Milestone Three

**Enhancement Two Narrative**

Audrey Weaver | Sep 28, 2025

For my second narrative to my custom app MyMessierTracker, I focused on data structures and algorithms where I designed an object rarity metric to help encourage users to keep logging their observations. The goal was to make it fun through tooltips to help quickly identify some of the most popular objects that they have not yet collected as well as reward them in their journal with a star when they have captured something fewer users have.

A screenshot of a video game

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

I selected this enhancement for my ePortfolio because it showcases algorithmic thinking applied on a full-stack project. A new database view leveraging indexes and joins was created that computes the rarity of each object in one grouped process over user\_object\_images and single user counts. This avoids counting per object/N+1 queries in SQL or inefficient loops in Python. This essentially results in O(U + I + M log M) where U indicates the rows in the users table, I as rows in user\_object\_images and M as the rows in the messier\_objects table. Since M is constant at 110, this can simplify to O(U+I) and O(1) in memory so it’s an efficient, speedy and resource friendly algorithm to compute this rarity metric.

create or replace view public.v\_object\_rarity as  
with total\_users as ( -- this is the denominator: a count of all registered users  
 select *count*(\*)::numeric as total\_users  
 from public.users),  
 obj as ( -- this is the numerator: a distinct count of users who have captured each object  
 select mo.id as messier\_id,  
 mo.messier\_number,  
 mo.common\_name,  
 mo.object\_type,  
 *count*(distinct uoi.user\_id)::numeric as user\_ct  
 from public.messier\_objects mo  
 left join public.user\_object\_images uoi  
 on uoi.messier\_id = mo.id  
 group by mo.id, mo.messier\_number, mo.common\_name)  
-- final query calculates the rarity per object  
select o.messier\_id,  
 o.object\_type,  
 o.messier\_number || ': ' || o.common\_name as object,  
 -- gets the % val that can be used downstream to sort and limit by but also append  
 -- a '%' sign to and make sense to the user  
 case  
 when t.total\_users = 0 then 0  
 else *round*(100 \* o.user\_ct / t.total\_users, 0) end as rarity\_pct  
from obj o  
 cross join total\_users t  
order by rarity\_pct desc;

Throughout this process I gained a deeper understanding of why its important to build a robust relational database structure so that computations required for metrics like these can be handled efficiently when using the right tool for the problem instead of forcing this logic into loops within python. Here python can then be used to filter and push this information to the front end.

A computer screen shot of a program

AI-generated content may be incorrect.

This enhancement met all of the outcomes I had originally planned to meet for this enhancement:

**#2** Through clear and professional communication of what the rarity score is and able to easily see why objects were labeled the way they were so the audience can appropriately interact with the app. This also includes well-commented code with appropriate variable names for other developers to understand and maintain the app as well.

**#3** This enhancement included taking into consideration the algorithmic design and trade-offs through computing a rarity score in one grouped pass (using a PostgreSQL view), O(U+I) which avoids N+1 queries and unnecessary python loops. It uses a server side order by and limit to get the top-N and handles nulls and rounding.

**#4** Integrated Postgres > Python > Flask/Jinja framework that uses Highcharts for user facing analytics. This provides a smooth and clear UI for the end user that provides snappy data and interactions.

One challenge I faced was getting the tooltips to display properly, initially I had the sort wrong so it was displaying objects in the wrong order but that was an easy fix. The second issue I came across was that the tooltip felt clucky on the donut charts and would completely cover up the visuals in the center of them. I fixed this by moving the tooltip to the right and using *outside: true* in the html script.