SQL for STEM and Astronomy Research

SQL databases can provide significant benefits for astronomy STEM researchers, especially when dealing with large datasets, cross-referencing astronomical catalogs, or performing complex queries. Here are some key advantages:

1. Efficient Data Management

- **Structured Storage:** SQL databases allow researchers to store structured data in tables, making it easy to retrieve and organize observations, metadata, and research results.
- **Scalability:** Many SQL databases (e.g., PostgreSQL, MySQL, SQLite) can handle large datasets, and some (e.g., PostgreSQL with TimescaleDB) support time-series data, useful for astronomical observations.

2. Advanced Query Capabilities

- **Complex Queries:** SQL supports powerful queries for filtering, aggregating, and analyzing data efficiently.
- **Indexing & Optimization:** Indexing speeds up data retrieval, which is useful when working with vast astronomical datasets like Gaia, Sloan Digital Sky Survey (SDSS), or NASA Exoplanet Archive.

3. Integration with Other Tools

- **Python & Data Science Tools:** SQL databases integrate well with Python (pandas, SQLAlchemy), R, and other scientific computing tools.
- **GIS & Spatial Data Support:** PostgreSQL with PostGIS provides support for spatial queries, useful for sky surveys and celestial object mapping.

4. Data Integrity & Security

- **ACID Compliance:** Ensures consistency and reliability in large collaborative research projects.
- **Access Control:** SQL databases allow user permissions, ensuring secure multi-user collaboration.

5. Handling Large-Scale Astronomy Data

- **Cloud & Distributed Databases:** SQL databases like Google BigQuery (used for analyzing astronomy datasets) can handle petabyte-scale data efficiently.

- **Parallel Processing:** Some SQL engines (e.g., Amazon Redshift, PostgreSQL with parallel query execution) optimize performance for large queries.

6. Examples of Astronomy Use Cases

- Storing and querying data from **telescope observations** (e.g., timestamps, celestial coordinates, spectra).
- Analyzing **light curves** and **time-series** data for variable stars and exoplanets.
- Cross-matching data from multiple astronomical catalogs to find **correlations** between celestial objects.

Alternatives to SQL

- **NoSQL Databases (e.g., MongoDB, Cassandra)**: Useful for unstructured data like raw telescope images, but SQL is generally better for structured numerical datasets.
- **HDF5 / FITS Files**: These are standard formats for storing large-scale astronomical data, but they lack SQL's querying power.

For astronomy researchers handling structured data like object catalogs, time-series data, and relational datasets, **SQL databases provide powerful tools for querying, analyzing, and managing data efficiently**. Combining SQL with other tools (e.g., Python, HDF5, NoSQL for images) creates a robust data ecosystem for research.

Use Cases of SQL Databases in Astronomy

1. **Storing and Querying Celestial Object Data**
    - Example: A database containing stars, galaxies, and exoplanets with their properties (coordinates, brightness, spectral type, etc.).
    - Use: Retrieve all objects in a specific region of the sky, filter by magnitude, or find objects of a certain spectral type.
2. **Analyzing Time-Series Data (Light Curves, Observations Over Time)**
    - Example: Data from telescopes monitoring a variable star or exoplanet transit.
    - Use: Query time-series data to detect periodic variations in brightness.
3. **Cross-Matching Astronomical Catalogs**
    - Example: Combining datasets from Gaia, SDSS, and Kepler to find new patterns.

- Use: Match objects based on coordinates and properties.

4. **Galaxy Morphology Classification**
   - Example: A database storing galaxy images along with features like shape, size, and classification.
   - Use: Query and filter galaxies based on their classification and study correlations.
5. **Astronomical Surveys and Big Data Queries**
   - Example: A researcher needs to analyze millions of stars in a large sky survey dataset.
   - Use: SQL enables fast filtering and aggregating of huge datasets.

Example Dataset in Python (Using SQLite)

Let's create a simple SQLite database storing information about stars in a sky survey.

