Photometry Database description

Version 1.0

Table of contents

[Database Overview](#)
- [Complete database scheme](#)
- [Simplified relations scheme](#)
- [Table Overview](#)
- [Table Details](#)
  - [The lightcurves table](#)
    - [The basic user request](#)
  - [The metadata_lc_json table](#)
    - [Example request](#)
- [The observations table](#)
  - [The metadata_obs_json table](#)
  - [The objects table](#)
    - [Cone-search request](#)

[The Database access](#)

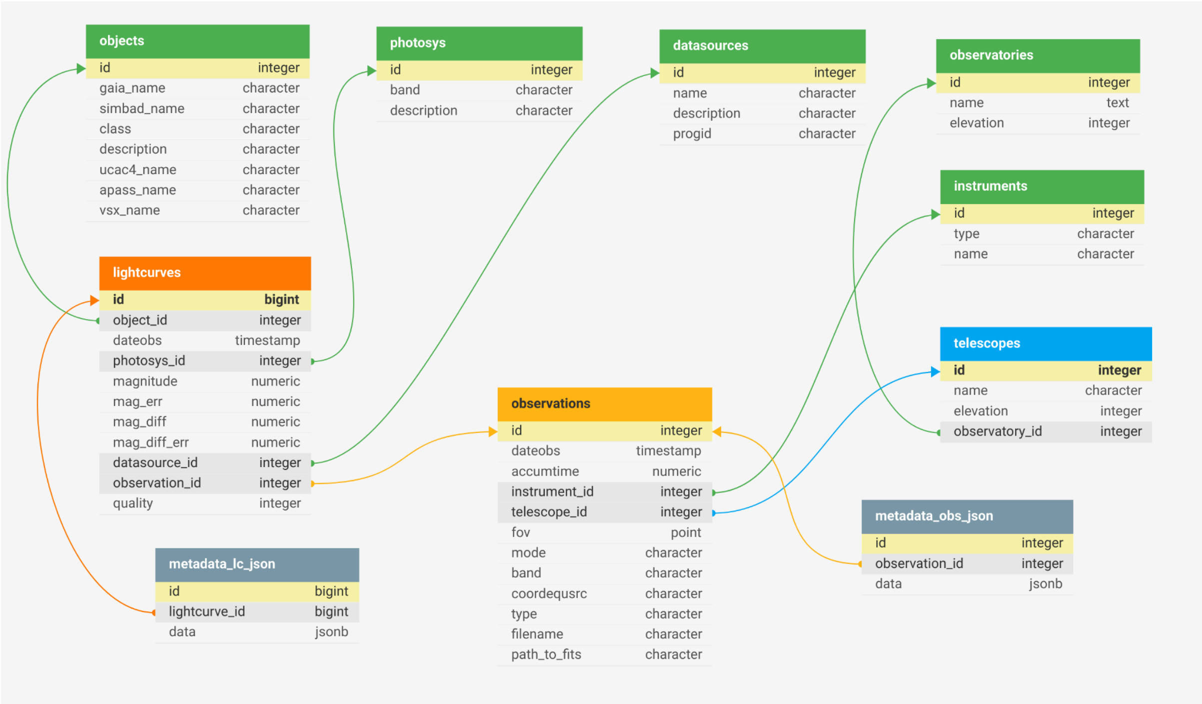
[Scripts overview](#)
  - [Tips for Working with Environment Variables](#)
  - [Scripts](#)
    - [Uploading Sequence](#)
    - [Deletion Sequence](#)

Database Overview

The PostgreSQL Database is designed to store observations made with small telescopes, primarily located in Kolonica. The database stores information on both, raw observations and the results of photometric processing. The main table containing the photometric processing results is the `lightcurves` table, while the base table for raw observations is the `observations` table.

Additional tables store information on observed objects, observation details, and the processing workflow.

