Ocean Biodiversity & Climate Change Explorer

Group Members

- Allen Liu
- Etienne Lee
- Elizabeth Jiang
- Ian Lamont

1. Introduction

Our project combines data from the World Ocean Database (WOD) and the Ocean Biodiversity Information System (OBIS) to create an interactive platform for exploring marine biodiversity and ocean conditions. The application integrates ocean temperature, salinity, and species occurrence data to provide researchers and interested users with tools to explore marine life and ocean characteristics across different locations.

1.1 Project Goals

- Create an accessible interface for exploring marine species data
- Provide interactive visualization of ocean conditions and species distributions
- Enable efficient search and discovery of marine species information
- Analyze relationships between species occurrence patterns and ocean temperatures throughout the year

1.2 Application Functionality

The application provides users with:

- Interactive ocean mapping with detailed location-based data
- Comprehensive species search and information system
- Visualization of ocean conditions including temperature and salinity
- Analysis of species occurrence patterns in relation to seasonal temperature changes

1.3 Architecture and Design

Technologies Used:

1. Frontend
    - React (for building user interface)
    - React Router (for page navigation)
    - MUI (for UI components and styling)
    - Leaflet (for interactive maps)
2. Backend
    - Node.js (runtime environment)
    - Express.js (web framework)
3. Database and Hosting
    - PostgreSQL (database)
    - AWS (hosting)
    - Python (for data loading scripts)
    - Google Colab (for initial data preprocessing)

System Architecture:

The application follows a client-server architecture where:

1. Users interact with React frontend
2. Frontend makes API requests to the Node.js/Express backend
3. Backend queries the PostgreSQL database for ocean and species data
4. Results are returned to frontend to display

Application Flow: Users → React Frontend → Node.js/Express Backend → PostgreSQL Database → Backend → Frontend → Users

1.4 Application Pages

1. Home Page
    - Description: Landing page for all users, simply contains a title and short description of our website, along with a marine image
    - Features: N/A
2. Map Page
    - Description: Contains an interactive ocean map for users to interact with.
    - Features: Interactive map that can zoom in and out, move around, and upon clicking a location on the map gives you detailed information on ocean climate and species around that location.
3. Species Page
    - Description: Contains information about various ocean species
    - Features: Advanced search on ocean species that filters on various fields, detailed species cards for each species containing detailed information about the species.
4. Insights Page
    - Description: Contains insights on ocean data, such as monthly trends for species as well as temperature.
    - Features: Tabular visualizations of marine species distribution patterns over time and space, including monthly observation trends alongside water temperatures and geographic movement analysis between different time periods.

2 Data

2.1 OBIS dataset

2.1.1 Link to source

https://obis.org/

2.1.2 Description of data

OBIS is a global open-access database on marine biodiversity, providing data on where and when marine species are/were present. It aggregates data from many sources and includes information about marine species distributions, abundance, and related environmental data.

2.1.3 Relevant summary statistics

a. Size Statistics:
    i. Number of rows: ~136,119,232 (each row is a species occurrence). Note that we were unable to count exact rows since the dataset was too big to fit into google colab.
    ii. Number of attributes: 283 attributes (includes location, time, species names, depth, environmental conditions, taxonomic levels, and conservation status)
    iii. Download size: 18.9 GB as a parquet file

b. Summary Statistics for a few attributes: (Note that due to the size of the dataset we could not get summary statistics for the full dataset in google colab, so these statistics were taken from the OBIS website.
    i. 196,301 accepted species
    ii. Coverage: all of world's oceans
    iii. Temporal range: 1605-2023
    iv. Taxonomic coverage: from bacteria to whales
    v. Depth: 0-11,034m (maximum ocean depth)

2.1.4 How we used the data

The OBIS dataset was one of the two core datasets used in our project, the other one being WOD. We used this dataset to get detailed information on ocean species and this allowed us to get the data for our species page as well as the interactive map.

