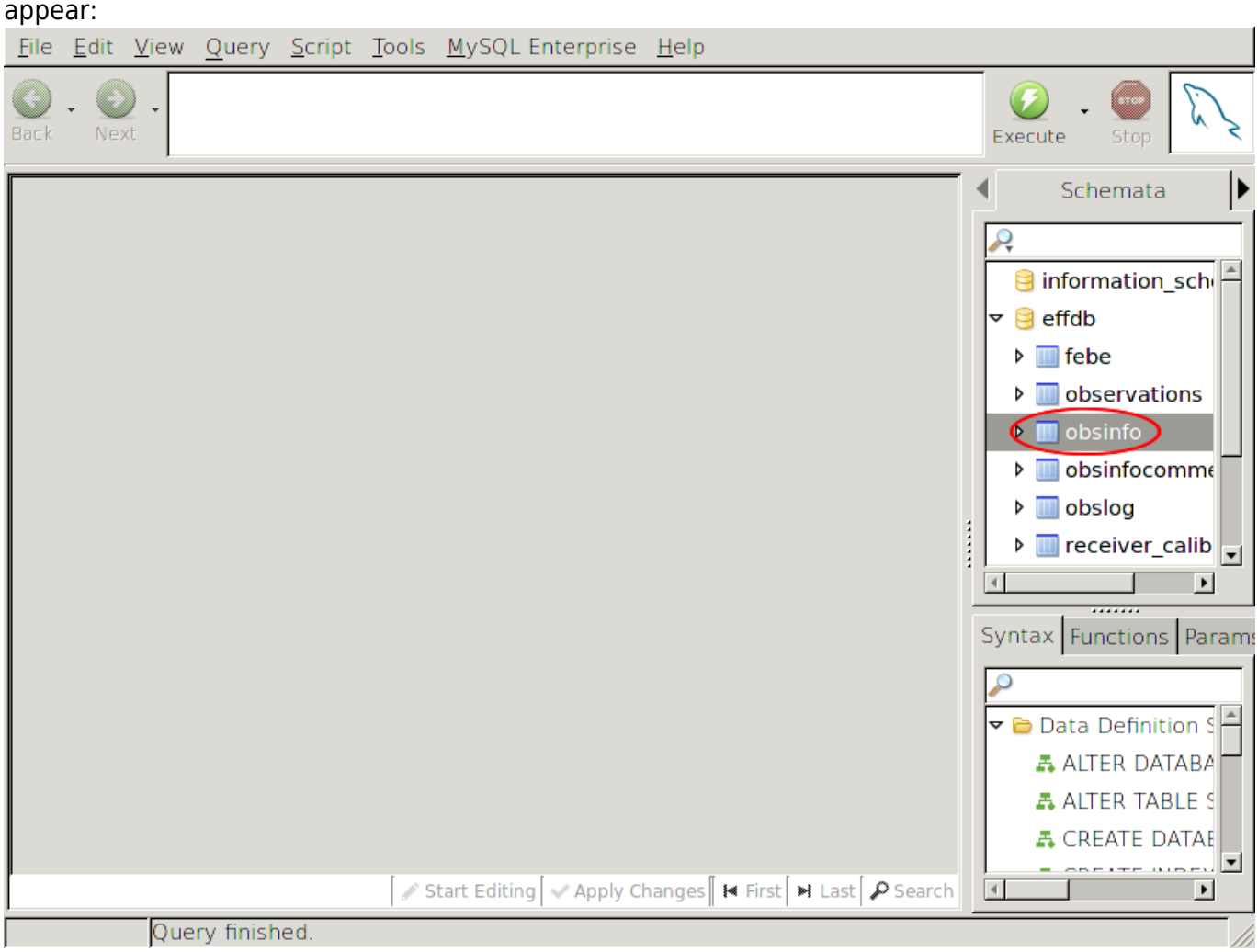
Effelsberg Database (MySQL)

In Effelsberg we store a lot of **auxiliary information** (not the astronomical data itself!) regarding the observations in an SQL database. For example, timestamps, project id's, sky positions, frequency setup, etc. Using a tool, like `mysql-query-browser`, one can easily query the database for information. (Please ask to Effelsberg staff, preferably H. Hafok or B. Winkel, on login names and passwords.)