Complete database scheme



Simplified relations scheme

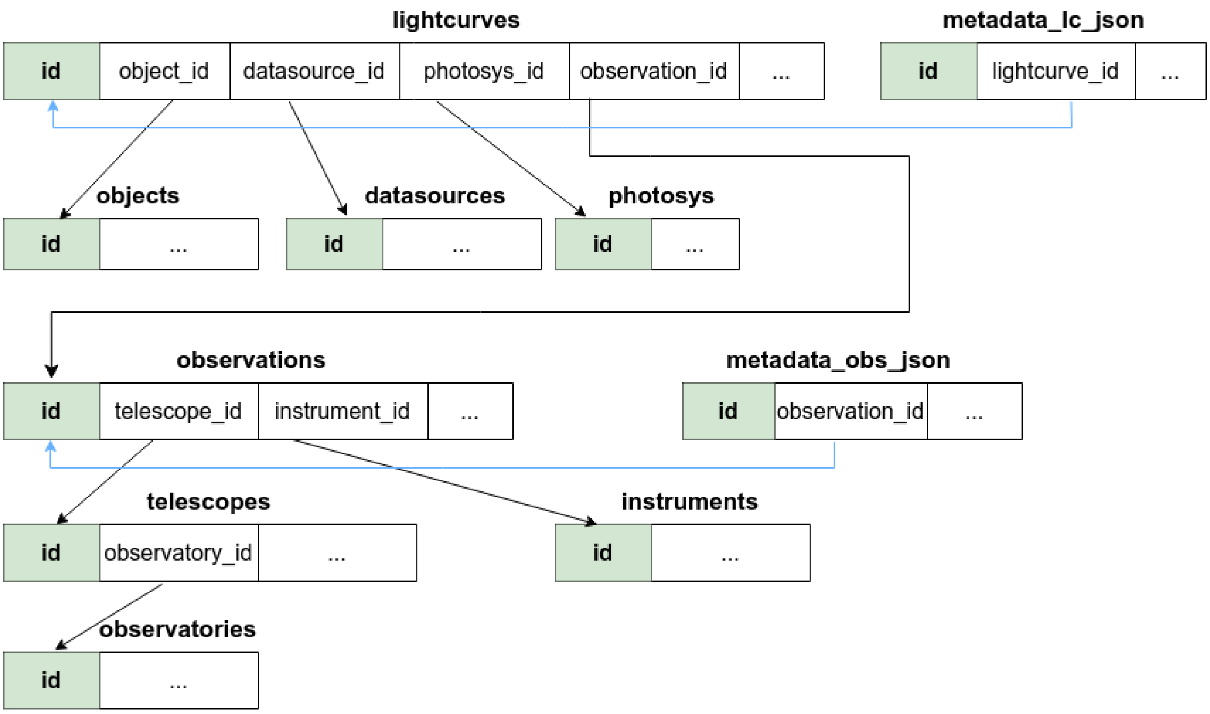


Table Overview

All tables can be divided into four categories: primary, descriptive, raw result and calculated results

Table name	category	What for
objects	primary	List of Objects for Calculated Data (e.g., lightcurves) (14,331 records)
datasources	primary	Contains copyright information, listing contributors responsible for observation programs, data processing, and publication
observatories	primary	Contains basic parameters of each observatory's location
instruments	primary	Contains parameters for the instruments used (e.g., CCDs)
photosys	primary	Describes the photometric system and bands used, related to the calculated data
telescopes	primary	Contains parameters of the telescopes used
observations	raw	Contains the basic parameters of the observations (e.g., related to CCD images) (8,442 records)
metadata_obs_json	descriptive	Contains additional information related to observations (e.g., FITS header for images), provided in JSON format
lightcurves	calculated	Contains all light curve points, relationally linked to the corresponding object, observation, etc. (1,466,177 records)
metadata_lc_json	descriptive	Contains additional details of the lightcurve calculation, provided in JSON format

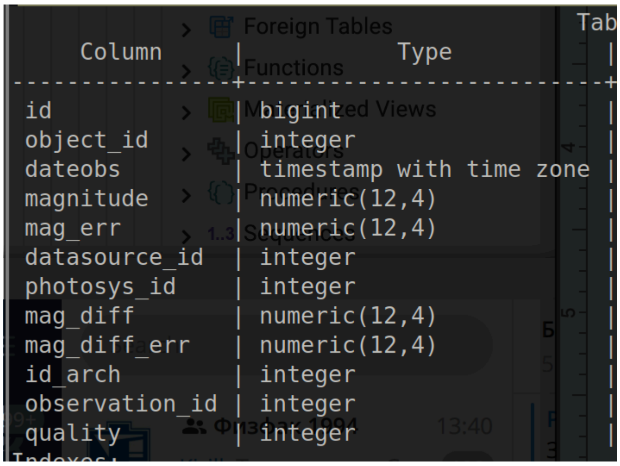
Table Details

The lightcurves table

id	object_id	dateobs	magnitude	mag_err	mag_diff	mag_diff_err	photosys_id

dataource_id	observation_id	quality	id_arch[1]