2.1.5 Preprocessing Details

For the OBIS data, it actually contained two tables which were MOF and occurrences respectively and the data came in the form of many parquet files. From observing the first parquet file for each table, we found that the MOF table contains details on how each occurrence was measured including measurementValue, measurementType etc. Next, the occurrences table contains extreme detail on each occurrence including detailed species identification, location and temporal info and more, leading to it having 283 columns. The two tables can be joined on Occurence_ID.

At the beginning of preprocessing, we first looked at the MOF table and found that though the data was quite clean, there was an unacceptable number of null values for almost all columns. Apart from the ID columns, most of the columns were completely null and Occurence_ID which was our join condition was also over 50% null. Since this caused the data to be practically unusable, and we found there was some measurement data already in the occurrences table, we decided to drop the MOF table and only use the occurrences table for our project.

Next, we started preprocessing for the occurrences table. First, we found that there were 5207 parquet files in the occurrences folder and that the total size of the parquet files was 13.87 GB, making it impossible to load all the parquet files into a single dataframe due to the sheer volume of data. Thus, we decided to take a random sample of 50 out of the 5207 parquet files and perform EDA on this random

sample to get familiar with the data in the occurrences table.

Using the sample_df, we started looking at which columns we wanted to keep, aiming to go from 283 columns to around 20-30 relevant columns. To summarize, after looking at the null percentages and relevancy of each column through some EDA, we ultimately decided to keep 21 columns initially. Some of the columns were arguably not needed but we decided to be cautious with what we dropped since loading the dataset in again would take at least half a day due to the volume of data.

For reference here, the initial columns we kept were:
id, dataset_id, occurrenceID (core identifiers); aphiaid, family, scientificName (taxonomic information); decimalLatitude, decimalLongitude, marine, brackish, bathymetry, shoredistance, sst, sss, areas (geographic/environmental data); occurrenceStatus, basisOfRecord, flags, dropped, absence (quality and status fields); and eventDate (temporal data)."

Next, we also decided to only use the last ten years of data from both the occurrences table and also the wod table which we were ultimately going to join on for our project. The reasoning behind this was that the size of the datasets was just too big for our project, leading to insane loading times. We decided on keeping the last ten years because it was a good compromise on size reduction and still being able to extract some time dependent trends from our data. Thus, we performed EDA on the eventDate column and wrote a function that would extract the year from this column for later preprocessing.

Finally, we now had to actually preprocess the data based on our findings. Due to the size of the data, we could not just concatenate all parquet files into a dataframe and start preprocessing using Pandas. Thus, instead we decided to:

1. For each chunk of 10 parquet files in the occurrences folder
2. Concatenate them together using Dask
3. Apply the column and year filtering
4. Load the chunk into a temp parquet file stored in a temporary folder

Note that we utilized chunking because simply concatenating all the parquet files would crash colab. We also used Dask over Pandas since Dask handles large datasets much better.

After all parquet files were pre processed into temp parquet files stored in the temporary folder, we were able to concatenate all the processed parquet files into one Dask dataframe due to the size being greatly decreased during filtering, going from around 14 GB to 2-3 GB. Finally, we downloaded the Dask dataframe as a zip file so that we could load the table into AWS locally for our next step.

Database – Explanation of data ingestion procedure and entity resolution efforts, ER diagram, number of instances in each table, and proof of 3NF/BCNF.

Data Ingestion Procedure and entity resolution efforts for OBIS related tables:

First, we had to load the Dask dataframe consisting of the obis data locally into AWS. To do this, we created a python script using sqlalchemy that utilized batching to load in batches of parquet files into AWS at once, which ultimately led to us having a table called obis in our database containing all of the data. This process took around 3 hours to run locally even though I closed everything and have a RTX 3060 GPU, which shows that we still have a massive amount of data even after filtering.

Next, we started to explore the data using datagrip now that it had been fully loaded into the obis table.

The table had around 45 million rows and took up 12GB in our database, furthermore just running a query to check the number of rows took around 2 minutes to run, and doing things like specifying a key or changing a type of a column would crash since our database was limited to 20GB. Therefore, after some discussion with our project TA, we decided to further limit the size of the table by only using data from the year 2015. This reduced our number of rows to around 5 million, making it possible to actually work with the data.

