In response to some questions which came up during and after the October GotoMeeting presentation about this small and simple JAVA frontend to browse into IMS data ("S2I" at Oct 10th) I will give here answers as best I can. I also added a few more tool details.

1) **Security**

There is nothing extra or special regarding our little JAVA frontend. We had already PSB security setup, we are using RACF and we use the default RCLASS "IMS":

```
 BROWSE     PIMS.PROCLIB(DFSPB000) - 01.06
OTMA=Y,
GRNAME=PRODGRP,
OTMANM=PROD,
OTMASE=C,
USERVAR=,
RRS=Y,
CSLG=000,
DC=000,
AOI1=R,
AOIS=R,
DFSDF=000,
APPCSE=N,
ODBASE=Y,
```

As you can see from DFSPB we use IMS/Connect Security and ODBASE=Y.

(As well RRS ... not only for the CTLREG, it's also switched on for ODBM as you can see the msg CSL4002I in the following ODBM Joblog picture)

Once I tried a PSB without Authority … (meanwhile I deleted/refreshed the S2I "information" log bar, anyway …) … for that I still could see at the backend side (my failing APSB call) IMS message:

```
13:04:50  DFS554A PIMSODBM 00011 PIMSODBM NULL    (7)          000,0438
13:04:50    2017/298 13:04:50 PROD
```

… in addition this msg was issued in ODBM ASID (13:04:50) :

```
10:32:57  IEF403I PIMSODBM - STARTED - TIME=10.32.57
10:32:58  CSL4002I ODBM Registration with RRMS/MVS complete  PIMSOD
10:32:58  CSL4005W ODBM Failed to connect to IMS datastore PROD due to IMS not available  PIMSOD
10:32:58  CSL0020I ODBM READY PIMSOD
10:33:37  CSL4004I ODBM Connected to IMS datastore PROD  PIMSOD
13:04:50  ---- INT6853 THE CURRENT DATE IS WEDNESDAY, 25 OCT 2017 ----
13:04:50  DFS2855A APSB SECURITY CHECK FAILED FOR ODBA0070  USERID-SYS1HKI  PSB-NULL    SAF RC-04 RACROUTE-AUTH    RC(0004,
13:04:50 0008) PROD
```

From IMS Msg & Codes:

0438

Explanation: The ODBA user request to schedule the PSB named on the APSB call failed SAF RACROUTE processing.

Analysis: The APSB request is unsuccessful. Message DFS2855A is issued if SAF RACROUTE AUTH call was not successful.

System action: The APSB request is unsuccessful…

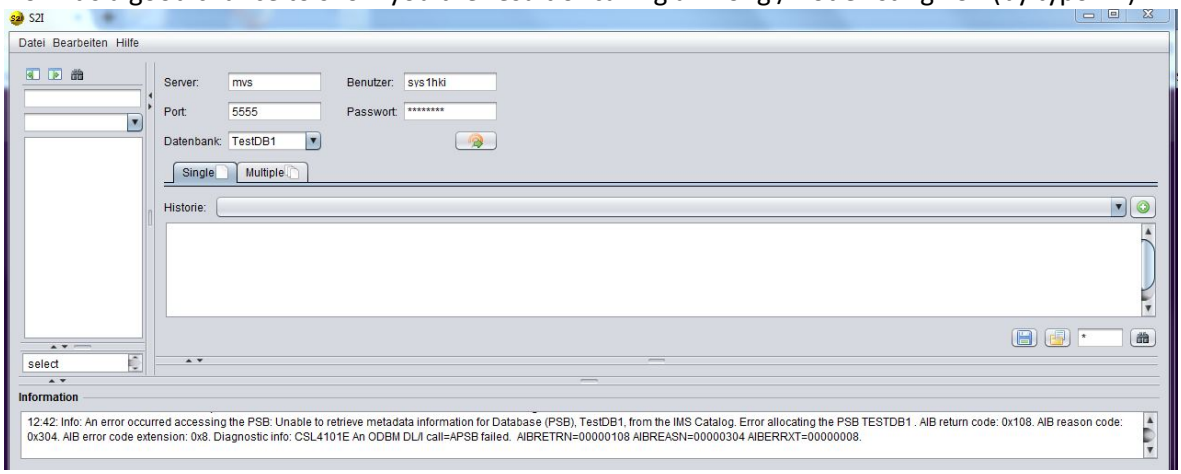
… Module: DFSDASP0

Not sure why the PSB isn't named here (only the thread name ODBA0070), anyway for me the "APSB SECURITY CHECK FAILED" was sufficient.

