

# Surf-Under-Line

<https://github.com/notmaurox/SurfUnderLine>

---

Course Section: CS605.641.84  
Summer, 2020

Prepared by

**Mauro Chavez**  
**7/09/2020**

## Table of Contents

<b>1. INTRODUCTION .....</b>	<b>3</b>
1.1. SCOPE AND PURPOSE OF DOCUMENT .....	3
1.2. PROJECT OBJECTIVE .....	3
<b>2. SYSTEM REQUIREMENTS .....</b>	<b>3</b>
2.1 HARDWARE REQUIREMENTS .....	3
2.2 SOFTWARE REQUIREMENTS .....	3
2.3 FUNCTIONAL REQUIREMENTS .....	4
2.4 DATABASE REQUIREMENTS .....	4
<b>3. DATABASE DESIGN DESCRIPTION .....</b>	<b>4</b>
3.1 DESIGN RATIONALE .....	4
3.2 E/R MODEL .....	6
3.2.1 <i>Entities</i> .....	6
3.2.2 <i>Relationships</i> .....	9
3.2.3 <i>E/R Diagram</i> .....	11
3.3 RELATIONAL MODEL .....	14
3.3.1 <i>Data Dictionary</i> .....	14
3.3.2 <i>Integrity Rules</i> .....	16
3.3.3 <i>Operational Rules</i> .....	17
3.3.4 <i>Operations</i> .....	17
3.4 SECURITY .....	18
3.5 DATABASE BACKUP AND RECOVERY .....	19
3.6 USING DATABASE DESIGN OR CASE TOOL .....	19
3.7 OTHER POSSIBLE E/R RELATIONSHIPS .....	19
<b>4. IMPLEMENTATION DESCRIPTION .....</b>	<b>20</b>
4.1 DATA DICTIONARY .....	20
4.2 ADVANCED FEATURES .....	21
4.3 QUERIES .....	23
4.3.1 <i>Get all of a surfer's logged surf sessions</i> .....	23
4.3.2 <i>Find all wetsuits recommended for a region</i> .....	24
4.3.3 <i>Find the most popular beach for a region</i> .....	24
4.3.4 <i>Find the average height for a specific beach by name</i> .....	24
4.3.5 <i>List all the swells associated with good surf for a beach</i> .....	25
4.3.6 <i>Find all the instances when a surfer disagreed with the Surfline reported wave height by more than 1 foot.</i> .....	25
<b>5. CRUD MATRIX .....</b>	<b>26</b>
5.1 LIST OF ENTITY TYPES .....	26
5.2 LIST OF FUNCTIONS .....	26
<b>6. CONCLUDING REMARKS .....</b>	<b>26</b>
<b>APPENDICES .....</b>	<b>28</b>
<b>REFERENCES .....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>

## **1. Introduction**

This document describes the design and implementation of a relational database which aims to record metrics involved in predicting surf conditions from popular surf resource Surflineline.com. Surfers sometimes find themselves disagreeing with the predictions Surflineline makes about surf quality, these moments of disagreement provide an opportunity to study gaps in Surflineline's knowledge. Surfers know their local beaches best and a collection of surfers from the same area know their beaches even better. The goal of this project was to develop a system for tracking surf metrics, saving historical Surflineline predictions, and recording surfer's feedback on how good those predictions were.

### **1.1. Scope and Purpose of Document**

Within the pages of this document, readers will find a through explanation of the requirements, design, and implementation of the system described above. Extra detail will be provided on the goals of the system, the motivating factors behind choices made in the design of the system, and the way in which the system was implemented. This document touches on the processes of conceptual design, via descriptions of entities and relationships, and logical design, via the presentation of E/R diagrams. Readers will also find descriptions of the features and intended capabilities of this system. A later section will discuss the implementation of this system as a web app via the Django framework.

### **1.2. Project Objective**

The purpose of this project was to design and implement a relational database to address a topic of choice. Project milestones included; obtaining a database design tool, learning to use that design tool, completing conceptual and logical design phases, discussing data integrity, describing business operations, writing SQL queries, and outlining a data dictionary. Due to the personal investment in the functionality provided by this system, the author took extra steps to implement the database in the Django framework in an effort to deploy a web app useable by him and his friends.