Though more complex queries still took some minutes to run, we could now proceed with cleaning the data further as well as normalizing it.

First, we decided to drop some columns as we now had an updated understanding on which columns we would need for our website. The columns that we decided to drop first were dataset_id, occurenceID, shoredistance, basisOfRecord, year. Since we were not joining any tables on id, we only needed to keep one id over our data and thus could drop the two other id columns. Next, we dropped the year since we were only keeping 2015 data. Next, the other two columns dropped were just not relevant to our website. After this, upon running some queries and looking at the data, we found that occurenceStatus, flags, dropped, absence had all not loaded in properly or were corrupted since initially they were lists within the parquet files. At this point, rather than loading everything in again, we decided in interest of time to just drop these columns. This was also done since the data would have been very hard to parse, and most of the rows seemed to be in good quality from previous EDA.

Next, we decided to clean the data to ensure the quality of the data. To start, we changed the type of id to UUID and made it the primary key of the table. Next, we removed all completely duplicate rows from the table, which ended up being quite a large number of rows (may have been due to removing quality flags). This was actually great because it significantly reduced row count again, further decreasing query speed. Next, we updated the NOT NULL constraints for a number of columns (aphiaid, decimalLatitude, decimalLongtitude). Finally, we had to parse through the eventDate column and extract out month,day to match how the wod stored time. This ended up being a lot of work since there were around 10 different formats that date was stored within the eventDate column. After running a lot of queries involving regex to explore the column, we were eventually able to extract month, day from each eventDate into two new columns month and day. Note that during this process we also had to extrapolate some dates randomly from ranges, and remove some rows with incomplete date formats within the eventDate column.

Next, we decided to normalize the table into 3NF as required by the specification. It was easy to identify id as our candidate key here since it was already the primary key of our table. Next, by how species are recorded, we had the functional dependencies aphiaid → (scientificName, family) and scientificName → family. These were the only functional dependencies in the table apart from id determining everything, and they violated 3NF.

To normalize the table, we ended up making two new tables: scientific_names(aphiaid, scientificName) and families(scientificName, family). During this process, there was a very small number of rows that did not conform to the FDs listed above, these were either deleted or resolved with correct information. Finally, we set up the constraints (foreign keys) between the obis, scientific_names and families tables, and also set up constraints within the new tables which included primary keys and NOT NULL constraints.

2.2 WOD dataset

2.2.1 Link to source

https://www.ncei.noaa.gov/products/world-ocean-database

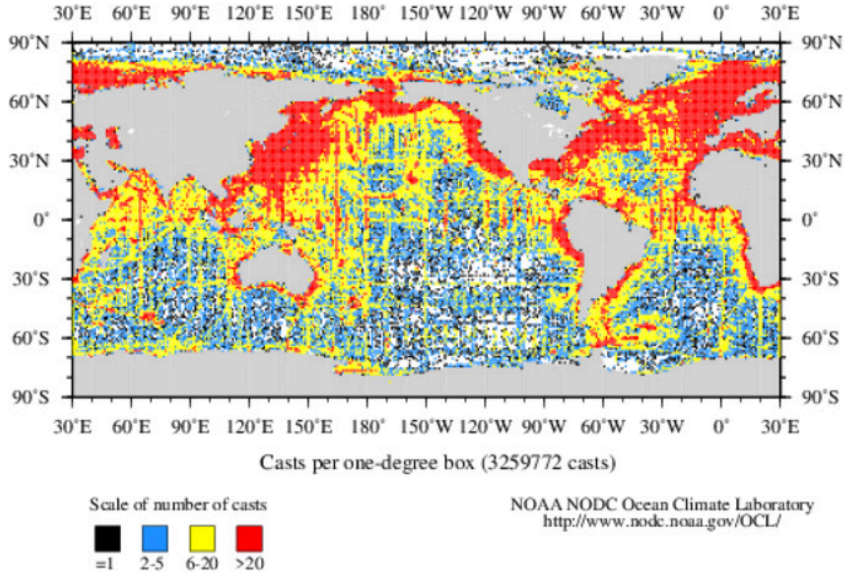
2.2.2 Description of data

The World Ocean Database (WOD) is a collection of datasets pertaining to ocean data. It is maintained by the National Oceanographic and Atmospheric Administration (NOAA). For the sake of this project we used the Ocean Station Data (OSD) dataset to collect data from. OSD is a collection of readings from "stations" which are mostly ships recording data as they are sailing (however there are some other data collection methods).

