BEAM Sun Observation Database

Documentation

October 24, 2025

Contents

Introduction

The **BEAM Sun Observation Database** main objectives are:

- To store and organize metadata for each observation.
- To manage the raw telescope measurement data from `.dat` files.
- To provide a structure for storing analysis results, such as calculated solar temperatures.

Currently, the database runs **locally** on PostgreSQL for testing and development. In the future, it can be deployed on a **remote server** to support multi-user access and integration into broader systems.

Database Connection Details

This database is hosted **locally** on the user's machine and uses a **PostgreSQL** server. Below are the connection parameters used by the Python scripts and pgAdmin:

Parameter	**Value**
Host	localhost
Port	5432
Database Name	beam
User	chris
Password	(empty)
Server Type	Local PostgreSQL Server

All scripts and database queries assume this configuration. If the database is on a remote server, the `host`, `port`, and `password` must be updated accordingly.

Schemas Overview

The database is divided into two main schemas for better organization and separation of responsibilities:

- **Observations Schema** Stores information related to telescope observations, including:
  - Metadata describing each observation.
  - Raw measurement data extracted from `.dat` files.
- **Analysis Schema** Stores processed data derived from the raw observations, such as sun temperature calculations and calibration parameters.

Observations Schema

The **Observations** schema contains two tables:

- **ObservationInfo**: Stores metadata about each observation.
- **ObservationData**: Stores the raw numeric data from the `.dat` files.

Role of `.dat` Files

Each telescope observation generates a corresponding `.dat` file containing raw numerical measurements. These files are not stored directly in the database but are kept on the local or remote server. The database only stores:

- The **filename** of the associated `.dat` file.
- The extracted **data points** from the file in the `ObservationData` table (one row per value; indices start at 0).

Table: ObservationInfo

This table stores detailed metadata for each recorded observation.

Column Name	Data Type	Description
obs_id	SERIAL (PK)	Unique ID of the observation.
obs_date	DATE	Date of the observation.
obs_start_time	TIME	Observation start time.
obs_end_time	TIME	Observation end time.
receiver	TEXT	Name/type of receiver used.
rest_freq_MHz	FLOAT	Rest frequency in MHz.
bandwidth_MHz	FLOAT	Bandwidth used in MHz.
ra_deg	FLOAT	Right Ascension in degrees.
dec_deg	FLOAT	Declination in degrees.
integration_time_s	FLOAT	Integration time per sample in seconds.

observation_type	TEXT	Type of observation (`Cold`, `Hot`, `Sun`, etc.).
object	TEXT	Name of the observed object.
rf_gain	FLOAT	RF amplifier gain.
if_gain	FLOAT	IF amplifier gain.
bb_gain	FLOAT	Baseband amplifier gain.
dat_filename	TEXT	Name of the corresponding `.dat` file.

Note on identifiers and case: PostgreSQL is case-insensitive for unquoted identifiers. If a column is created with quotes and mixed case (e.g., `"Object"`), it must be quoted exactly in SQL. The scripts below use quoted identifiers like `"Object"`, `"rest_freq_MHz"`, etc.

Table: ObservationData

This table stores the raw data values for each observation, extracted from the corresponding `.dat` files.

Column Name	Data Type	Description
data_id	SERIAL (PK)	Unique ID for each data point.
obs_id	INTEGER (FK)	Links to `ObservationInfo.obs_id`.
value_index	INTEGER	Index of the data point within the `.dat` file (0-based).
data_value	FLOAT	Numeric value from the `.dat` file.

Analysis Schema

The **Analysis** schema contains tables derived from observations, used for calibration and Sun temperature analysis.

Table: SunTemperatureAnalysis

This table stores information about Sun temperature calculations derived from specific observations.