## **2. System Requirements**

### **2.1 Hardware Requirements**

System was developed on a machine with the following specs:

Processor: 2.8 GHz Dual-Core Intel Core i5

Memory: 8 GB 1600 MHz DDR3

Free Disk Space: 30 GB

### **2.2 Software Requirements**

OS: macOS Catalina (V. 10.15.3) or better

RDBMS: sqlite3

Other: Django 1.11.2 and Python 3.5

## **2.3 Functional Requirements**

This system is required to support saving data associated with surf reports as they are presented on <https://www.surflines.com/>. Surf metrics surfaced in the Surflines UI should be stored within the database alongside the projected surf quality. Surf reports provide descriptions of projected surf quality which are either in the scope of discussing the conditions at a single beach or a single region(which contains multiple beaches). Surf reports at both levels of scope should be storable within the DB. Additionally, there should be a component of the data base that allows for users to create profiles and comment on surf reports. This system must support users in submitting their own observed values in response to Surflines's surf projections. Over time, as data is collected within the database, one should be able to figure out how often the surfer's experienced surf conditions differ from those projected by Surflines. It should also contain enough datapoints for each surf metric to identify which dimensions of the data are strong or weak indicators of good surf conditions at various locations. Given unlimited time and resources, a machine learning framework could be deployed on the database to generate novel surf reports.

## **2.4 Database Requirements**

This project was built using SQLite3 as it is the default RDBMS packaged with Django.

## **3. Database Design Description**

### **3.1 Design Rationale**

First and foremost, the purpose of this database is to store data which is sourced from the Surflines user interface. This presents the obvious design rational of modeling entities based off of UI components. Appendix A contains screen grabs of the Surflines iOS user interface. These help define the database entities required for data storage. The first of which being one that captures surf reports which come in the form of wave heights and a quality label. Figures 1 and 2 show that these reports can relate to a single beach or a collection of beaches grouped under a region. Figure 3 gives a more detailed look some of the metrics that go into a surf report, these are the tide and the wind, but more detail is provided elsewhere. Figures 4-6 give a breakdown of each metric that goes into a surf report and how many data points are collected. First thing to notice is that data is reported in increments of three hours throughout the day. The second thing to highlight is that predicted surf heights, which are another presentation of the surf report, can be broken down into predictions at three hour increments as well. Furthermore, following the surf height predictions portion of the UI are other boxes of data with distinct headers. These are swell, wind , tide, weather, and nearby buoys. These components to the UI define most of the schema. The rest of the schema is to be designed to support users and a comment system.

Entities of this schema can be broken up into three overlapping realms of data capture; the beach, the surf, and the reviews. The first being a storage space for information about beaches, their names, their locations, and what general region they belong to. The second being involved in recording objective measurements of various datatypes that are frequently captured when studying waves as they break on the shore. The third realm of data capture is concerned with the subjective reviews of a given set of surf condition metrics. All three of these elements come together to support a system which tracks the surf conditions at various locations, and records both professional and amateur reviews of these conditions. A commonality across almost every entity in this schema is that they all have unique id's as a primary key (with the exception of the relationship entity capturing the many to many relationship between BEACHes and BUOYs). In the back end, unique ids would be implemented as an incrementing counter for each table. The entities in this schema model things in the real world where uniqueness amongst attributes is not clear cut. It's hard to say with certainty that no two beaches tracked by Surflines will have the same name, or that no two surfers will have the same combination of first name, last name, and skill level. In the case of BEACHes, a composite primary key of the latitude and longitude attributes would be unique, but then all child entities would have to store both of these values which are both 8 characters long. This would ultimately require more space to store data and make the schema harder to work with in both diagramming and implementing steps. Due to the concern that one cannot say with certainty which attribute of each entity will always be unique, the choice was made for all strong entity types to have a unique id assigned for each record.

Measurements associated with surf conditions mostly come in discrete sets. For a three-hour window at a specific beach, there should be a wind, swell, tide, and weather data point. This presents a single table called SURF\_DATA where all attributes of each of these metrics are merged into a single record. That being said, certain surfers might find certain measurements more important than others. It's not unreasonable to assume that most users of this system wouldn't really care to insert weather data as it's not particularly indicative of surf conditions. Certain users will have specific types of metrics they report more than others. In an effort not to enforce strict completeness of data upon entry to make it as easy as possible for users to submit data, a single table for all data types loses its utility as it would be very sparse with many null fields. Furthermore, the end goal of this system is to identify specific measurement indicators of good surf conditions. It is easier to find the best swell direction for a beach if swell data is partitioned out into its own table.