2.2.3 Relevant summary statistics

c. Size Statistics:
   ii. Amalgamation of 10 different datasets
   iii. 18.5 million casts (sets of datapoints containing multiple
      - 3 million in OSD
   iv. 67 thousand rows in OSD in 2015

a. Summary Statistics for a few attributes (OSD):
   v. At most 24 attributes per record
   vi. Location Distribution:



2.2.4 Reason for dataset use

This dataset is important for our project to get accurate and detailed ocean temperature and nutrient information. While OBIS contains some of this information, it is only at the time of sighting, therefore we cannot get the change in extent we are looking for. Additionally information on nutrient availability could be an important indicator on the underlying reason for locational changes.

2.2.5 Dataset Cleaning and Ingestion

To retrieve the OSD from NOAA, you use a tool called WODSelect that retrieves the data from NOAA and emails it to you in csv format. This tool is on the WOD website. We selected a range of 10 years (which we later narrowed down to just 2015 in the wod_2015 table) and only the attributes mentioned below in the schema. Additionally within WODSelect you can specify flags for cleaning data directly in the data request. We used the flag "7 - failed annual and monthly standard deviation check" which would remove rows that are statistical outliers meaning that there was likely an issue in data collection. After obtaining the data in csv format, since the data was small enough to download directly to a laptop, we uploaded it directly to the database through DataGrip into the "wod" table. After realizing that OBIS could only reasonably be managed by limiting it to just 2015, we created a new table of just the WOD entries from 2015.

2.3 Additional Considerations - Joins:

See section 5.1 for an in depth explanation of why we added the rows grid_id, depth_id, and dayOfYear.

3 Data Modeling

3.1 ER Diagram

Link:
https://drive.google.com/file/d/1WHYM4vsHIuFYrM3jY2B3Mr7Ykr5bE60G/view?usp=drive_link

3.2 Tables in the Postgresql database

1. date_lookup(dayOfYear, month, day)
   All column types: INTEGER
   Constraints: month,day NOT NULL

   Description: Provides a quick lookup from dayOfYear to month and day for the year 2015, thus includes all 365 days in the year 2015

   Number of Instances: 365

2. families(scientificName, family)
   All column types: TEXT

   Description: Maps scientific names of different species to the family that they belong to, note family may be unknown for a species and thus can be null.

   Number of Instances: 23380

3. scientific_names(aphiaid, scientificName)
   Types:

- aphiaid: BIGINT
- scientificName: TEXT

Constraints:

- scientificName NOT NULL
- scientificName FOREIGN KEY REFERENCES families(scientificName)

Description: Maps unique aphiaid of each species to their scientific name, each species must have a scientific name so that column is not nullable. Note that this table has a one-to-one relationship with families.

Number of Instances: 23380

4. nm_grid_15(id, lat_minutes, lon_minutes)

   Types:

   - id: INTEGER
   - lat_minutes, lon_minutes: DOUBLE PRECISION

   Constraints:

   - lat_minutes, lon_minutes NOT NULL

   Description: Partitions the globe into 15 nautical mile squares, where each id represents a unique square. The lat_minutes and lon_minutes represent the center coordinates of each square.

   Number of Instances: 30193

5. obis(id, aphiaid, decimalLatitude, decimalLongtitude, marine, brackish, sst, sss, dayOfYear, grid_id, depth, depth_id)

   Types:

   - id: UUID
   - aphiaid: BIGINT
   - decimalLatitude, decimalLongtitude, sst, sss,depth: DOUBLE PRECISION
   - marine, brackish: BOOLEAN
   - dayOfYear, grid_id, depth_id: INTEGER

   Constraints:

   - aphiaid, decimalLatitude, decimalLongtitude, dayOfYear, grid_id NOT NULL
   - aphiaid FOREIGN KEY REFERENCES scientific_names(aphiaid)
   - dayOfYear FOREIGN KEY REFERENCES date_lookup(dayOfYear)
   - grid_id FOREIGN KEY REFERENCES nm_grid_15(id)

   Description: Contains all species data obtained from obis for the year 2015 specifically.