So it's just as IBM recommends: IMS/Connect is doing authentication and IMS CTL checks for authorization. A quite precise overview you can find searching for a recent presentation from Deepak Kohli (NE CAN IMS RUG 2016 – Quebec City) in Slideshare and elsewhere,
called "IMS Open Database & IMS Connect".

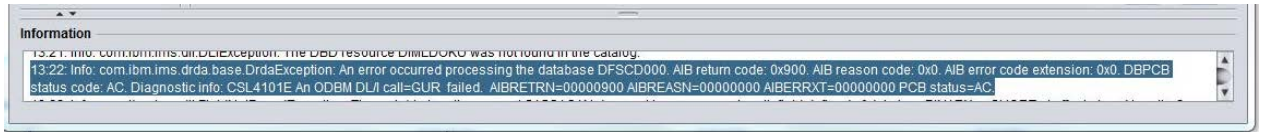
Slide 35 there shows how we did the RRS setup (no DRA for our ODBM-IMS threading) and slides 37-39 show the security setup.
Another interesting presentation (thanks Richard Tran and Kevin Hite from the IMS lab for all their valuable input !) is the “Java Development on System z Best Practices” by David Ormsby at the Share Anaheim Winter 2014 event. The presentation is listed under 14748 here:
https://share.confex.com/share/122/webprogram/uploadlistall.html
At slide 20 the security considerations are covered ... among further considerations and interesting hints!

Now it’s a good chance to show you the result of calling a wrong / not existing PSB (by typo f.i.) :



… the typical AIBRETRN / RSN combination of 108/304 .. PSB not found.

I forgot what exactly I tried here … I just “played hard” to provoke any error and to check how robust the software behaves:

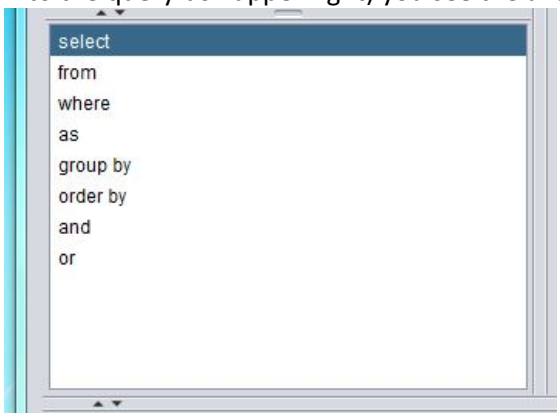


… what also reveals that calling the metadata from the CATALOG (a specific DBPCB against the entire DBD metadata information) will trigger a GUR call (entire record) against the CATALOG (HALDB Database). The AIBRETRN x’900’ looks a bit strange and funny within the docs – like a “spare” reason for unknown … The DBPCB statcode AC gives a better clue: Hierarchic error in SSAs.