[1] Temporary column, to populate a table from the old database



```
     Column       |           Type
------------------+--------------------------
 id               | bigint
 object_id        | integer
 dateobs          | timestamp with time zone
 magnitude        | numeric(12,4)
 mag_err          | numeric(12,4)
 datasource_id    | integer
 photosys_id      | integer
 mag_diff         | numeric(12,4)
 mag_diff_err     | numeric(12,4)
 id_arch          | integer
 observation_id   | integer
 quality          | integer
```

The `lightcurves` is a narrow table with photometry. Each row contains one photometric point in one band

The basic user request

You can withdraw lightcurve using the following request:

```sql
SELECT
    EXTRACT(JULIAN FROM dateobs AT TIME ZONE 'UTC+12') AS jd,
    magnitude,
    mag_err,
    p.band
FROM
    lightcurves l
JOIN
    objects o ON l.object_id = o.id
JOIN
    photosys p ON l.photosys_id = p.id
WHERE
    o.simbad_name ILIKE 'TYC 4568-4%';
```

copy-paste it:

```
select extract(julian from dateobs at time zone 'UTC+12'), magnitude,
mag_err, p.band from lightcurves l join objects o on l.object_id = o.id
join photosys p on l.photosys_id = p.id where o.simbad_name ilike 'TYC
4568-4%';
```

```
              jd                | magnitude | mag_err |  band
--------------------------------+-----------+---------+--------
 2459510.3438700000000000000000 |   10.2437 |  0.1190 | i_sdss
```

or

```sql
SELECT
    dateobs AT TIME ZONE 'UTC' AS dateobs_utc,
    magnitude, mag_err, p.band
FROM
```

```
    lightcurves l
JOIN
    objects o ON l.object_id = o.id
JOIN
    photosys p ON l.photosys_id = p.id
WHERE
    o.simbad_name ILIKE 'TYC 4568-4%';
```

copy-paste it:

```
select dateobs at time zone 'UTC' as dateobs_utc, magnitude, mag_err,
p.band from lightcurves l join objects o on l.object_id = o.id join
photosys p on photosys_id = p.id where simbad_name ilike 'TYC 4568-4%';
```

```
        dateobs_utc          | magnitude | mag_err |  band
----------------------------+-----------+---------+--------
 2021-10-22 20:15:10.368    |   10.2437 |  0.1190 | i_sdss
```

Warning! Postgres interprets Julian Day in its own way, so to see the *astronomical Julian Day* we have to say:

vo=# … `EXTRACT(JULIAN FROM dateobs AT TIME ZONE 'UTC+12')` …

To write astronomical jd to a Postgres table in Python, say:

python> `dateobs_str = f'J{jd}-12'`

I don't know how Postgres handles leap second at the *leap-second-add-day*, so I think it would be safer to convert astronomical jd to datetime and vice versa using astropy.

The metadata_lc_json table

id	lightcurve_id	data

```
    Column     |  Type  |
---------------+--------+
 id            | bigint |
 lightcurve_id | bigint |
 data          | jsonb  |
```

`metadata_lc_json` table stores data related to the photometric processing in the JSON form.

```
SELECT jsonb_pretty(data) FROM metadata_lc_json ...;
 {                          +
     "binning": "2x2",      +
     "detrend": "0",        +
     "exp_time": 30.0,      +
```

```
     "num_comp": 2,            +
     "aver_comp": 7.5607,      +
     "num_calib": 42,          +
     "comp_stars": [           +
         "4944071931187818 24",+
         "4945910864386658 56" +
     ],                        +
     "privileges": "full",     +
     "type_calib": "0",        +
     "detrend_mag": 99.99,     +
     "temperature": -20.0,     +
     "differential": 1,        +
     "type_detrend": "None",   +
     "detrend_mag_err": 9.99   +
 }
```