   Columns include:

   - **aphiaid**: Unique aphiaid to identify species, can obtain scientific name and family of species through joins involving scientific_names and families
   - **decimalLatitude, decimalLongtitude**: Coordinates of where the species was found/spotted
   - **sst, sss**: Data on sea surface conditions, which includes sea surface temperature and sea surface salinity.
   - **marine, brackish**: Habitat flags that tell us if the species lives in a marine and/or brackish environment.
   - **depth**: The depth where the species was found/spotted.

- **dayOfYear**: Day of the year 2015 the species was spotted, exact month and day can be found by joining with date_lookup
- **grid_id**: The grid at which the species was found in, determined by the exact coordinates. This will be used to join with the wod_2015 table since it is impossible to join on specific coordinates.
- **depth_id**: The depth category at which the species is found, partitioned based on depth ranges. This is similarly used to join with wod_2015 since it is impossible to join on specific depths. Note that depth/depth_id can be null here as depth is not always recorded when a species is spotted.

Number of Instances: 2048013

6. wod_2015(id, depth, temperature, salinity, phosphate, nitrate, longitude, latitude, chlorophyll, ph, dayOfYear, grid_id, depth_id)
   Types:
   - id: INTEGER
   - depth, temperature, salinity, phosphate, nitrate, longitude, latitude, chlorophyll, ph: DOUBLE PRECISION
   - dayOfYear, grid_id, depth_id: INTEGER

   Constraints:
   - depth, latitude, longitude, dayOfYear, grid_id, depth_id NOT NULL
   - dayOfYear FOREIGN KEY REFERENCES date_lookup(dayOfYear)
   - grid_id FOREIGN KEY REFERENCES nm_grid_15(id)

   Description: Contains ocean profile data obtained from the WOD for the year 2015 specifically

   Columns include:
   - **depth**: Depth where measurement was taken
   - **temperature**: Temperature where measurement was taken. Note that though temperature can be null it is very rarely unknown.
   - **salinity**: Salinity where measurement was taken
   - **phosphate**: Phosphate level where measurement was taken
   - **nitrate**: Nitrate level where measurement was taken
   - **chlorophyll**: Chlorophyll level where measurement was taken
   - **ph**: pH level where measurement was taken
   - **latitude, longitude**: Coordinates of measurement
   - **dayOfYear**: Day of the year 2015 the measurement was taken, exact month and day can be found by joining with date_lookup
   - **grid_id**: The grid where the measurement was taken, determined by the exact coordinates. This will be used to join with the obis table since it is impossible to join on specific coordinates.
   - **depth_id**: The depth category at which the species is found, partitioned based on depth ranges. This is similarly used to join with obis since it is impossible to join on specific depths.

   Number of Instances: 67613

7. wod(id, depth, temperature, salinity, phosphate, nitrate, longitude, latitude, chlorophyll, ph, year, month, day)
   Types:
   - For columns that exist in wod_2015, the types are identical

- year, month, day: INTEGER

Constraints:

- depth, latitude, longitude, year, month, day NOT NULL

Description: Contains ocean profile data obtained from the WOD for the last 10 years, specifically 2015-2024. This table will be used to obtain simple ocean trends over the last 10 years and will not be joined with other tables.