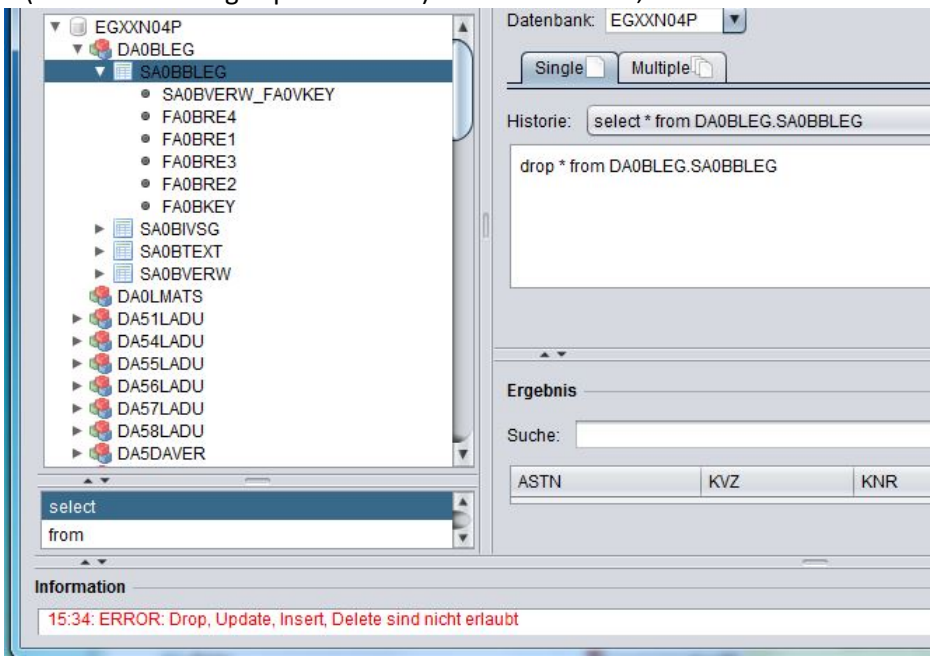
2) Locking

a. The suppressed manipulation

The locking happens according the PROCOPT of the used PSB … that's for sure !
But its behavior here is the same as we use a regular "READ FOR UPDATE":
Since it will never really update any segment (as long all manipulation statements are suppressed!) it is more a short time "consistent read" lock along the way your select call cruises thru the segments.
As Michael said in the web meeting, the property file of the end user, underlying the JAVA client software, will suppress any manipulation (the property file has also security on, so any manipulation on that is excluded as well).
In the left lower side in the statement bar (you can drag& drop from there for editing the call statement into the query box upper right) you see the allowed ones:



Well, if a frontend user would write his own query command and using f.i. DROP…smthg… this ends up in a (German text msg expression for) "… not allowed" , see the red information down here.

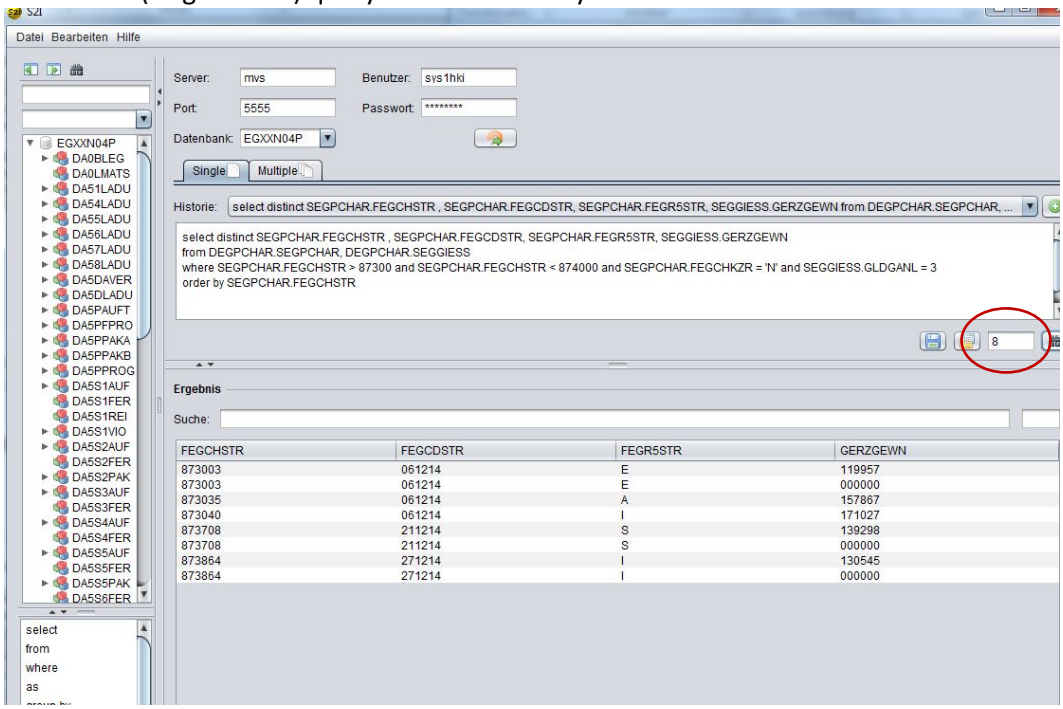


But of course this might be a serious "issue" if any manipulation will be allowed … and then, matching the real intent with the defined "higher" PROCOPTs it can lead to more potential LOCKing. Then it's up to the user … and needs to be considered, yes! Locking considerations as usual …

Also, it's up to the Sysprogs/ DBAdmins to build some "read only" PSBs and provide those for certain use cases. Then it's driven by GOx intent – no (Locking) harm at all.

b. Tracing the story

Anyway, I did some traces on the backend (for just our default scenario; with suppressed manipulation using PCB with an A or R intent).
Here is the (single mode) query call submitted by the frontend:



Please note the result limit of 8. This is a useful function and quite nice … so for a first try to see if the result is as expected…
(Similar to the use of the MaxRows connection property).