One thing that will be discussed later is that the SWELL measurements have optional 1:1 relationships with both BEACH and BUOY\_READING. They must either have a foreign key to a BEACH record or a BUOY\_READING record, and sometimes both. However, they cannot have a foreign key to neither. This is a result of a proprietary algorithm called LOLA. Surflines filters readings coming from buoys and selects a subset which will appear as influencing surf conditions at a specific beach. Without any historical data to study, it's hard to tell when a swell measurement will only relate

to buoy and when it will show up under a beach, so flexibility exists in the schema to make these measurements a child of either a buoy and/or a beach.

## **3.2 E/R Model**

### **3.2.1 Entities**

#### **The Beaches...**

##### **BEACH**

The BEACH entity stores information about the beaches, the places where surfing happens. Beaches in Surflines are reported in the Surflines UI as having a name and a vague location as identified by a dot on a map. These characteristics were translated into the name, latitude, and longitude attributes of the table. The time\_impact attribute is a rough estimate of how long it takes to get to the beach from where one can park his/her car. Some spots have parking by the water and others have to be ridden to on motor bikes. The domain of this attribute is a 2-character integer to capture values 1-10. Beaches with a time impact of 1 have parking right on the beach and values of 10 require auxiliary transportation to get to. Beaches also have a minimum recommended skill level as reported by a value 1-10, beginners being level 1 and pros being level 10. Upon the creation of a BEACH record, the user creating the record will give a rough estimate of this value. Over time, as surfers are registered in the DB, DB triggers can be used to find surfers who surf at that beach and this attribute can be set to an average of all their self-reported skill levels.

##### **REGION**

REGION captures the broader grouping that BEACHES are classified under. Surflines already provides this grouping but it is useful to store in the database as it would be interesting to one day study trends amongst beaches in the same region. Beaches in the same region are geographically close together and down the line, one might want to identify outliers which respond differently from other beaches in the region to similar conditions.

##### **LIGHTING**

LIGHTING is a bit of an edge case as it is like a recorded surf metric, but doesn't belong to the surf conditions category. Each day, one can measure and record first light, last light, sunrise, and sunset from tracking the movement of the sun. Surflines reports a set of lighting conditions for each beach and has a new set of datapoints every day. Lighting reports are increments throughout the day that more so let surfers know which conditions to pay attention to, and less so function as component of the conditions themselves. A good set of conditions will not matter if they are occurring when it's pitch-black outside. LIGHTING then serves as a way

to filter all the measurements associated with surf conditions on a given day to find the subset of that will be relevant to the surfer figuring out when to hit the beach.

## **The Surf Conditions...**

### **WIND**

Via the Surfline UI, each beach has its own set of wind conditions which are measurements with increments occurring every three hours. This means that for a given day, each beach will have 8 wind readings. Winds have a speed recorded in kts, a direction from a 16 point compass rose, and a degree. The 16 point compass rose reports directions with up to 3 characters and degrees are always between 1 and 360.

### **SWELL**

Similar to winds, swells are also reported every three hours, but multiple swells can be reported at a beach for each increment. There can be between 1 to 6 swells per increment as allowed in the Surfline UI. Swells have a height in feet which is recorded up to 1 decimal place. Similar to wind, they also have a direction on a 16 point compass and a degree. Additionally, they have a period which reports the time between waves. Swells can either rise or fall, so the growth\_direction attribute will record whether the swell is growing in strength or becoming weaker.

### **BUOY\_READING**

Buoy readings, as discussed above, are a summary of the various readings coming from off-shore buoy systems, the primary source being the National Data Buoy Center. Surfline uses a proprietary algorithm which summarizes data coming from a buoy into a significant wave height, peak period, and mean wave direction. These metrics have the same domain as the height, period, and degree attributes in the SWELL table. Each beach is reported to have a set of nearby buoys and each swell for a beach has to have been recorded by some buoy out in the ocean. The Surfline UI reports the individual swells captured by each buoy, for a given date and time, but it's not clear which one of those swells will be reported for a beach.

### **WEATHER**

Similar to the case of both SWELL and WIND, weather is reported in increments of three hours. For each increment, there is a reported temperature and sky condition.