Step 1: Create the Database and Table

```
import sqlite3

import pandas as pd


# Create a SQLite database

conn = sqlite3.connect("astronomy.db")

cursor = conn.cursor()


# Create a table for stars

cursor.execute('''

  CREATE TABLE IF NOT EXISTS stars (

    id INTEGER PRIMARY KEY,

    name TEXT,
```

```
        ra FLOAT,   -- Right Ascension (degrees)
        dec FLOAT,  -- Declination (degrees)
        magnitude FLOAT,  -- Apparent Magnitude
        spectral_type TEXT
    )
''')
conn.commit()
```

Step 2: Insert Sample Data

```
data = [
    (1, "Alpha Centauri", 219.9, -60.8, 0.0, "G2V"),
    (2, "Betelgeuse", 88.8, 7.4, 0.42, "M1Ia"),
    (3, "Sirius", 101.3, -16.7, -1.46, "A1V"),
    (4, "Vega", 279.2, 38.8, 0.03, "A0V"),
    (5, "Antares", 247.3, -26.4, 1.06, "M1.5Ib")
]

cursor.executemany("INSERT INTO stars VALUES (?, ?, ?, ?, ?, ?)", data)
conn.commit()
```

Step 3: Query Data Using Pandas

```
# Read data into a Pandas DataFrame
df = pd.read_sql_query("SELECT * FROM stars", conn)
print(df)
```

Step 4: Filtering Stars by Magnitude

```
# Get all bright stars (magnitude < 1)
bright_stars = pd.read_sql_query("SELECT * FROM stars WHERE magnitude < 1", conn)
print(bright_stars)
```

SQLite is useful for **small to medium datasets**, especially for local research or prototyping. However:

- It **does not scale well** for large-scale astronomical datasets.
- It **lacks parallel query execution** and advanced indexing compared to PostgreSQL.

For **big data astronomy projects**, **PostgreSQL** (with **PostGIS** for spatial queries) or cloud-based solutions (Google BigQuery, Amazon Redshift) would be better.

What About NoSQL?

🚀 **When to Use NoSQL in Astronomy**

- **Handling Unstructured Data:** NoSQL is useful for **storing raw telescope images, JSON metadata, or logs**.
- **Scalability:** MongoDB or Cassandra can handle distributed data across multiple nodes, useful for massive sky surveys.
- **Flexible Schema:** If the data structure varies (e.g., different telescope instruments storing different attributes), NoSQL avoids rigid table structures.

📌 **Example NoSQL Use Case in Astronomy**

- **Storing raw image metadata from telescopes** (MongoDB)
- **Managing logs from astronomical simulations** (Cassandra)
- **Storing heterogeneous data (text, metadata, sensor data)** (MongoDB)

🔷 **SQL vs NoSQL for Astronomy**

Feature	SQL (PostgreSQL, SQLite)	NoSQL (MongoDB, Cassandra)
Structured Data	✅ Excellent	❌ Not ideal
Complex Queries	✅ Excellent (JOINs, aggregations)	⚠️ Limited (No JOINs)

Scalability	⚠️ Moderate (better with PostgreSQL)	✅ Excellent
Handling Big Data	⚠️ OK for structured, but limited for unstructured	✅ Great for massive datasets
Use Case Example	Star catalogs, time-series data	Telescope images, logs

---

Which One Should You Use?

- **Use SQL (SQLite, PostgreSQL) if your research involves structured data like catalogs, light curves, or cross-matching databases.**
- **Use NoSQL (MongoDB, Cassandra) if you deal with unstructured data like telescope images, logs, or highly dynamic datasets.**
- **For massive datasets (terabytes/petabytes), consider PostgreSQL with optimizations or cloud solutions like Google BigQuery.**

Multi-Table Example in SQLite for Astronomy Research

A real-world astronomy database often consists of multiple related tables. For example, we can store **stars**, their **observations**, and the **telescopes** used for observations in separate tables.

📌 Database Schema

We will create three related tables:

1. **Stars**: Stores basic information about stars.
2. **Telescopes**: Stores details about different telescopes.
3. **Observations**: Stores observation data (linked to both stars and telescopes).

Relationships:

- Each observation is linked to **one star** and **one telescope**.
- A star can have multiple observations.
- A telescope can be used for multiple observations.