Just to show you and prove this process I did a PSB trace. Converted into DDLT0 format it looks like this (the GHU is complete, for subfollowing cal**L** multi line and ompar**E** DATA lines are truncated):

```
U     DATE=2017/301 TIME=13.33.17 EGXXN04P           01700002
S 1 1 1 1 1                              01700004
S 1 1 1 1 1    000591                      01700006
L    GHU  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700008
     CONT )                             X01700010
        SEGGIESS(GLDGANL EQ3)                01700012
E    DATA 873003071214DRNJNNNNNJNNT 3N 000 06121450146570000003107X01700014
        1824D 11SMNPB30 X24 HD   107182411SMNPB30 X24 HD   DX01700016
        044644201000131090SBG BURBACH SIE  4 3 0       X01700018
                          X X EE2LDX3 0000000X01700020
        000008685928685948685910000000000000000000000O0 E205095NX01700022
         A071214MK 0   0   0   0   0   0   0X01700024
           0   00000000000000000000000000000000000000000 X01700026
                                  X01700028
                                  X01700030
                                  X01700032
                               2402402402X01700034
        402400001000001001001000000010              X01700036
             0119957        3100000003000000047000X01700038
        0051000005000000000 00000000000000000000000000000000000X01700040
        00000 000000000000000000000000000000000000000003    X01700042
         100 119957          026C475   3744J 1519 X01700044
        000000154716052515292430352275 2 8080808080 205X01700046
         000000-A2                        01700048
E  02  SEGGIESS 0007873003A                    01700050
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700052
     CONT )                             X01700054
        SEGGIESS(GLDGANL EQ3)                01700056
E    DATA 873003071214DRNJNNNNNJNNT 3N 000 06121450146570000003107X01700058
....                                      .
E  02  SEGGIESS 0007873003B                    01700094
```

```
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700096
....                                    .
E  02  SEGGIESS 0007873035A                  01700138
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700140
...                                     .
E  02  SEGGIESS 0007873040A                  01700182
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700184
...                                     .
E  02  SEGGIESS 0007873708A                  01700226
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700228
...                                     .
E  02  SEGGIESS 0007873708B                  01700270
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700272
...                                     .
E  02  SEGGIESS 0007873864A                  01700314
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700316
...                                     .
E  02  SEGGIESS 0007873864B                  01700358
L    GN  SEGPCHAR*D(FEGCHSTRGT87300 &FEGCHSTRLT874000&FEGCHKZREQNX01700360
...                                     .
E  01 GE SEGPCHAR 0006873911                   01700378
```

As you can see the query call gets kicked off by a GHU and is moving on by GN calls only. The last GN (8th GN call) got a statcode GE in its comparE PCB statement, caused by the limit of 8.

c. Tracing the locks

Now about the locks we caused by that.
First the (x'67FA' and DFSERA40) PI trace formatted:

```
DATE:  10/28/17
MODULE PST TIME (*=ET) CALLR ACT LEV WC WFC SEQN FDBK RC PC ID= (RBA DMB ... COMMENTS
DLA0  0B  13:33:38.737   GN       08B7         45     DL/I CALL
DLA0  0B  13:33:38.739   GN       08CF         46     DL/I CALL
DLA0  0B  13:33:38.742   GN       08F0         47     DL/I CALL
DLA0  0B  13:33:38.742   GN       08FA         48     DL/I CALL
DLA0  0B  13:33:38.743   GN       0908         49     DL/I CALL
DLA0  0B  13:33:38.745   GN       091C         50     DL/I CALL
DLA0  0B  13:33:38.746   GN       0928         51     DL/I CALL
DFS707I END OF FILE ON INPUT
```

(I have no clue why the GHU isn't captured here, just all 7 GNs)