Entries in the database include measurements since January 2009.

Basics

If one starts the `mysql-query-browser` on observer5, after logging in, a window like this will appear:



In the right part, several tables are listed. The most useful are

- **obsinfo** - contains a lot of observational parameters for every subscan of every measurement (except pulsar and VLBI observations)
- **receiver_parameters** - receiver parameters (frequency range, number of basebands/feeds, gain curves)
- **receiver_calibration** - typical calibration values (Tcal, antenna efficiency, etc.) for a number of frequencies

- **receiver_versions** - is the database version of the frontend files

Double-clicking a table brings up the most basic SQL query (into the query field):

```
SELECT * FROM effdb.obsinfo o LIMIT 0,1000
```

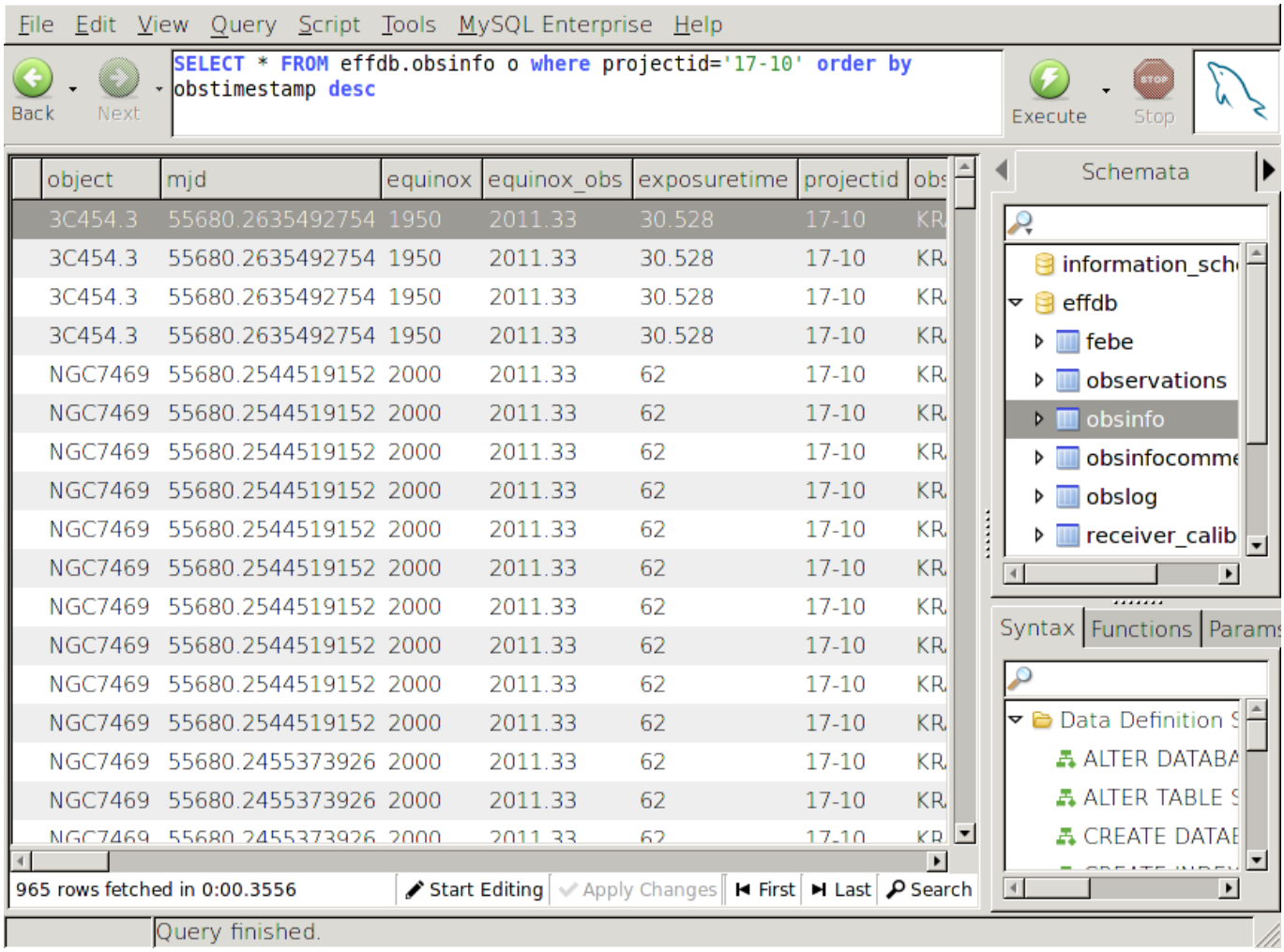
This will show the 1000 entries last added or changed. More useful is to order (ORDER BY) the results (e.g., by date and time) for example to show the 1000 most recent entries:

```
SELECT * FROM effdb.obsinfo o ORDER BY obstimestamp DESC LIMIT 0,1000
```