Step 1: Create Tables in SQLite

```python
import sqlite3

import pandas as pd


# Create a new SQLite database

conn = sqlite3.connect("astronomy_multitable.db")

cursor = conn.cursor()


# Create the 'stars' table

cursor.execute('''

  CREATE TABLE IF NOT EXISTS stars (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT,

    ra FLOAT,   -- Right Ascension (degrees)

    dec FLOAT,  -- Declination (degrees)

    spectral_type TEXT

  )

''')


# Create the 'telescopes' table

cursor.execute('''

  CREATE TABLE IF NOT EXISTS telescopes (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT,

    location TEXT,
```

```
        aperture FLOAT  -- Diameter of the telescope in meters
    )
''')

# Create the 'observations' table (linked to both stars and telescopes)
cursor.execute('''
    CREATE TABLE IF NOT EXISTS observations (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        star_id INTEGER,
        telescope_id INTEGER,
        observation_date TEXT,
        magnitude FLOAT,
        FOREIGN KEY (star_id) REFERENCES stars(id),
        FOREIGN KEY (telescope_id) REFERENCES telescopes(id)
    )
''')

conn.commit()
```

Step 2: Insert Sample Data

```
# Insert sample stars
stars_data = [
    ("Alpha Centauri", 219.9, -60.8, "G2V"),
    ("Betelgeuse", 88.8, 7.4, "M1Ia"),
    ("Sirius", 101.3, -16.7, "A1V"),
    ("Vega", 279.2, 38.8, "A0V")
```

```
]

cursor.executemany("INSERT INTO stars (name, ra, dec, spectral_type) VALUES (?, ?, ?, ?)",
stars_data)

conn.commit()


# Insert sample telescopes

telescopes_data = [

    ("Hubble Space Telescope", "Orbit", 2.4),

    ("Keck Observatory", "Hawaii", 10.0),

    ("VLT", "Chile", 8.2)

]


cursor.executemany("INSERT INTO telescopes (name, location, aperture) VALUES (?, ?, ?)",
telescopes_data)

conn.commit()


# Insert sample observations (star_id and telescope_id are foreign keys)

observations_data = [

    (1, 1, "2023-01-10", 0.01),  # Alpha Centauri observed by Hubble

    (2, 2, "2023-02-15", 0.45),  # Betelgeuse observed by Keck

    (3, 3, "2023-03-20", -1.46), # Sirius observed by VLT

    (4, 1, "2023-04-05", 0.03),  # Vega observed by Hubble

    (2, 3, "2023-05-12", 0.5)    # Betelgeuse observed by VLT

]
```

```
cursor.executemany("INSERT INTO observations (star_id, telescope_id, observation_date, magnitude) VALUES (?, ?, ?, ?)", observations_data)

conn.commit()
```

Step 3: Querying the Data Using Joins

Now that we have data in multiple tables, we can perform **SQL JOINs** to analyze it.

Get All Observations With Star and Telescope Names

```
query = '''
  SELECT
    o.observation_date,
    s.name AS star_name,
    s.spectral_type,
    t.name AS telescope_name,
    o.magnitude
  FROM observations o
  JOIN stars s ON o.star_id = s.id
  JOIN telescopes t ON o.telescope_id = t.id
  ORDER BY o.observation_date
'''

df = pd.read_sql_query(query, conn)
print(df)
```

Find All Observations Made by the Hubble Telescope

```
query = '''
  SELECT
```

```
        s.name AS star_name,
        o.observation_date,
        o.magnitude
    FROM observations o
    JOIN stars s ON o.star_id = s.id
    JOIN telescopes t ON o.telescope_id = t.id
    WHERE t.name = "Hubble Space Telescope"
'''

df = pd.read_sql_query(query, conn)
print(df)
```

Find the Brightest Stars Observed (Magnitude < 0.1)

```
query = '''
    SELECT
        s.name AS star_name,
        t.name AS telescope_name,
        o.observation_date,
        o.magnitude
    FROM observations o
    JOIN stars s ON o.star_id = s.id
    JOIN telescopes t ON o.telescope_id = t.id
    WHERE o.magnitude < 0.1
'''

df = pd.read_sql_query(query, conn)
```

```
print(df)
```

Conclusion: Why Use Multi-Table SQL for Astronomy?

✅ **Normalization:** Avoids redundancy by splitting data into separate tables.
✅ **Efficient Queries:** Faster and more structured than storing all data in one table.
✅ **Scalability:** Works for small SQLite databases and large PostgreSQL setups.

💡 Next Steps

1. **Spatial Queries (PostGIS)?** → Useful for finding nearby celestial objects.
2. **NoSQL Version (MongoDB)?** → Useful for unstructured metadata (images, telescope logs).