### **TIDE**

TIDE is different from the other metrics. Instead of discrete datapoints for each three hour increment throughout the day, the Surfline UI only reports a height in feet for when the tide changes. For example, if the tide is peaking and is about to

start going out, the tip of this peak will be reported with a height, time and whether or not the peak was at a local min or max. These are the crucial metrics to record for tide conditions. The growth direction attribute of this table will be used to capture if from this inflection point, the tide is going to increase or decrease in height. 0 meaning decreasing and 1 meaning increasing.

## READING\_INCREMENT

This entity is a joining table that collects all measurements that are reported for a specific time under a common record. As described above, many measurements are reported on three-hour increments throughout the day. This means that the exact same date and time will show up in at least once in the WIND, SWELL, and WEATHER relations. The rationale for having a designated READING\_INCREMENT table instead of date and time attributes in each of the above-mentioned tables, was that it would cut down on the validation required for each new set of measurements being uploaded at a time. A new READING\_INCREMENT would be created for a set of measurements and all measurements would have foreign keys to that record. Additionally, it would be very easy to identify a time and get all the records associated with that time. Furthermore, breaking date and time into separate attributes also makes queries for surf conditions for an entire day, regardless of time, easier as well. This is important as surf conditions are often looked at in the context of an entire day as people will spend multiple hours at the beach and be concerned with more than a single three-hour window.

## The Reviews...

## SURFLINE\_REPORT

The Surfline Surf Quality Scale  
© Copyright 2000-2010 Surfline/Wavetrak, Inc.

- 1 = FLAT: Unsurfable or flat conditions. No surf.
- 2 = VERY POOR: Due to lack of surf, very poor wave shape for surfing, bad surf due to other conditions like wind, tides, or very stormy surf.
- 3 = POOR: Poor surf with some (30%) FAIR waves to ride.
- 4 = POOR to FAIR: Generally poor surf with many (50%) FAIR waves to ride.
- 5 = FAIR: Very average surf with most (70%) waves rideable.
- 6 = FAIR to GOOD: Fair surf with some (30%) GOOD waves.
- 7 = GOOD: Generally fair surf with many (50%) GOOD waves.
- 8 = VERY GOOD: Generally good surf with most (70%) GOOD waves.
- 9 = GOOD to EPIC: Very good surf with many (50%) EPIC waves.
- 10 = EPIC: Incredible surf with most (70%) waves being EPIC to ride and generally some of the best surf all year.

The attributes of this table are modeled after the Surfline UI. For each day, Surfline reports the minimum and maximum expected surf height and a score from 1-10. Additionally, reports can be broken down and viewed as specific to three hour increments throughout the day. The scores have an associated label to describe the value of the score. The mappings from numerical scores to their descriptive label is explained in the figure below. In the Surfline app, they display the strings and not the values. For this schema, surf scores are recorded in numerical

form and not in string form. This allows for more obvious sorting and comparisons across records in this table.

## SURFER



This table serves as a sort of USER table. The goal of this database is to allow direct comparisons between what Surflin says the quality of the surf is and what surfers say the quality of the surf is. This table saves a surfer's name and skill level. Additionally, it records their region and favorite beach.

### **ANNOTATION**

Annotations are a surfer's comment on Surflin's predicted surf conditions. It allows for surfers to report their own observed minimum and maximum wave heights along with their own quality score. Additionally, it leaves space for a comment on that day's conditions. As the database is built up over time, the commonalities in attributes across the SURFLINE\_REPORT and ANNOTATION tables will allow for direct comparisons to measure how often they agree or disagree.

### **WETSUIT**

With each report, Surflin recommends a wetsuit for those conditions. These are most likely recommended based on thickness as determined by the temperature. However, this information is recorded along with the SURFLINE\_REPORT so that trends in recommended wetsuit thickness can be studied. If a 3/2 wetsuit in thickness is recommended 99% of the time for a given beach, this information can help consumers make better purchasing decisions.

### **Other**

### **BEACH\_BUOYS**

This relationship entity models the many to many relationship between beaches and buoys. More information will be presented in section 3.2.2 below, but in short, this table allows a single buoy to provide data to multiple beaches and for beaches to have data from multiple buoys.

## **3.2.2 Relationships**

The relationships amongst entities in this schema come in two types, relationships that map measurements to beaches, and relationships that map surf reports to either beaches or regions. For the most part, there are two major orchestrators of relationships amongst entities in this database, the BEACH and READING\_INCREMENT tables.