Filtering

It is very easy to filter the information using the WHERE-clause.

```
SELECT * FROM effdb.obsinfo o WHERE projectid='17-10'
SELECT * FROM effdb.obsinfo o WHERE observer LIKE '%kraus%' OR observer LIKE
'%AK%' # the '%' works AS wild-card
```



Joins

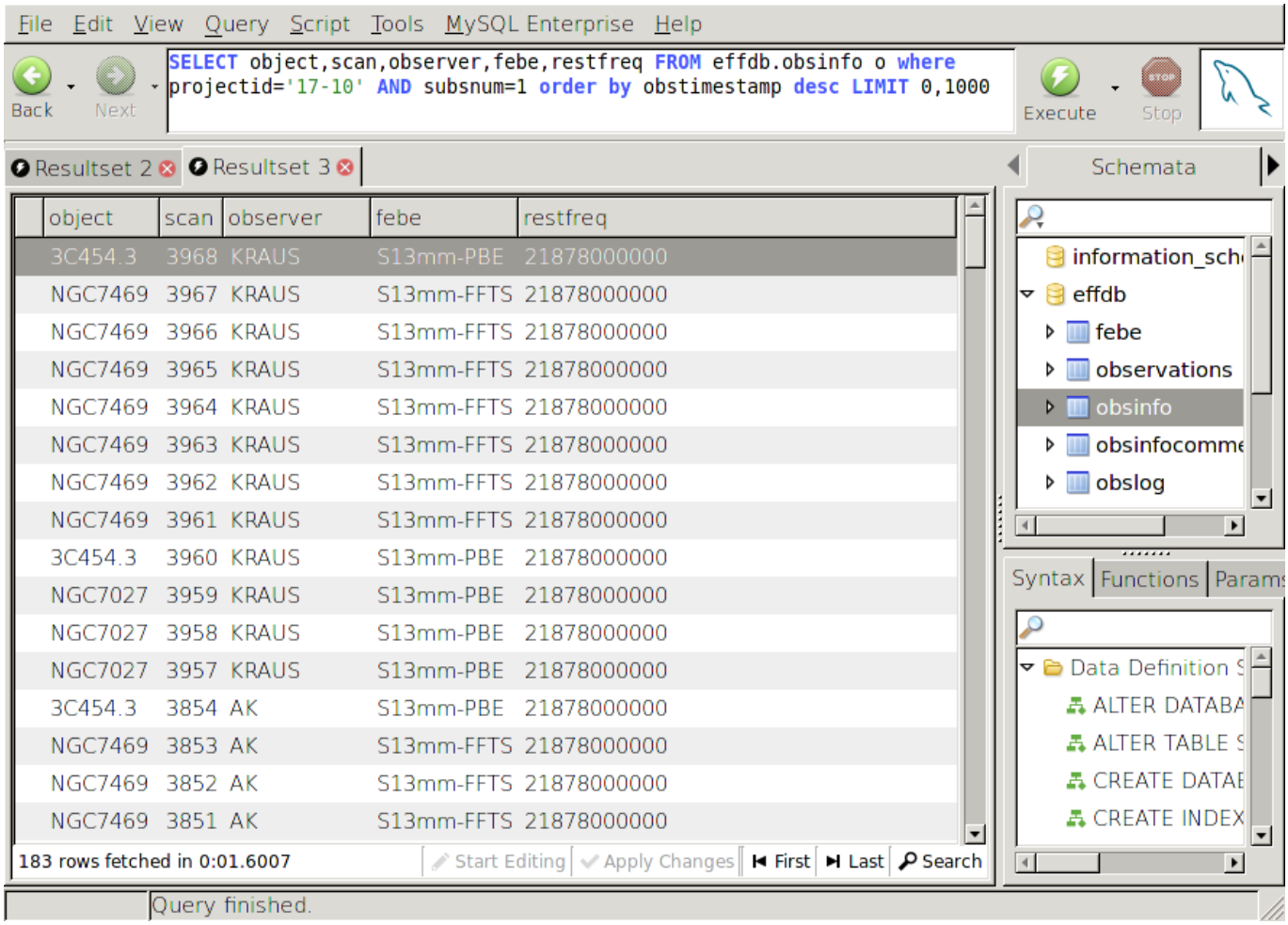
Joining means to link entries from different tables with each other. One differentiates between *inner* and *outer* joins (for more information see here).

```
SELECT * FROM effdb.receiver_calibration c INNER JOIN
effdb.receiver_parameters r ON
   c.frontend=r.frontend AND c.baseband=r.baseband AND c.subband=r.subband
```

