HOW TO ACCESS DATA ON ALL 1.8 BILLION STARS IN THE GAIA CATALOG

Mark Sproul and Keith Sproul

INTRODUCTION

The European Space Agency developed the Gaia space observatory[1] to create the largest, most precise three-dimensional map of the Milky Way by surveying about 1 percent of the galaxy's 100 billion stars. It was launched in December 2013 and became operational in July 2014. Gaia collected data on almost 2 billion stars: position, brightness, proper motion, and more. The first release of data, information on about 1 billion stars, was in September 2016. The second release of data, 1.6 billion stars, was in April 2018. The current release (EDR3) came in December 2020, and contains information on 1.8 billion stars.

Many of us use planetarium programs to learn the night sky or control a telescope for imaging or observing. Many such programs are available, some commercial, some free and open-source, on all platforms - Windows, Mac, Linux, and cell phones.

All planetarium programs support common star catalogs, such as Yale Bright Star (9,000 stars), Hipparcos (118,000), and Henry Draper (272,000). Today's computers can easily store this much data in random-access memory. It is desirable to keep all of the data in memory for speed. However, the Gaia star catalog is over 6,000 times the size of the Draper catalog. Fitting all that into memory would be tough. Therefore, although we would like to have the entire Gaia catalog at our fingertips, storing it in memory won't work.

Our software should provide quick access to all of the stars. However, we do not need or want all of the stars in memory at the same time, so only storing in memory the area we are currently looking at is acceptable, and desirable for speed. We also require the ability to search for a particular star by name, and to provide this with as little user interaction as possible. In other words, it should all happen "auto-magically." The solution is to put the data into a database that can be accessed from the planetarium software. However, it is still a daunting project if you don't know where to begin. Note that all of the software described below is open source and available on GitHub.

According to John Hoot in his article in the *2020 Symposium* of the Society for Astronomical Sciences, "Gaia Data Enables New Methods of Analysis and Discovery,"[2]

> It is possible with enough disk space and bandwidth to own your own copy of (Gaia) DR2. The separate tables can be downloaded directly from the ESA at *gea.esac.esa.int/archive*, or from other mirror sites. But once you get the data, you need to be somewhat of a database expert to load all the tables into a relational database, generate the right queries and cross link all these tables on the appropriate fields in order to tease insights out of this mountain of data.

Thankfully, we don't want or need all of that huge amount of data. We just want to be able to display these stars in a planetarium program. Most modern databases on the internet run using some variant of Structured Query Language (SQL). There are many SQL database programs, some commercial, and many free and open source, on all popular platforms. In order to limit the amount of data in each query, we want to be able to ask the database for all of the stars in any 1 by 1 degree block.

IMPLEMENTATION

The computer we are using is an older 2014 Dell desktop we had sitting around. We installed more memory, and the most important thing we did was install a 2-terabyte solid-state drive (SSD) instead of the older-style spinning-platter hard drive. SSDs are as much as 100 to 500 times faster than spinning-platter drives. If you really want to bump up the speed, get a commercial-grade NVMe SSD.

We are running Ubuntu Linux version 20 with either MariaDB or MySQL for the database engine. These are almost identical, and both have run this successfully. The code supplied in this project has successfully run on both MariaDB and MySQL. We recommend MariaDB. The entire Linux operating system can be installed from scratch in less than an hour.[3] Installing the SQL database server might take another hour.[4]

Then there is the database itself. We are currently using Gaia database EDR3 (Early Data Release 3).[5] This database consists of 3,386 compressed files totaling 660 gigabytes. Downloading these files alone can take hours even on a high-speed network connection. Each compressed file is over 180 megabytes. When uncompressed, they are over 400 megabytes each. Once these are all uncompressed, it is a whopping 1.5 terabytes of data.

Each of these 3,386 files has over 500,000 records and each record has 99 fields of data. We are only interested in a few of these fields.