Example request

For example, if we want to know how many photometric points were processed with the comparison star "1701925222477693952" we can write:

```
SELECT
    COUNT(lightcurve_id)
FROM
    metadata_lc_json
WHERE
    '"1701925222477693952"' <@ (data->'comp_stars');
```

copy-paste it:

```
select count(lightcurve_id) from metadata_lc_json where
'"1701925222477693952"' <@ (data->'comp_stars');

 count
-------
 84944
```

Note the quotes in this request. The comparison star name in json is a string (double quotes)

The observations table

Images (raw data)

id	dateobs	accumtime	telescope_id	instrument_id	fov	mode	band

	coordequ	coordequsrc	type	filename	path_to_fits

```
    Column      |           Type
----------------+--------------------------+
 id             | integer
 dateobs        | timestamp with time zone
 accumtime      | numeric(12,4)
 telescope_id   | integer
 instrument_id  | integer
 fov            | point
 mode           | character varying(8)
 band           | character varying(8)
 coordequ       | spoint
 coordequsrc    | character varying(16)
 type           | character varying(8)
 filename       | character varying(32)
 path_to_fits   | character varying(128)
```

The `observations` table contains basic information about the image.
The `coordequsrc` field (source of equatorial coordinates) may be ‘`wcs`’ or ‘`objectequ`’ (`header['OBJCTRA'], header['OBJCTDEC']`)
The `fov` field has a Postgres `point` format. It consists of two values: angular size_x and size_y in radians.
FOV was calculated this way:

```
python> deg_x(y) = header['NAXIS1(2)'] * scale_x(y)
python> f'\'({radians(deg_x)},{radians(deg_y)})\''
```

Here `scale_x, and scale_y` are wcs scales, if present in the fits header or `scale_x(y) = 206265 / header['FOCALLEN'] * header['X(Y)PIXSZ'] / 1000 / 3600,` if not.

```
SELECT * FROM observations LIMIT 1;

id | dateobs                 |accumtime|telescope_id|instrument_id
---+------------------------+---------+------------+---------------+
 2 | 2021-10-22 22:13:42+02 | 30.0000 |  3         | 2

|                 fov                       | mode   |  band
+-----------------------------------------+-------+--------+
 (0.024806042453147315,0.018733806927093 98)| light | r_sdss

| coordequ                            | coordequsrc | type   |
+-----------------------------------------+-------------+-------+
|(259.946337377513d , 76.6987591700176d) | wcs         | image

|filename                    |              path_to_fits
---------------------------+-------------------------------------+
| 2021-10-22T20:13:42_r.fit| /home/skvo/data/upjs/Alica/2021-10-22
```

The metadata_obs_json table

id	observation_id	data

```
     Column     |  Type   |
----------------+---------+
 id             | integer |
 observation_id | integer |
 data           | jsonb   |
Indexes:
```

`metadata_obs_json` table stores the fits-header in the JSON form.

```
SELECT jsonb_pretty(data) FROM metadata_obs_json WHERE observation_id = 100;
 {                                                      +
     "BITPIX": 16,                                      +
     "NAXIS": 2,                                        +
     "NAXIS1": 1679,                                    +
     "NAXIS2": 1268,                                    +
     "OBJECT": "VY UMi",                                +
     "RA": "17 20 39.00",                               +
     "DEC": "+76 42 40.0",                              +
     "UT": "22:01:23",                                  +
     "DATE": "22/10/21",                                +
     "BZERO": 32768.0,                                  +
     "CD1_1": -0.00084016812215,                        +
     "CD1_2": -0.0000253522262298,                      +
     "CD2_1": 0.0000253903567968,                       +
     "CD2_2": -0.000838906379862,                       +
     "FILTER": "g",                                     +
     ...
}
```

The objects table

id	coordequ	gaia_name	simbad_name	ucac4_name	apass_name	vsx_name	id_arch[2]

[2] Temporary column, to populate the lightcurves table from the old database

```
   Column    |         Type          | Collation | Nullable |
-------------+-----------------------+-----------+----------+
 id          | integer               |           | not null |
 coordequ    | spoint                |           |          |
 gaia_name   | character varying(32) |           | not null |
 simbad_name | character varying(32) |           |          |
 class       | character varying(32) |           |          |
 description | character varying(32) |           |          |
 ucac4_name  | character varying(32) |           |          |
 apass_name  | character varying(32) |           |          |
 vsx_name    | character varying(32) |           |          |
 id_arch     | integer               |           |          |
```