This example will perform an inner join of the receiver parameters and receiver calibration tables forming a large table containing a lot of information on the receivers (just compare with a simple query on the individual tables...).

Joins can be very useful. In the following we will show increasingly complex queries to associate calibration information to the obsinfo database. Lets start simple

```
SELECT object,scan,observer,febe FROM effdb.obsinfo o WHERE
projectid='17-10' AND subsnum=1
   ORDER BY obstimestamp DESC LIMIT 0,1000
```



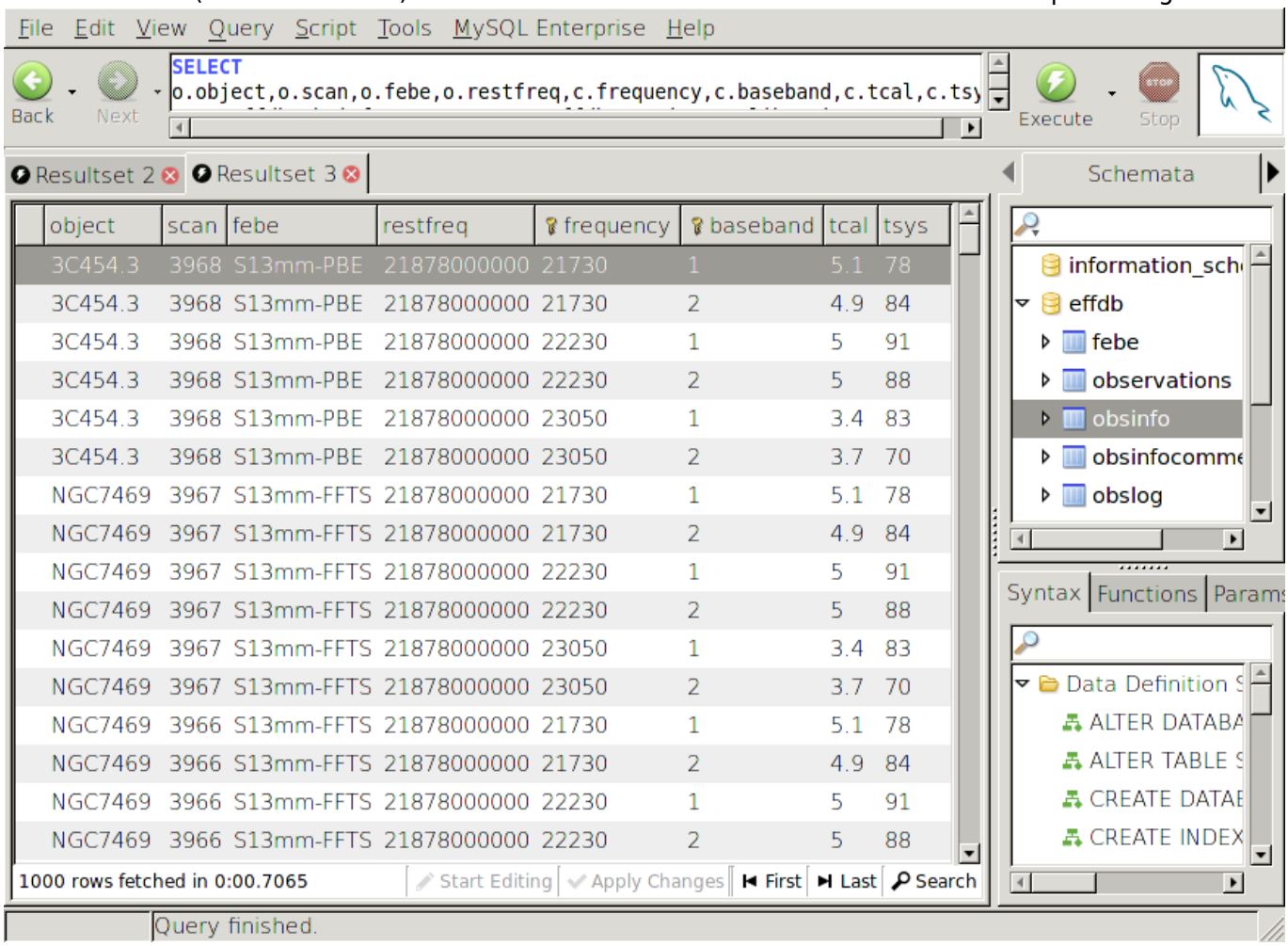
Now it would be nice, if we had calibration information (Tcal/Tsys values) for each entry.