Second the DLI/Lock table trace (x'67FA and DFSERA60, our ODBA thd has PST number x'0E'):

```
FUNCTION    WORD 0  WORD 1  WORD 2  WORD 3  WORD 4  WORD 5  WORD 6  WORD 7
I/O POSTED   610008B1 0E27A3C0 00069598 F00108B0 11F27000 103C7038 7F000019 DE000201
PSTBYLCT    E20B08B2 002C0100 D9050800 118A77B4 11F27004 11E9DAE0 AE000100 005B7494
PSTBYLCT    E20B08B3 002C0100 D9050800 118A77B4 11F27008 11E9DAE0 42000000 005B7498
PSTBYLCT    E20B08B4 002C0100 D9050800 118A77B4 11F27010 11E9DAE0 42000000 005B74A0
PSTRELLR    DD0B08B5 002C0100 C1000000 118A7724 11F27010 00000000 80000000 00000000
ANALYZE CALL  AA0B08B6 0E28EC58 GN    0E446674 01090800 005B74A0 00004040 118A77B4
PI TRACE    CA0B08B7 08000000 C4D3C1F0 0B5DD2D0 0000002D C7D54040 00000000 00000000
PSTBYLCT    E20B08B8 002C0100 D9050800 118A77B4 11F2700C 11E9DAE0 56000000 005B749C
I/O STARTED   600008B9 0E27A3C0 0E114060 000000B8 11F1F000 080C0858 01000019 DE000301
I/O POSTED   610008BA 0E27A3C0 00069598 F00108B9 11F1F000 00000000 7F000019 DE000301
PSTBYLCT    E20B08BB 002C0100 D9050800 118A77B4 11F1F004 11E9DA50 AE000100 005BF484
PSTBYLCT    E20B08BC 002C0100 D9050800 118A77B4 11F1F008 11E9DA50 42000000 005BF488
PSTBYLCT    E20B08BD 002C0100 D9050800 118A77B4 11F1F00C 11E9DA50 42000000 005BF48C
I/O STARTED   600008BE 0E27A3C0 0E114060 000000B9 11F17000 00000000 01000019 DE000401
I/O POSTED   610008BF 0E27A3C0 00069598 F00108BE 11F17000 00000000 7F000019 DE000401
PSTBYLCT    E20B08C0 002C0100 D9050800 118A77B4 11F17004 11E9D9C0 AE000100 005C7474
PSTBYLCT    E20B08C1 002C0100 D9050800 118A77B4 11F17008 11E9D9C0 42000000 005C7478
PSTBYLCT    E20B08C2 002C0100 D9050800 118A77B4 11F1700C 11E9D9C0 42000000 005C747C
I/O STARTED   600008C3 0E27A3C0 0E114060 000000BA 11F0F000 00000000 01000019 DE000501
I/O POSTED   610008C4 0E27A3C0 00069598 F00108C3 11F0F000 00000000 7F000019 DE000501
PSTBYLCT    E20B08C5 002C0100 D9050800 118A77B4 11F0F004 11E9D930 AE000100 005CF464
PSTBYLCT    E20B08C6 002C0100 D9050800 118A77B4 11F0F008 11E9D930 42000000 005CF468
PSTBYLCT    E20B08C7 002C0100 D9050800 118A77B4 11F0F00C 11E9D930 42000000 005CF46C
I/O STARTED   600008C8 0E27A3C0 0E114060 000000BB 11F07000 00000000 01000019 DE000601
I/O POSTED   610008C9 0E27A3C0 00069598 F00108C8 11F07000 3000704C 7F000019 DE000601
PSTBYLCT    E20B08CA 002C0100 D9050800 118A77B4 11F07004 11E9D8A0 AE000100 005D7454
PSTBYLCT    E20B08CB 002C0100 D9050800 118A77B4 11F07008 11E9D8A0 42000000 005D7458
PSTBYLCT    E20B08CC 002C0100 D9050800 118A77B4 11F07010 11E9D8A0 42000000 005D7460
PSTRELLR    DD0B08CD 002C0100 C1000000 118A7724 11F07010 00000000 80000000 00000000
ANALYZE CALL  AA0B08CE 0E28EC58 GN    0E446674 01090800 005D7460 00004040 118A77B4
PI TRACE    CA0B08CF 08000000 C4D3C1F0 0B5DD2D9 0000002E C7D54040 00000000 00000000
…
```

And all the way down for 7 GN calls in total the same trace pattern: The x'CA' proving PI gets traced, some more or less byte locates, I/Os, then the release record (x'DD') and next AA call. As expected no x'CC' or others ... so no other lock (we are not updating), neither a block lock - of course not ☺ (we are not deleting or inserting)!