Columns that are not in wod_2015 include:

- **year:** year that the measurement was taken
- **month:** month that the measurement was taken
- **day:** day that the measurement was taken

Number of Instances: 320438

3.2 3NF Proofs

First, note that for all tables we are using atomic types such as INTEGER, all tables are in 1NF. Now, let us show that each table are also in 3NF

1. date_lookup
   - Functional Dependencies: dayOfYear → month, day
   - Candidate key used: dayOfYear
   - This table is in 3NF since for the only FD dayOfYear → month, day, the LHS dayOfYear is clearly a superkey for date_lookup

2. families
   - Functional Dependencies: scientificName → family
   - Candidate key used: scientificName
   - This table is in 3NF since for the only FD scientificName → family, the LHS scientificName is clearly a superkey for families

3. scientific_names
   - Functional Dependencies: aphiaid → scientificName
   - Candidate key used: aphiaid
   - This table is in 3NF since for the only FD aphiaid → scientificName, the LHS aphiaid is clearly a superkey for scientific_names

4. obis
   - Functional Dependencies:
     - id → (aphiaid, decimalLatitude, decimalLongtitude, marine, brackish, sst, sss, dayOfYear, depth)
     - (decimalLongitude, decimalLatitude) → grid_id
     - depth → depth_id
   - Candidate key used: id
   - Note that this table is not in 3NF because of the violating FD's (decimalLongitude, decimalLatitude) → grid_id and depth → depth_id. These FD's could have been removed easily by creating two new tables each containing the attributes in the violating FD's, but we ultimately decided to keep grid_id and depth_id in the table for the sake of query time. For more details see bottom of this section.

5. wod_2015
   - Functional Dependencies:
     - id → (depth, temperature, salinity, phosphate, nitrate, longitude, latitude, chlorophyll, ph, dayOfYear)
     - (decimalLongitude, decimalLatitude) → grid_id
     - depth → depth_id
   - Candidate key used: id
   - Note that this table is not in 3NF for the exact same reason as obis, because of the violating FD's (decimalLongitude, decimalLatitude) → grid_id and depth → depth_id. Thus for more details see bottom of this section.

6. wod
   - Functional Dependencies: id → (depth, temperature, salinity, phosphate, nitrate, longitude, latitude, chlorophyll, ph, year, month, day)
   - Candidate key used: id
   - This table is in 3NF since for the only FD, the LHS id is clearly a superkey for wod

7. nm_grid_15
   - Functional Dependencies: id → (lat_minutes, lon_minutes)
   - Candidate key used: id
   - This table is in 3NF since for the only FD, the LHS id is clearly a superkey for wod

Reasoning behind keeping violating FD's in obis and wod_2015 tables:

We decided to keep these violating FD's in the two tables because most of the rows uniquely map to grid_id and depth_id depending on (lat,lon) or depth. Thus, creating the two additional tables for 3NF would need every row to be copied over, resulting in two tables with millions of rows. Furthermore, since we regularly join obis and wod_2015 on both grid_id and depth_id simultaneously, this would cause us to have to do 4 additional joins every time just to obtain the grid_id and depth_id. Therefore, since our queries were already taking substantial time to run without the additional tables, we decided in this case maintaining performance would outweigh the benefits of normalization, especially since we are using static data that will not be updated.

4 Queries

Query Examples:

- *Species Monthly Observation Trends*

This query is used in the InsightsPage.js component to show how species observations change throughout the year alongside ocean temperature patterns.

The query takes a scientific name as a parameter and finds all observations of that species, groups observations by month to count how many times the species was seen each month, calculates month-to-month percentage changes in observation counts, joins with average water temperature data for each month. It returns a time-series dataset displaying the relationship between species occurrences and ocean temperature.

- *Species Shift Trends*

This query is used in the geographic shift analysis tool in the InsightsPage.js component. It helps users understand how a marine species' distribution changes between two time periods.

The query takes parameters for two time periods (start and end dates converted to days of year), finds the centroid of a species' observations in each time period, calculates the distance between these centroids, and requires a minimum number of observations in both periods (customizable by the user). It returns the geographic shift data sorted by the most significant movements.

- *Map Page Region Climate Query*

This query is used in the Interactive Map page. It takes the clicked coordinates and determines which predefined ocean region they fall within, calculates climate metrics for that region,and returns a climate summary that's displayed in the "Climate Data Summary" section of the map page

- *Advanced Species Search Query*

This query is used in the species page to do an advanced search over the species. It takes in the name that the user searches as well as all of the filters specified and returns all species matching the criteria. To get all of this information, the query joins scientific_name and obis on every single call as no matter what the filters are we need both of these tables to output the number of sightings and name of species. It then optionally joins with wod_2015 when a temperature filter is specified. Finally, this query uses limit and offset to work with the pagination from frontend as well as sorting by num sightings.