To do this, we perform a join like this

```
SELECT
```

```
o.object,o.scan,o.febe,o.restfreq,c.frequency,c.baseband,c.tcal,c.tsys FROM
effdb.obsinfo o
   INNER JOIN effdb.receiver_calibration c ON locate(c.frontend,o.febe)>0
   WHERE o.projectid='17-10' AND o.subsnum=1 ORDER BY o.obstimestamp
DESC,c.frequency ,c.baseband ASC LIMIT 0,1000
```

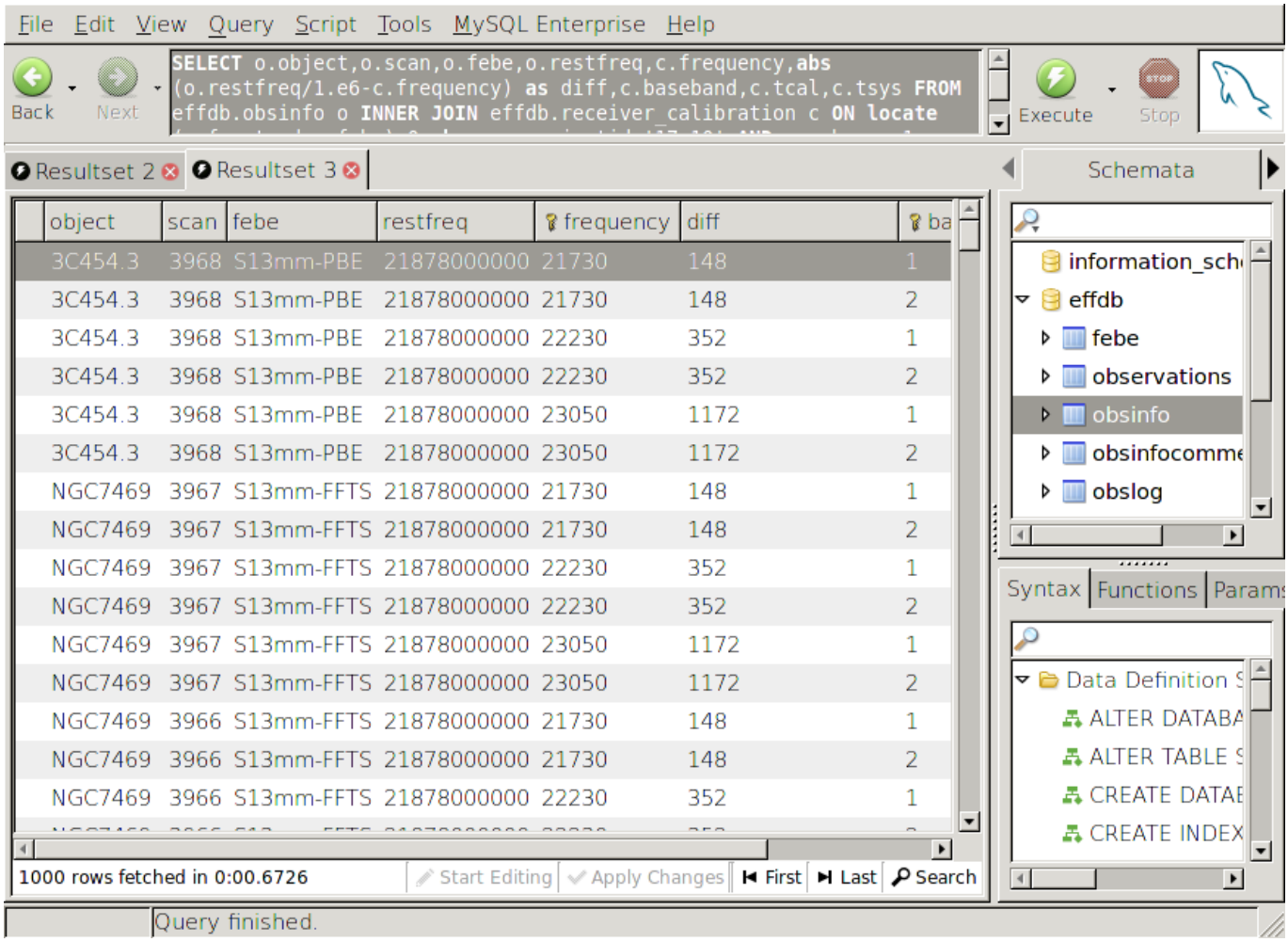
The basic syntax is the same as before. The key to associate the rows from obsinfo with receiver_calibration is the frontend name (e.g., "P13mm"). Unfortunately, in the obsinfo database there is only a `febe` keyword, which contains the frontend-backend pair (e.g., "P13mm-XFFTS"). To produce a match, we just perform a substring search, using the `LOCATE(s1,s2)` function, which will either return 0 (if s1 is not in s2) or the first occurrence of s1 in s2. The result looks promising.



However, for each row in obsinfo we might have several entries in receiver_calibration (for different polarization channels and frequencies).

To find the row which is closest in frequency, we could let compute MySQL the difference in frequency between both tables:

```
SELECT o.object,o.scan,o.febe,o.restfreq,c.frequency,abs(o.restfreq/1.e6-
c.frequency) AS diff,c.baseband,
   c.tcal,c.tsys FROM effdb.obsinfo o INNER JOIN effdb.receiver_calibration
c ON locate(c.frontend,o.febe)>0
   WHERE o.projectid='17-10' AND o.subsnum=1 ORDER BY o.obstimestamp
DESC,c.frequency ,c.baseband ASC LIMIT 0,1000
```