Converting FITS Files to SQL Database for Astronomy Research

FITS (**Flexible Image Transport System**) files are the standard format for astronomical data, used for storing images, spectra, and tables. Researchers often convert FITS files to **SQL databases** for structured querying and analysis.

🔷 Why Convert FITS to SQL?

✅ **Structured Queries**: Retrieve specific rows/columns efficiently.
✅ **Multi-table Support**: Link data with stars, telescopes, observations.
✅ **Large Dataset Handling**: FITS can be large; databases optimize access.
✅ **Indexing**: Faster lookups compared to raw FITS files.

📌 Step 1: Install Dependencies

```
pip install astropy pandas sqlite3
```

- **Astropy**: Handles FITS files.
- **Pandas**: Converts tables into structured data.

- **SQLite3**: Stores data efficiently.

📌 Step 2: Load FITS File & Inspect Data

```
from astropy.io import fits

# Open a FITS file (Example: Gaia star catalog)
fits_file = "example.fits"

# Read the FITS data
with fits.open(fits_file) as hdul:
    hdul.info()  # Show FITS structure
    data = hdul[1].data  # Load table from first extension

# Print first few rows
print(data[:5])
```

- hdul.info() shows FITS extensions (headers, tables, images).
- data = hdul[1].data extracts table format (stars, spectra, etc.).

📌 Step 3: Convert FITS Table to Pandas DataFrame

```
import pandas as pd

# Convert FITS table to Pandas DataFrame
df = pd.DataFrame(data)

# Show first few rows
print(df.head())
```

- Pandas provides **easy filtering, joins, and analysis**.
- Check column names: print(df.columns).

📌 Step 4: Store Data in SQLite Database

```
import sqlite3

# Connect to SQLite
conn = sqlite3.connect("astronomy.db")
cursor = conn.cursor()

# Create a table based on FITS data structure
cursor.execute('''
  CREATE TABLE IF NOT EXISTS stars (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    ra FLOAT,   -- Right Ascension (degrees)
    dec FLOAT,  -- Declination (degrees)
    mag FLOAT,  -- Magnitude
    parallax FLOAT
  )
''')

# Insert FITS data into SQLite
df[['ra', 'dec', 'mag', 'parallax']].to_sql("stars", conn, if_exists="append", index=False)

print("Data inserted successfully into SQLite database.")
```

📌 Step 5: Query Data Using SQL

```
query = "SELECT * FROM stars WHERE mag < 10 ORDER BY mag ASC"

df_query = pd.read_sql_query(query, conn)

print(df_query.head())
```

📌 Step 6: Multi-Table Example (Stars + Observations)

We can create **linked tables** to store telescope observations.

```
# Create an observations table linked to stars

cursor.execute('''

  CREATE TABLE IF NOT EXISTS observations (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    star_id INTEGER,

    telescope TEXT,

    observation_date TEXT,

    magnitude FLOAT,

    FOREIGN KEY (star_id) REFERENCES stars(id)

  )

''')


# Insert example observation

cursor.execute("INSERT INTO observations (star_id, telescope, observation_date, magnitude) VALUES (1, 'Hubble', '2023-01-10', 0.01)")

conn.commit()
```

📌 Step 7: Query Data Across Tables

```
query = '''
```

```
    SELECT s.ra, s.dec, s.mag, o.telescope, o.observation_date
    FROM stars s
    JOIN observations o ON s.id = o.star_id
    WHERE s.mag < 5
'''
df_query = pd.read_sql_query(query, conn)
print(df_query)
```

💡 Alternative: Store FITS in NoSQL (MongoDB)

If your FITS files contain **unstructured metadata** (e.g., telescope logs, images), NoSQL databases like **MongoDB** may be better.

```
from pymongo import MongoClient


client = MongoClient("mongodb://localhost:27017/")
db = client["astronomy"]
collection = db["stars"]


# Convert DataFrame to JSON and insert into MongoDB
collection.insert_many(df.to_dict("records"))
```

📌 **MongoDB Pros:** Good for images, metadata, and nested structures.

🚀 Conclusion

✅ SQLite is great for **structured tabular data** (stars, observations).
✅ NoSQL is useful for **unstructured metadata** (logs, images).
✅ FITS-to-SQL allows **faster queries, joins, and large dataset handling**.