType:(@0,@1) OrdSpec (0=@0) Where: Target=403,
 558 From Name=V 558 RowType:(@2,@3,@4) OrdSpec (0=@3) Where: Target=@7,
 @0 SqlCopy Name=S @0 From:557 Domain CHAR copy from 410,
 @1 SqlCopy Name=U @1 From:557 Domain INTEGER copy from 433,
 @2 SqlCopy Name=Q @2 From:558 Domain INTEGER copy from 158,
 @3 SqlCopy Name=S @3 From:558 Domain CHAR copy from 159,
 @4 SqlCopy Name=A @4 From:558 Domain INTEGER copy from 160,
 @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)
        ([@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: @9,
 @9 QueryExpression @9 RowType:(@2,@3,@4) Where: Left: @10 ,
 @10 QuerySpecification @10 RowType:(@2,@3,@4) Where: TableExp @11,
 @11 TableExpression @11 Nuid=@12 RowType:(@2,@3,@4) Where: Target: @12,
 @12 From Name=P @12 RowType:(@2,@3,@4) Where: Target=23,
 @13 SqlCopy Name=R @13 From:@12 Alias=S Domain CHAR copy from 93,
 @14 SqlCopy Name=A @14 From:@12 Domain INTEGER copy from 115)
Data: (..
 403 TableRowSet 403(410,433) targets: 403=403 Source: _,
 555 TableExpRowSet 555(@0,@1,@2,@4|@3) key (@0,@1,@2,@4) targets: 23=64,403=557 Source: 556,
 556 JoinRowSet 556(@0,@1,@2,@4|@3) targets: 23=64,403=557 JoinCond: (@5)
        matching @0=@3 @3=@0 First: @6 Second: @15,
 557 SelectedRowSet 557(@0,@1) targets: 403=557 Source: 403 SQMap: (@0=557[410],@1=557[433]),
 558 SelectedRowSet @12(@2,@3,@4) key (@2,@3,@4) targets: 23=64 Source: 64
        SQMap: (@2=@12[43],@3=@12[93],@4=@12[115]),
 @6 OrderedRowSet @6(@0,@1) key (@0) order (@0) targets: 403=557 Source: 557,
 @11 TableExpRowSet @11(@2,@3,@4) key (@2,@3,@4) targets: 23=64 Source: @12,
 @12 SelectedRowSet @12(@2,@3,@4) key (@2,@3,@4) targets: 23=64 Source: 64
        SQMap: (@2=@12[43],@3=@12[93],@4=@12[115]),
 @15 OrderedRowSet @15(@2,@3,@4) key (@3) order (@3) targets: 23=64 Source: @12) ObRefs: ..
Result 556 Results: (,161 163,162 163,163 163,164 164,552 555,553 555,555 555,556 556)}
```

There are quite a few complicating factors here. First, note that the reference to view V in object 558 results in an instancing of V to the new uid @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 556 and marked hidden. Third, the join process requires the two operands of the join to be ordered (see rowSets @6 and @15). Fourth, note that in 556 we are tracking equivalent expressions @0 and @3. But all of these aspects are standard implementation details.

## Updating a Viewed Join

This time, the VirtualRowSet is

```
{VirtualRowSet #8(%0,%1,%2,%3,%4) where (#31) matches (%2=2) targets: #8=549
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True,UpdateAssignment Vbl: %3 Val: #21=True)}
```

and the result of Instancing (again omitting rows that are the same as the above) is

```
{(#1=UpdateSearch #1 Nuid=#8 Target: 549,
  #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=549 Query select * from t natural join v Ppos: 549
        Cols (A:%3[%18],Q:%2[%18],S:%0[403]%4[%18],U:%1[403]) 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: 23,403,
  %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=403,
  %13=SqlCopy Name=U %13 From:%12 Domain INTEGER copy from 433,
  %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 159,
  %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)
        ([%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: %20,
  %20=QueryExpression %20 RowType:(%2,%4,%3) Where: Left: %21 ,
  %21=QuerySpecification %21 RowType:(%2,%4,%3) Where: TableExp %22,
  %22=TableExpression %22 Nuid=%23 RowType:(%2,%4,%3) Where: Target: %23,
  %23=From Name=P %23 RowType:(%2,%4,%3) Where: Target=23,
  %24=SqlCopy Name=R %24 From:%23 Alias=S Domain CHAR copy from 93,
  %25=SqlCopy Name=A %25 From:%23 Domain INTEGER copy from 115,
  %27=SqlCopy Name=A %27 From:%16 Domain INTEGER copy from 160)}
```

After Review the rowsets are:

```
{(#8=JoinRowSet %11(%0,%1,%2,%3|%4) key (%0,%1,%2,%3,%4) where (#31) matches (%2=2)
        targets: 23=64,403=%12
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True,UpdateAssignment Vbl: %3 Val: #21=True)
        JoinCond: (%15) matching %0=%4 %4=%0 First: %14 Second: %26,
  %10=TableExpRowSet %10(%0,%1,%2,%3|%4) key (%0,%1,%2,%3) targets: 23=64,403=%12 Source: %11,
  %11=JoinRowSet %11(%0,%1,%2,%3|%4) key (%0,%1,%2,%3,%4) where (#31) matches (%2=2)
        targets: 23=64,403=%12
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True,UpdateAssignment Vbl: %3 Val: #21=True)
        JoinCond: (%15) matching %0=%4 %4=%0 First: %14 Second: %26,
  %12=SelectedRowSet %12(%0,%1) key (%0,%1) targets: 403=%12
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) Source: 403
        SQMap: (%0=%12[410],%1=%12[433]),
  %14=OrderedRowSet %14(%0,%1) key (%0) order (%0) targets: 403=%12
        Assigs:(UpdateAssignment Vbl: %1 Val: #16=True) Source: %12,
  %22=TableExpRowSet %22(%2,%4,%3) key (%2,%4,%3) targets: 23=64 Source: %23,
  %23=SelectedRowSet %23(%2,%4,%3) key (%2,%4,%3) matches (%2=2) targets: 23=64
        Assigs:(UpdateAssignment Vbl: %3 Val: #21=True) Source: 64
        SQMap: (%2=%23[43],%3=%23[115],%4=%23[93]),
  %26=OrderedRowSet %26(%2,%4,%3) key (%4) order (%4) matches (%2=2) targets: 23=64
         Assigs:(UpdateAssignment Vbl: %3 Val: #21=True) Source: %23)}
```

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

However, JoinRowSet has an override for Update (also Insert and Delete). The Split method traverses the JoinRowSet and creates explicit rowsets for left and right. These are added to the join operands to override the usual traversal rules. Then the Update (rep Insert, Delete) happens first for the first operand and then for the second. In this way, for example, a statement-level trigger will be called just once for the group of updates (inserts, deletes).

Here the two explicit rowsets have just one row each

```
{(0=(505, (%0=Fifty,%1=48) %14))}
{(0=(359, (%2=2,%4=Fifty,%3= Null) %26))}
```

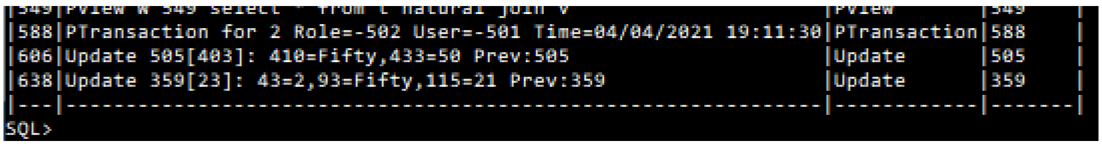
Then %14.Update() leads to %12.Update() leads to 403.Update(). The Transition Cursor is `{(%0=Fifty,%1=48) %28}` and the TargetCursor `{(410=Fifty,433=48) %28}`. After the UpdateAssignments have been applied, we have values `{(410=Fifty,433=50)}`, and _Update gives

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

Similarly for the secodn operand of the join, we get

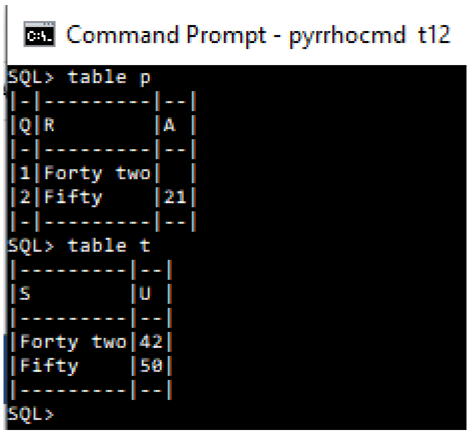
```
{Update 359[23]: 43=2,93=Fifty,115=21 Prev:359}
```

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



```
|349|PView W 349 select * from t natural join v                |PView       |349    |
|588|PTransaction for 2 Role=-502 User=-501 Time=04/04/2021 19:11:30|PTransaction|588    |
|606|Update 505[403]: 410=Fifty,433=50 Prev:505                 |Update      |505    |
|638|Update 359[23]: 43=2,93=Fifty,115=21 Prev:359              |Update      |359    |
|---|---------------------------------------------------------------|------------|-------|
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



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