Column Name	Data Type	Description
analysis_id	SERIAL (PK)	Unique ID for each analysis entry.
cold_obs_id	INTEGER (FK)	Links to cold observation in `ObservationInfo`.
hot_obs_id	INTEGER (FK)	Links to hot observation in `ObservationInfo`.
sun_obs_id	INTEGER (FK)	Links to Sun observation in `ObservationInfo`.
c_start	INTEGER	Masking start index for cold data.

h_start	INTEGER	Masking start index for hot data.
s_start	INTEGER	Masking start index for Sun data.
final_sun_temp_K	FLOAT	Calculated Sun temperature in Kelvin.

Relations Between Tables

- Each **Observation** in `ObservationInfo` can have **multiple** associated data points in `ObservationData` (One-to-Many).
- Each row in `SunTemperatureAnalysis` references **three type of measurements**: **cold**, **hot**, and **sun**.
- The same observation in `ObservationInfo` can be reused in **multiple** analysis calculations (Many-to-One).
- Relationships overview:
  - `ObservationData.obs_id` → `ObservationInfo.obs_id` **(One Observation → Many Data Points)**
  - `SunTemperatureAnalysis.cold_obs_id` → `ObservationInfo.obs_id` **(Many Analyses → One Observation)**
  - `SunTemperatureAnalysis.hot_obs_id` → `ObservationInfo.obs_id` **(Many Analyses → One Observation)**
  - `SunTemperatureAnalysis.sun_obs_id` → `ObservationInfo.obs_id` **(Many Analyses → One Observation)**

Data Ingestion & Update Scripts

This section is for the two Python scripts used to **insert new observations** and to **load raw `.dat` files** into the database.

Script A: Insert Metadata into `ObservationInfo`

Purpose. Parse a metadata text file and insert a new row into `"Observations"."ObservationInfo"`.

Input.

- A text file (e.g., `observation_metadata_2025-08-07_19-21-33.txt`) containing key–value pairs such as `Observation_Start`, `Observation_End`, `Receiver`, etc.

Key Details.

- Datetime strings are parsed with the format `%Y-%m-%d_%H-%M-%S`.
- Several identifiers are quoted (e.g., `"Object"`, `"rest_freq_MHz"`) to match case-sensitive column names.

Usage. Set `TXT_FILE` to the target metadata file and run the script.