Designation	Text string with source and index number
source_id	Star Index Number (integer)
ra	Right Ascension
dec	Declination
phot_g_mean_mag	Magnitude
bp_rp	Blue/red color information

All but the first two of these fields are floating-point numbers. Searching and indexing floating-point numbers is very slow; it is much faster to search on integers. Therefore, we added a couple fields to make indexing much faster. We take the integer portion of right ascension and declination and use those integer fields for indexing and searching. We use the FLOOR() function rather than the INT() function, because using INT(), both -0.5 and +0.5, for example, end up as zero. It is important to note that the integer declination goes from -90 to +89. The final database structure looks like this:

Field	Type
designation	char(32)
source_id	bigint
ra	double
decl	double
phot_g_mean_mag	double
bp_rp	double
ra_int	int
dec_int	int

The newly generated SQL files are now only 80 gigabytes compressed, smaller than the original Gaia files, which were 660 gigabytes compressed. These compressed SQL files are available on our GitHub site.[6]

DATABASE FEATURES

SQL is a fairly simple language for what we are doing. But like any programming language, it can be hard to understand until you learn it. This entire system is set up so that a non-programmer can do it. We have simplified the searches for a couple of reasons. The first is simplicity, which also buys us speed. But the other reason is that doing it this way allows tighter security.

The SQL queries that we have written could be used by anyone wanting to incorporate the Gaia data into their software, not just the software we are writing (described below).

The routines are:

GetRaDec(raInt, decInt); Returns all star data in the specified 1 by 1 degree grid.

GetDeclination(decInt); Returns all star data in that declination. Only for use near the poles, that is, poleward of 84 degrees declination.

GetStarFromID(source_id); Returns the star data with the given source ID (integer).

GetVersion; Returns the current Gaia version number (currently "Gaia EDR3").

These routines can be called from any language with an SQL library. We use C/C++, but it would be easy to do from Python or another programming language with remote SQL capabilities. Additional routines can be written easily as needs change.

SECURITY

As with any computer endeavor, security is a concern, and we have done a fair amount in that regard. SQL servers allow you to limit what accounts can and cannot do. We set up the account for this to be read-only, and only able to execute predefined scripts in the SQL system. It can't issue SQL commands directly. It can't even look at the raw data. It can only do what it is supposed to be doing. This prevents SQL insertion hacks.

If you are only using it in-house, behind a firewall, you don't have to worry about much, but if you open your SQL server to the outside, you need to pay a lot more attention to security. We have the server on the standard SQL port, and it does get attacked, especially from IP addresses from overseas. These can be restricted using IPTABLES security settings on the Linux server. On Windows or Mac, use the built-in firewall settings. You also need to lock down the admin accounts on the SQL server. However, these security measures are the default when you install the MariaDB or MySQL software.

More detailed information on the security implementation is on the GitHub site below.

PLANETARIUM SIDE

Mark has been writing a planetarium program called SkyTravel in conjunction with my AlpacaPi observatory control project.[7] This planetarium program was modified to be able to request the data from the SQL server. The logic is as follows:

If Gaia is enabled, and if the screen field of view is 5 degrees or less, send a request for the current 1 x 1 degree square.

This is where the integer values for right ascension and declination speed up the database search dramatically. This occurs in the background so that you can continue to use the planetarium. When the server returns the data, it is then displayed *(Figure 1)*. Figures 1 and 2 have the "all stars" option set. Normal operation would display fewer stars.

Options exist in the software to read adjacent blocks as well, either 3 x 1 or 3 x 3 one-degree blocks.

Exceptions: If the declination is above 84 degrees or below -84 degrees, then the entire right ascension ring is requested for that value. This is because the 1 x 1 degree blocks get very small, some containing only 40 stars. In memory, each of these is treated as one block of data (see Figure 2).

When you are zoomed in enough, the star designations, magnitude, and spectral class