- *Species card query*

This query is used in the species page to display a species card for each species when their name is clicked. It obtains detailed information on the species based on the input scientificName including information on habitat, ocean environment, nutrients among other things. To do this, the query joins scientific_names, obis, and wod_2015 to obtain the base information. Then it aggregates to obtain averages in many fields, and employs fallback methods when the data is unavailable and correctly outputs this to the frontend.

Complex Query	Inputs	Before Optimization Runtime	After Optimization Runtime
Species Monthly Observation Trends	Scientific Name	~18 seconds	~9 seconds
Species Shift Trends	2 Pairs of Dates	~10 seconds	~2 seconds
Advanced Search	The letter c (matches all species whose names start with c)	~7 seconds	~7 seconds
Species card	Species named Copepoda with ~82000 sightings	~6 seconds	~1-2 seconds

Query 1: Heavy use of obis_name scientific name and aphiaid indices
Query 2: Use of obis_name scientific name and aphiaid indices
Query 3: Use of grid_id and depth_id as well as searching aphiaid and (grid_id, depth_id, dayOfYear) which are all indexed in obis. Our hypothesis as to why optimization did not increase query speed here is that this query was composed with a lot of pushes and optional filters, which made it quite difficult to see how the optimization would work.
Query 4: Use of grid_id and depth_id as well as searching aphiaid and (grid_id, depth_id, dayOfYear) which are all indexed in obis and obis_name scientific name index. The indexes here seemed to have significantly sped up this query, though they did not help in query 3.

5. Optimization

5.1 Optimization on Grid_id, depth_id, and nm_grid_15

Early on in planning our design of the database we realized that we would need to discretize various forms of data in order to effectively compute and join on those columns. Take for example 2 readings recorded 0.0001 decimal latitude away from each other. Since they technically have different latitude values any meaningful operation like JOIN ON or AVG would count these as 2 separate locations, even though they functionally are the same location when it comes to measurements.

For values in time, this discretization was easier as our main datasets (obis and wod_2015) were using data from 2015 so simply using a day of year field was sufficient for time of measurement. For value in space (depth, latitude, and longitude), significant clearing was necessary.

For depth and position (latitude and longitude) we decided to “bucket” our data. This involved splitting depth into 6 discrete ranges and position into 15 nautical mile squares covering the globe. Depth ranges were roughly based on ocean zones using a simple case function for the range ([0-30], (30,100], (100,200], (200,500], (500,1000], (1000, inf)). For position we used the function: floor(arc_minute * 60 / 15) which corresponds to a 15 nautical mile chunk (SIDENOTE: 15 nautical miles is about 27km we decided on this size because it is reasonably small on an ocean scale but still allows for many joins).

When running these queries as is we run into time and memory issues. For the depth join we can query results in around 30s. For the location join we run out of memory during the query. Additionally if we were to have memory to complete this operation, according to EXPLAIN we could be running billions of io operations and millions of expensive floor routines.

This is unacceptable to do at a query level and requires way too many resources. To combat this we included a grid_id and depth_id field to precalculate the buckets that each entry would be placed in. For depth the process was calculating simply the value. For position we created temporary columns to store the calculated 15 nautical mile position, added all new pairs into nm_grid_15, joined on the positions, and stored the grid id (then dropped the temporary columns).

After using these optimizations, a depth join is run in under a second and grid join is around 6 seconds. When looking at EXPLAIN, it is clear this is due to the simple scans that query is using without the costly filters/sorts.

5.2 Materialized views

Since all of the data we’re operating on is static, we decided to use materialized views for some of the most common subqueries to speed up operation.