Listing 1: Insert observation metadata into ObservationInfo

```
# import libraries
# psycopg2 allows to work with PostgreSQL
import psycopg2
from datetime import datetime

# DB configuration
DB_NAME = "beam"
DB_USER = "chris"
DB_PASSWORD = ""
DB_HOST = "localhost"
DB_PORT = "5432"
TABLE_NAME = '"Observations"."ObservationInfo"'

# txt file with observation
TXT_FILE = "/Users/chris/Desktop/BEAM/Database/
   observation_metadata_2025-08-07_19-21-33.txt"

# parse text file to the data dictionary
def parse_metadata(filepath):
    data = {}
    with open(filepath, "r") as file:
        for line in file:
            if ":" in line:
                key, value = line.strip().split(":", 1)
                data[key.strip()] = value.strip().strip('"')

    # handle datetime parsing
    obs_start_str = data.get("Observation_Start")
    obs_end_str = data.get("Observation_End")
```

```
29
30      obs_start_dt = datetime.strptime(obs_start_str, "%Y-%m-%d_%
            H-%M-%S")
31      obs_end_dt = datetime.strptime(obs_end_str, "%Y-%m-%d_%H-%M
            -%S")
32
33      return {
34          "obs_date": obs_start_dt.date(),
35          "obs_start_time": obs_start_dt.time(),
36          "obs_end_time": obs_end_dt.time(),
37          "receiver": data.get("Receiver"),
38          "rest_freq_MHz": float(data.get("rest_freq_MHz")),
39          "bandwidth_MHz": float(data.get("Bandwidth_MHz")),
40          "ra_deg": float(data.get("ra_deg")),
41          "dec_deg": float(data.get("dec_deg")),
42          "integration_time_s": float(data.get("
                integration_time_s")),
43          "observation_type": data.get("observation_type"),
44          "Object": data.get("Object"),
45          "rf_gain": float(data.get("rf_gain")),
46          "if_gain": float(data.get("if_gain")),
47          "bb_gain": float(data.get("bb_gain")),
48      }
49
50  # insert the new row(observation) to the DB
51  def insert_into_db(metadata):
52      try:
53          conn = psycopg2.connect(
54              dbname=DB_NAME,
55              user=DB_USER,
56              password=DB_PASSWORD,
57              host=DB_HOST,
58              port=DB_PORT
59          )
60          cursor = conn.cursor()
61
62          sql = f"""
63              INSERT INTO {TABLE_NAME} (
64                  obs_date, obs_start_time, obs_end_time,
65                  receiver, "rest_freq_MHz", "bandwidth_MHz",
66                  ra_deg, dec_deg, integration_time_s,
67                  observation_type, "Object", rf_gain, if_gain,
                        bb_gain
68              )
69              VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
                    %s, %s, %s)
70          """
71
72          values = (
73              metadata["obs_date"],
74              metadata["obs_start_time"],
```

```
            metadata["obs_end_time"],
            metadata["receiver"],
            metadata["rest_freq_MHz"],
            metadata["bandwidth_MHz"],
            metadata["ra_deg"],
            metadata["dec_deg"],
            metadata["integration_time_s"],
            metadata["observation_type"],
            metadata["Object"],
            metadata["rf_gain"],
            metadata["if_gain"],
            metadata["bb_gain"]
        )

        cursor.execute(sql, values)
        conn.commit()
        cursor.close()
        conn.close()
        print("Observation inserted successfully.")
    except Exception as e:
        print("Error inserting data:", e)

# RUN
if __name__ == "__main__":
    metadata = parse_metadata(TXT_FILE)
    insert_into_db(metadata)
```

Output. A new row in `ObservationInfo`. The generated `obs_id` can be used to link raw data in the next step.

Script B: Load Raw `.dat` Samples into `ObservationData`

Purpose. Read a binary `.dat` file as `float32` and insert each sample as a row in `"Observations"."ObservationData"` for a given `obs_id`.

Input.

- `dat_file_path`: absolute path to the `.dat` file.
- `associated_obs_id`: the `ObservationInfo.obs_id` that these samples belong to.

Usage. Set the two variables above and run the script.

Listing 2: Insert .dat samples into ObservationData

```
import psycopg2
import numpy as np
import os

# DB configuration
DB_NAME = "beam"
DB_USER = "chris"
DB_PASSWORD = ""
DB_HOST = "localhost"
```

```
DB_PORT = "5432"
TABLE_NAME = '"Observations"."ObservationData"'

# data file path and id of ObservationInfo DB
dat_file_path = "/Users/chris/Desktop/BEAM/Observations/Hot.dat"  # path to .dat file
associated_obs_id = 8  # changes based on observation

# load .dat data(binary)
try:
    data = np.fromfile(dat_file_path, dtype=np.float32)
    print(f"Loaded {len(data)} values from: {os.path.basename(dat_file_path)}")
except Exception as e:
    print("Error loading .dat file:", e)
    exit()

# insert to DB
try:
    conn = psycopg2.connect(
        dbname=DB_NAME,
        user=DB_USER,
        password=DB_PASSWORD,
        host=DB_HOST,
        port=DB_PORT
    )
    cursor = conn.cursor()

    sql = f"""
        INSERT INTO {TABLE_NAME} (obs_id, value_index, data_value)
        VALUES (%s, %s, %s)
    """

    records = [(associated_obs_id, i, float(val)) for i, val in enumerate(data)]
    cursor.executemany(sql, records)

    conn.commit()
    cursor.close()
    conn.close()

    print("Inserted all data points successfully into the database.")

except Exception as e:
    print("Error inserting into database:", e)
```

Output. `len(data)-1` rows added to `ObservationData`.