The `objects` table stores the basic information about known objects.
The `coordequ` field stores ICRS equatorial coordinates in the `spoint` format

```
SELECT
    coordequ,
    simbad_name,
    vsx_name
FROM
    objects
WHERE
    vsx_name ILIKE 'ss cyg';
```

copy-paste it:

```
select coordequ, simbad_name, vsx_name from objects where vsx_name ilike 'ss cyg';
                 coordequ                  | simbad_name | vsx_name
-------------------------------------------+-------------+----------
 (5.68417085460261 , 0.760722698432752) | V* SS Cyg   | SS Cyg
```

`coordequ` is measured in radians by default, but we can use degrees too:

```
SELECT set_sphere_output('DEG');
SELECT coordequ, simbad_name FROM objects WHERE vsx_name ILIKE 'ss cyg';

       coordequ         | simbad_name | vsx_name
------------------------+-------------+----------
 (325.679d , 43.5862d)  | V* SS Cyg   | SS Cyg
```

And vice versa, we can load coordinates in degrees to the table:

```
python> coordequ_str = f'\'({radec.icrs.ra.deg}d,
{radec.icrs.dec.deg}d)\''
```

Cone-search request

Request all objects inside a circle with a radius of 6’ around the point RA= 325.679 deg, DEC=43.5862 deg. The list will be sorted by the distance from this point:

```
SELECT
    gaia_name,
    simbad_name,
    coordequ <-> spoint('(325.679d, 43.5862d)') AS dist
FROM
    objects
WHERE
    coordequ <@ scircle('<(325.679d, 43.5862d), 0.1d>')
ORDER BY
    dist;
```

copy-paste it:

```
select gaia_name, simbad_name, coordequ <-> spoint '(325.679d, 43.5862d)'
as dist from objects where coordequ <@ scircle('<(325.679d,
43.5862d),0.1d>') order by dist;
```

```
    gaia_name        |       simbad_name        |          dist
--------------------+-------------------------+------------------------
1972957892448494592 | V* SS Cyg                |                      0
1972957308333789312 |                          | 0.00017514574622243208
1972957690592232192 |                          | 0.00022790859998570036
```

The current `observations` table was populated from fits headers, which were read walking through the directories /home/skvo/data/upjs/*

The Database access

Currently, there are three users:

- `postrgres` (admin)
- `writer_user` (can upload/remove data, but cannot change database or table structure)
- `readonly_user` (can only request data)

Warning!

- Please never save passwords directly in scripts, whether you post them on GitHub or not. Instead, store them in environment files (`.env`)
- Avoid using the `postrgres` account in your scripts
- Use `writer_user` when uploading data. If you're interested, I can show you how to drop tables in the database through the web application using my earlier script.

Scripts overview

Tips for Working with Environment Variables

$ cat .env

```
DB_HOST=localhost
DB_NAME=vo
DB_WRITER_USER=writer_user
DB_WRITER_PASS=some_writer_password
DB_READONLY_USER=readonly_user
DB_READONLY_PASS=some_readonly_password
```

```
$ pip install python-dotenv
```

```
from os import getenv
from dotenv import load_dotenv
# load variables from .env file:
load_dotenv()
...
conn = psycopg2.connect(
        host=getenv("DB_HOST"),
        dbname=getenv("DB_NAME"),
        user=getenv("DB_WRITER_USER"),
        password=getenv("DB_WRITER_PASS")
    )
```

Scripts

The Python scripts I used for the initial database population can be found here:
https://github.com/olgavoz1971/photometry_db.git
Usernames and passwords are stored in the .env file on the server, I'll show you where.

Uploading Sequence

Due to the relationships between the tables (because of foreign key constraints), it is important to follow this sequence when uploading data:

1. Upload observation and a new object (if needed);
2. Upload the lightcurve.

Deletion Sequence

When deleting data, please consider also the relationships between metadata and their associated tables. The deletion sequence should be:

1. Delete from `metadata_lc_json`
2. Delete from `lightcurves`
3. Delete from `metadata_obs_json`

4. Delete from `observations`
5. Delete from `objects` (after lightcurves)

We can discuss cascading deletion. If we find it useful, I'll modify the foreign key definitions.