(The same here, I only see the GNs captured)
The x'CA' entries for PI correspond with the PI trace entries above (as expected).

The IMS story logged is between 08 and 07, as short as this:

```
08 RECORD -  2017-10-28 11:33:17.087570 UTC
 00000000 000000  009C0000 0840C5C7 E7E7D5F0 F4D7D6C4  C2C1F0F1 F0C50000 00400000 00000040
                             *..... EGXXN04PODBA010E... ..... *
 00000020 000020  40404040 404040C5 C7E7E7D5 F0F4D705  0011D6C4 C2C1F0F1 F0C50000 00070000
                             *    EGXXN04P...ODBA010E......*
 00000040 000040  00000000 00070000 00000007 00000000  00000000 0000000C 00000000 0000000C
 00000060 000060  00000000 0001334C 11B489A8 11B6FCE0  235A32B3 A8E1FF40 2017301F 11331708
 00000080 000080  7558008C 00000000 00000000 D35AA0E4  0EF52050 00000000 00000628
56 RECORD -  2017-10-28 11:33:17.087570 UTC
 00000000 000000  005C0000 56070000 E3C5E2E3 40404040  00000011 C5C7E7E7 D5F0F4D7 00000000
                             *.*......TEST  ....EGXXN04P....*
 00000020 000020  00000000 00000000 00000000 D6C4C2C1  F0F1F0C5 00000007 00000000 00000000
                             *............ODBA010E............*
 00000040 000040  2017301F 03201098 9226008C D35AA0E4  0EF5246C 00000000 00000629
-5616          Yep, under RRS !
-5610          PH1 kickoff
-5611          PH1 end
-37
-5612          PH2 final end
-5607-
07 RECORD -  2017-10-28 11:33:17.110869 UTC
 00000000 000000  01C80000 07C5C7E7 E7D5F0F4 D7404040  40404040 40000400 00000000 00000000
                             *.H...EGXXN04P    ..........*
 00000020 000020  00000000 00000000 00000000 E3C9D4E2  D6C4C2D4 E3C9D4E2 D6C4C2D4 00000000
 . . .
```

3) IMS SSA feedback area

The question came up how to see, to collect or to report the DL/I call as result of SQL call submitted and converted… maybe the SSA and its filled fdbk area.
Well, that is beyond that tool, even beyond other tools. The JDBC drivers provide some trace facilities, also those for capturing the DL/I results.
Of course the Image Capture trace (PSB trace) would tell you, but it doesn't limitate just to your JAVA driven calls only and therefore might be too much effort on the IMS server side.

But in general for JDBC tracing I want to refer to this blog from Richard Tran where he is talking about the different trace packages in the JDBC driver:
https://imsinsiders.wordpress.com/2016/04/18/how-to-enable-the-ims-jdbc-trace/
… especially the DL/I trace part and the more deeper digging into the OPNQRY codepoint, also listed there. There you can see how it works with such a sequence of calls and that IMS Connect provides more efficiently the "RETRIEVE" and ODBM delivers back the result set.
Also the Share-Anaheim session from David Ormsby (mentioned above) covers at slides 24 and subfollowing this topic. Since the JDBC driver has added support in the Connection.nativeSQL method to show the DL/I translation there are ways to facilitate it.
It seems f.i. the IMS Explorer has an easy way to switch on that trace and passing it under the cover forward to the JDBC driver instance.

If the customer here using this S2I will ever exploit all the possible SQLs including manipulations within that JAVA frontend the customer then should have the traces at specific levels set up for diagnostics and for debugging of any performance problems. Then he would also add some code to facilitate it somewhere. For now and for just this native "data viewer" mode there is no need.

4) RRS now in the game ?