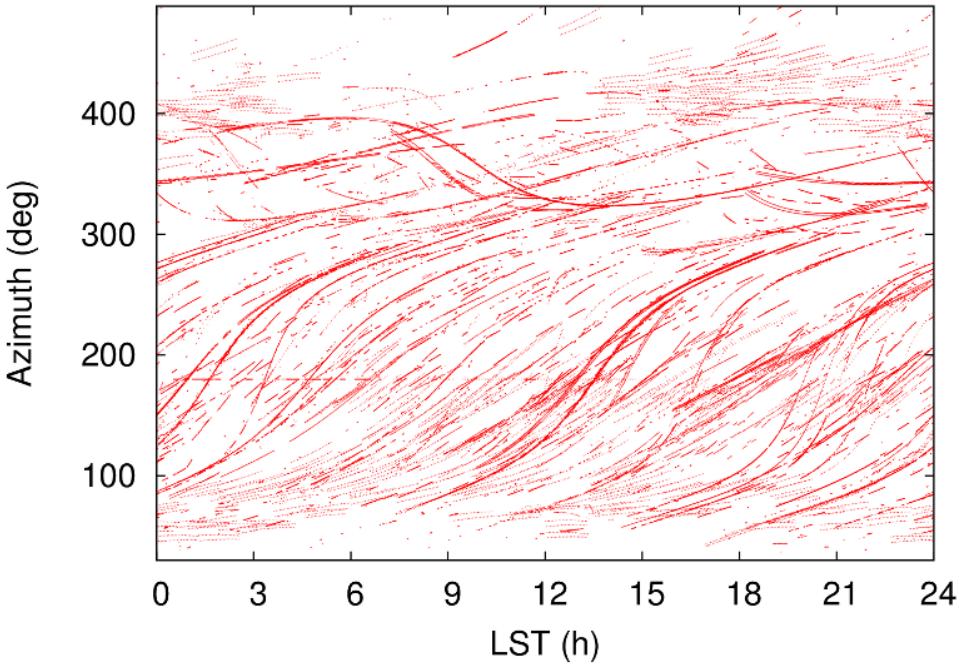
It would be also possible to just show the results with the smallest frequency "diff" values, but this would require nested queries, which we omit here for now.

Exporting data

It is very easy to save the results into a (csv-)file for further use (file → export resultset → csv). An example:

```
SELECT lst,azim FROM effdb.obsinfo o WHERE lst>0 LIMIT 0,100000
```

Plotting this does not reveal amazing scientific wisdom, yet looks nice.



Useful queries

Find all spectroscopic ON-OFFs toward a known calibrator.

```
SELECT object,projectid,observer,scan,obstimestamp,azim,elev,febe,feversion,
    bandwidth,restfreq FROM effdb.obsinfo o WHERE scantype='ONOFF' AND
scanmode='SAMPLE'
    AND object REGEXP
'(3C48|3C123|3C161|3C147|3C286|3C295|3C196|3C138|NGC7027|W3OH)'
    AND obstimestamp>='2011-05' ORDER BY obstimestamp DESC
```

Using python to work with the database

The following shows an example Python script which queries the database to show the exposure time of every W3MAIN observation since Feb, 2012.

```
import MySQLdb
import sys

# please ask our staff members for account details
user=""
passwd=""

def connectToDB(host = "127.0.0.1",port=3307,user = "" ,passwd = "",db =
"effdb"):
```

```
    connected=False
    conn=None
    try:
        conn = MySQLdb.connect (host = host,port=port,user = user,passwd =
passwd,db = db)
        print "successfully connected to db"
        connected=True
    except MySQLdb.Error, e:
        print "Error %d: %s" % (e.args[0], e.args[1])
        connected=False
    return conn,connected
#

conn,connected=connectToDB(user=user,passwd=passwd)
if not connected:
    print "could not connect to database"
    sys.exit(1)
#

squery="select obstimestamp,sum(24.*3600.*(sublastmjd-subfirstmjd))
        from effdb.obsinfo where object like 'W3MAIN' and
        obstimestamp>'2012-01' group by obstimestamp"
print squery
cursor = conn.cursor ()
cursor.execute (squery)
sqlResult = cursor.fetchall ()
#print self.row
if len(sqlResult)>0:
    for a in range(len(sqlResult)):
         print sqlResult[a][0],":",int(sqlResult[a][1]+0.5),"seconds"
else:
    print "no entries found"
```

Output:

```
2012-01-04T15:34:15 : 116 seconds
2012-01-04T23:01:49 : 116 seconds
2012-01-04T23:05:59 : 116 seconds
2012-01-04T23:12:07 : 116 seconds
2012-01-04T23:15:09 : 118 seconds
.
.
.
2012-02-06T21:05:10 : 116 seconds
2012-02-06T21:10:34 : 112 seconds
2012-02-08T15:51:34 : 116 seconds
2012-02-08T15:59:26 : 116 seconds
```

Note, that in order to connect to the server a local tunnel to the MPIfR SQL Server has to be created. Furthermore, the "MySQLdb" Python Module needs to be installed. Please ask local staff for help, or login to "observer5" (where everything is already installed) to run the scripts.