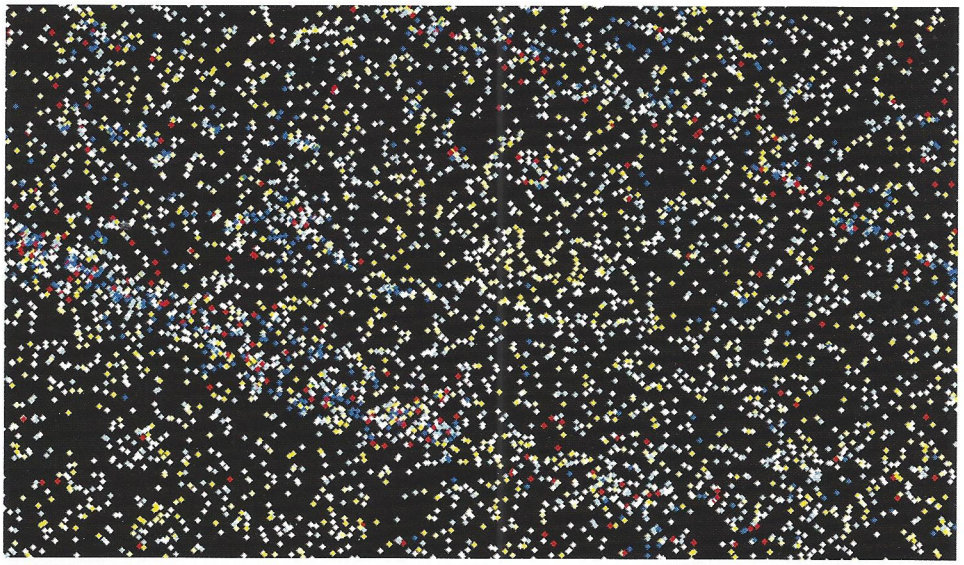


Figure 1: All Gaia-surveyed stars displayed in a portion of M31, the Andromeda Galaxy

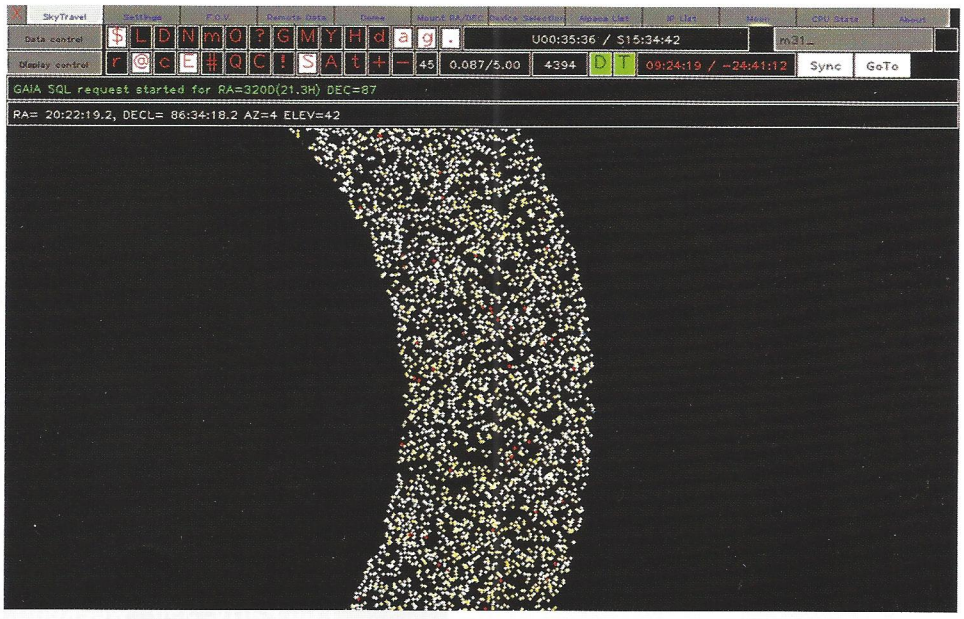


Figure 2: Northern sky showing declination of 87 degrees

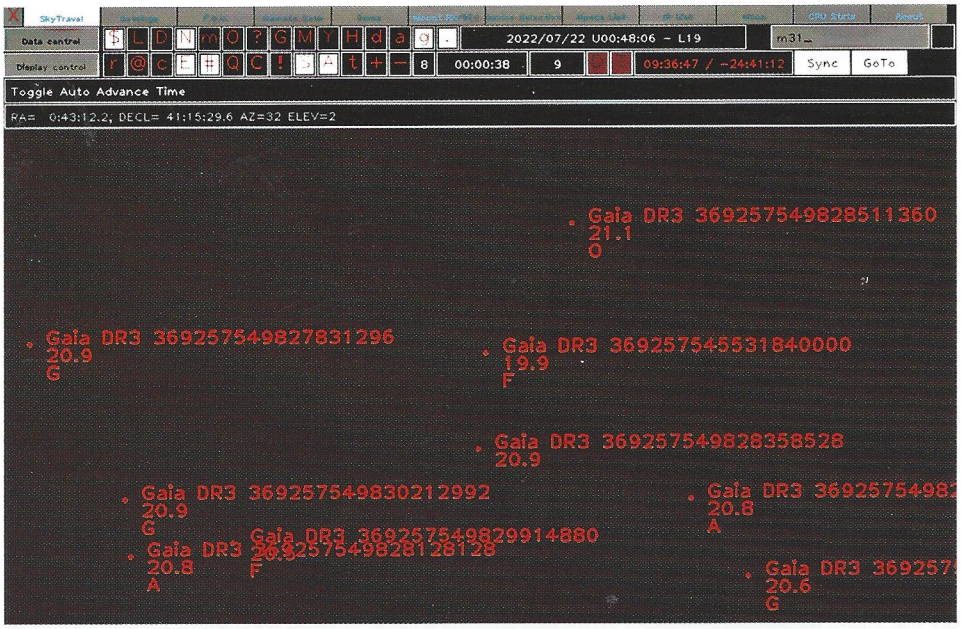


Figure 3: Andromeda zoomed in to show Gaia star labels

are displayed. These can be turned on or off in the settings.