But some more things might be now under consideration for those customers just starting with ODBM and starting with the RRS managed threads (instead of DRA). That opens up the entire IMS transaction flow thru RRS and will definitely create (or add more) workload to the constant RRS log stream of DELAYED.URs – and most of them ends up in an IMS logging small "overhead" of 5616-5610-5611-5612 record sequence... at least this is true as long as no real "GLOBAL" transaction driven from distributed (or the AOS - APPC/OTMA SHaredQ support at that RRS level) will force RRS to do a real 2PC syncpoint coordination! Then it's anyway more to do …

But for those who never checked for the RRS setting and new to exploit RRS now, they should do so according to the RRS log streams defined to the system logger and captured in a CF structure and as well those shops who are running single- image-single-IMS, where the RRS ASID needs at least STAGING and OFFLOAD data sets for all RRS log streams. My recommendation here is to have an eye for those structure / DS sizing (IXGMIAPU against logger) and check the SMF88 for thresholds or offloads.

It should be easy to find a sizing large enough (according to your peak workload) that any UR will stay in DELAYED.UR structure (or STAGING DS) for its entire lifetime.

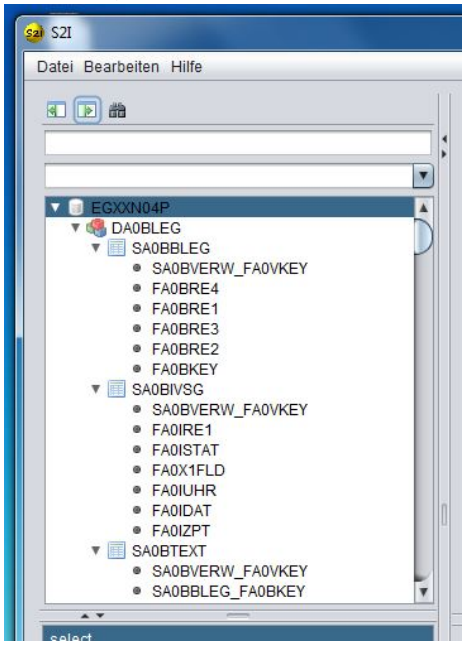
Experienced RRS users probably did already INACTIVATE the ARCHIVE log stream … so the shops new in RRS exploitation will consider the same.

Another issue I've noticed is the default value for *LOWOFFLOAD(0)* - this only makes sense for the continuous ARCHIVE stream (which might be suppressed anyway!), but never for the RM, RESTART or MAIN/DELAYED.UR streams. Well, here applies the same: RM and RESTART entries are usually not growing and should also never get offloaded from structure or Staging DS !

5) A few more screenshots from the frontend

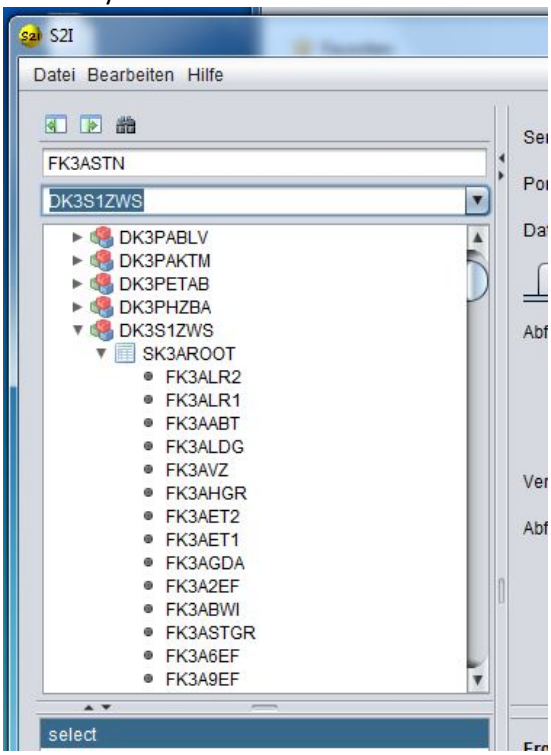
Back to the frontend itself,
It gives you – once you got your PSB metadata – the chance

a) To expand and close the view for the complete metadata structure (by the little expand menu icons)