- obis_name
    - This materialized view follows a join of obis on the scientific_names table to allow for faster search for getting human readable names (we also add an index on the name for speeding up the search)
- mv_wod_obis
    - This materialized view joins obis and wod on grid_id (2d location), depth_id (1d location), and dayOfYear (time) giving us a full position of the measurement. This view allows for extremely rapid access to queries at the heart of our goal to analyse the relation between species range and water parameters.

5.3 Indices

- obis_name
    - Index on scientific name in order to speed up search for queries that input human readable names. This index had a real significant impact
- mv_wod_obis
    - Index on aphiaid (species id) so that the joined table can be rapidly searched to find data relating to a specific species quickly

Indices specifically for complex queries 3 and 4 (species searching and species card queries):

- obis
    - Index on aphiaid within the obis table. This index aims to speed up the join between obis and scientific_names which is needed for each of these queries since we are searching using scientificName which is contained solely within the scientific_names table.
    - Index on depth within the obis table. For complex query 3, we filter using depth ranges and for complex query 4 we iterate through depth calculating average depth. Thus for both of these queries, the index on depth aims to shorten computation time for these operations, especially for the depth range filtering as a B+ tree index is extremely fast at range queries
    - Index on (grid_id, depth_id, dayOfYear)within the obis table. Since we frequently join between obis and wod_2015 on these three attributes, this index aims to speed up the join and reduce overall query time
- wod_2015
    - Index on temperature within the wod_2015 table. SImilar to the depth index in obis, since we filter and compute average temperature, this index aims to drastically reduce computation time for those operations.

6 Technical Challenges

Our main technical challenges arose in wrangling the data in OBIS and managing joins in the database. See section 5.1 for an in depth explanation of the issue we found with the out-the-box latitude and longitude data as well as depth.

For the challenge of wrangling the data in OBIS, the main technical challenge here was the size of the dataset. Due to the size of the dataset, initial EDA and data loading were extremely difficult with the code often taking hours to run. This was also augmented by the fact that google colab did not have enough memory for the initial obis dataset and after loading it into AWS we had the same problem. For a more detailed overview of this challenge, view the obis section within the data loading and data ingestion sections where we go into detail about the challenges and how we overcame them in the end.

7 Code

OBIS dataset cleaning links:

1. Initial cleaning and EDA: https://colab.research.google.com/drive/1eE3699mUIJmrAQhDGS4WA9WdBxGG1XEl?usp=sharing
2. Filtering and downloading to local computer: https://colab.research.google.com/drive/1wKkZi_xyOwi0K_9LxAv9Onn_-d1Hzzm7?usp=sharing
3. Supplementary depth data downloading: https://colab.research.google.com/drive/1GCfrsJD2XhAM9mwMShmAhpyGM0t93BVu?usp=sharing

OBIS and WOD dataset ingestion code: Contained within the data_cleaning_and_ingestion folder on Github repo.

8 Acknowledgements

- Used generative AI chatbots (ChatGPT) for ideating and Github Copilot for help with developing the app.
- Thank you to all TAs for helping throughout this project

9 Appendix

9.1 Other Queries:

- Map Page: Get all species in a specified region
    - This query is used in the Interactive Map page. It takes the same region bounds as the climate query, retrieves specific species observations within that geographic area, and includes detailed information about each observation.
- Species Page: Get most observed species
    - This query retrieves the ten species with the highest number of recorded observations. It joins the species and observation tables, counts the total observations per species, and returns a ranked list.
- Species Page: Get species count by habitat type

    - This query groups species by their habitat classification (marine, brackish, both, or neither) using the habitat flags in the observation data. It counts the number of distinct species in each category.
- Species Page: Get species count by month
    - This query groups species count by month from the observation day-of-year field.
- Species Page: Get top species co-occurrences
    - This query identifies pairs of species that frequently co-occur within the same geographic grid cell by performing a self-join on spatially rounded observation data. It counts co-occurrence frequencies and returns the top pairs.
- Species Page: Get random species
    - This query selects a random set of species from the dataset along with their counts and calculated rarity scores.
- Species Page: Filter species results
    - This query filters species based on scientific name, marine/brackish environment, sightings count, depth, temperature, and regions.