Up to 30 blocks are stored in memory. When the array is full and another block needs to be loaded, the block that is farthest from the current center of the screen is discarded. The value of 30 has no particular significance; it could easily be changed.

CURRENT STATUS

We have the Gaia EDR3 database with 1.8 billon stars and we have the Two Micron All Sky Survey (2MASS) database with 470 million stars.[8] The SkyTravel program can select which one it uses and it retrieves stars from that database. It is also easy to add more star databases by updating the configuration files. We have been testing this with both of us using it at the same time and exploring areas of the sky with high star counts. We also have one user connecting from Oregon all the way to New Jersey.

In the Gaia database, the average number of stars per one degree grid is 28,000; the minimum is less than 50 at the poles and the maximum is 1.1 million. The average time to retrieve all the stars in a single grid is about 3 seconds.

Our current configuration is with the server in central New Jersey with a gigabit connection to the Internet, and Mark is in northeastern Pennsylvania, 100 miles away, with a 50 megabits per second connection. We both run the planetarium program at the same time and have even tried loading the same large data sets at the same time. It worked perfectly! The difference in response time from Mark using it remotely versus Keith using in the same house is minimal.

Full documentation on how to implement this SQL server is also on our GitHub site; see footnotes. Mark's SkyTravel program is also on the GitHub site. If anyone would like to either connect to the SQL server or set up their own SQL Gaia star database server, please contact Keith Sproul at *ksproul@skychariot.com.* For help with the SkyTravel program or the AlpacaPi project, contact Mark Sproul at *msproul@skychariot.com.* ✶

REFERENCES

1. *solarsystem.nasa.gov/missions/gaia/in-depth*

2. *Proceedings for the 39th Annual Symposium of the Society for Astronomical Sciences,* "Gaia Data Enables New Methods of Analysis and Discovery," John Hoot, *socastrosci.org/wp-content/uploads/2021/12/2020_Proceedings_final.pdf.*

3. Installing Ubuntu Server: *ubuntu.com/tutorials/install-ubuntu-server#1-overview*

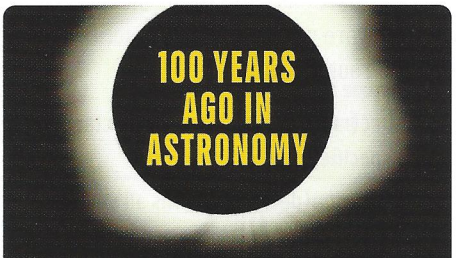
4. Installing MariaDB Server: *mariadb.org. Installing MySQL Server: mysql.com.*

5. Gaia EDR3: *cosmos.esa.int/web/gaia/earlydr3, gea.esac.esa.int/archive*

6. Gaia SQL project on GitHub: *github.com/msproul/GaiaSQL*

7. AlpacaPi project on GitHub: *github.com/msproul/AlpacaPi*

8. 2MASS (Two Micron All Sky Survey): *irsa.ipac.caltech.edu/Missions/2mass.html*



While Arthur Stanley Eddington's observations of the May 29, 1919, total solar eclipse are credited by most sources with providing the first incontrovertible evidence for Einstein's general theory of relativity, the scientific history is more complicated. Questions concerning the accuracy of Eddington's results led astronomers to continue to test the predicted deflection of starlight at later eclipses. William Wallace Campbell, director of Lick Observatory, brought 35 tons of equipment, including a 40-foot camera, on his massive expedition to Western Australia to photograph apparent star positions during the September 21, 1922, eclipse. Careful measurements of the shifted positions of over 100 stars provided more rigorous evidence that the bending of starlight matched that predicted by Einstein. Campbell's work demonstrates how, in science, it's not always about being first. For more information about his eclipse expedition, see:

atlasobscura.com/articles/the-1922-eclipse-expedition-to-remote-western-australia

–Kristine Larsen