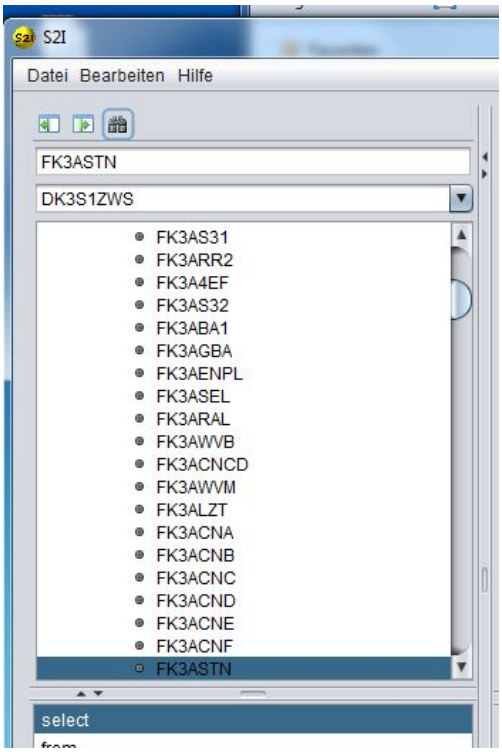


b) To search for a field across all PCBs … or even filtered by the second line PCB filter pull down menu.

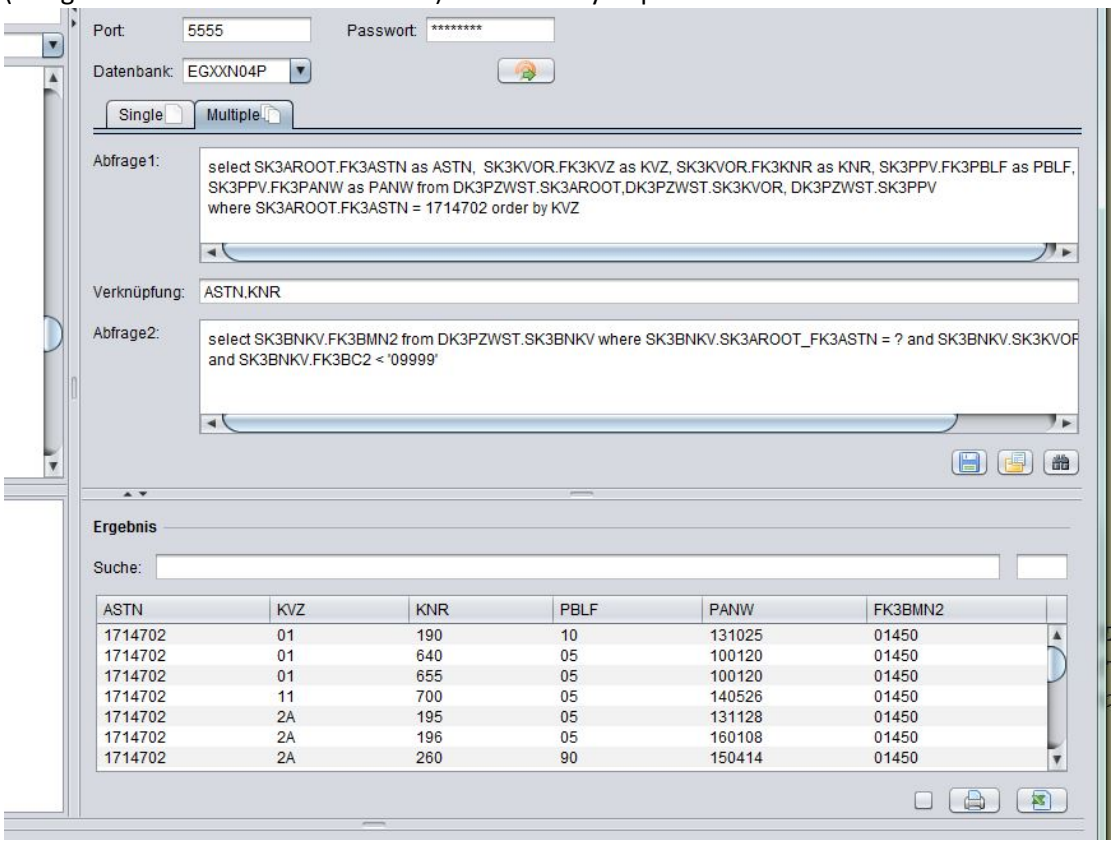
search by filter:



hit:



About the Search and substring functions we already told,
so once there is a resultset…
(lower right corner, here snapped from the result of the „JOIN“ call agaisnt two PCBs)
… you could search for strings there or summarize in columns a.s.o.
(all against incore buffered resultset) and as well you print or offload the table into an excel file:



Kind regards
Michael Schäfer (Saarstahl AG) & Henry Kiesslich (KIECO)