### **READING\_INCREMENT relationships**

Each measurement in the database is taken at a specific time. As discussed above, these are most often reported in increments of three hours. Additionally, reports are given for specific times of the day. Every record that has a time associated with it

(WIND, SWELL, BUOY\_READING, WHEATHER, TIDE, SURFLINE\_REPORT) must contain a foreign key to READING\_INCREMENT. Each measurement is required to have a foreign key to a READING\_INCREMENT, but a READING\_INCREMENT does not require a relationship with every type of measurement. This allows for partial data reporting. For example, if a user only reports the SWELL and WIND for a given day, there should still be a READING\_INCREMENT for that time even if WEATHER and TIDE data are missing. In the case of SURFLINE\_REPORTS, sometimes they summarize an entire day's conditions and other times they are specific to three-hour windows. This can be captured via foreign keys to READING\_INCREMENT records with either a date and time, or just a date. For each BEACH, every time someone submits data, there will be a set of measurements. When two people submit measurements for the exact same time, but occurring at two different beaches, a single READING\_INCREMENT will have a set of measurements for beach A and a set for beach B. This means that a single READING\_INCREMENT record can have zero to many children from each measurement table.

### **BEACH relationships**

A BEACH will have many data points associated with it as historical data is captured over time. Similarly, it will also have many reports published by Surflin which summarize these data points. This connection, that a beach will have a collection of data points associated with it, is captured by its one to many relationships with the WIND, SWELL, WHEATHER, TIDE, and SURFLINE\_REPORT tables. Records in each of these tables must have a foreign key to the beach they are reporting data for.

Beaches also have a one to one relationship with REGION, and region has a zero to many relationship with BEACH. A Beach must belong to a region but it's not unlikely that a region will be added to the database before any beaches are associated with it.

### **BEACH to BUOY\_READING many to many relationship**

Via the Surflin UI, there are a collection of buoys which are providing readings for beaches in the area. Each beach has multiple buoys, and each buoy reports data to multiple beaches. This presents a many to many relationship which is captured by the BEACH\_BUOYS relationship entity.

### **SWELL to BUOY\_READING and BEACH relationships**

Buoys have a significant swell reported in addition to a collection of individual swells. Surflin's proprietary algorithm takes the swells reported from each buoy and figures out which of the swells appear at each beach. This means that there may be buoys associated with a beach that are reporting swells, but that not all of these swells are identified as showing up at the beach itself. This being the case, each

SWELL must either have a foreign key to a BEACH, or a foreign key to a BUOY. As records are added to the database, as either originating from a BEACH or BUOY, collisions in records will reveal which swells Surfline's proprietary algorithm highlighted as significant.

### **SURFER relationships**

As mentioned earlier, the SURFER table functions as a sort of USER table. Each surfer reports their name and skill level, along with a local beach and local region. Each surfer must report this information, and this means that a BEACH will have zero to many surfers. It is also the case that a REGION might have zero to many surfers as well. Surfers leave their comments on SURFLINE\_REPORTS, so a Surfer can have zero to many ANNOTATIONS. Each ANNOTATION must be recorded with respect to the surfer who wrote it. To support ease of data entry from users, while ANNOTATIONS eventually end up as a comment on a Surfline report, they may be entered into the database before a SURFLINE\_REPORT is. The critical components to the ANNOTATION are; which beach was the observation taken, and at what time was it taken? Given these two pieces of information, the SURFLINE\_REPORT for the same beach at that same time can be found after the fact via scheduled database operations. It is not critical that relationships to the parent SURFLINE\_REPORT record are established upon the creation of an ANNOTATION. Allowing users to insert their data without asking them to specify the SURFLINE\_REPORT for that same window of time at that same beach will make it easier for users to report their observations, an activity that must be encouraged to support crowd sourcing local knowledge. Additionally, the query of finding all surfer annotations for a certain beach require a join between BEACH and ANNOTATION. If ANNOTATIONS were only comments on SURFLINE\_REPORTS where the knowledge of the BEACH and TIME were stored in the SURFLINE\_REPORT record, a three-way join would be required to get the same data. When SURFERS and SURFLINE\_REPORTS are initially created, it is reasonable that they will not have any ANNOTATIONS yet.

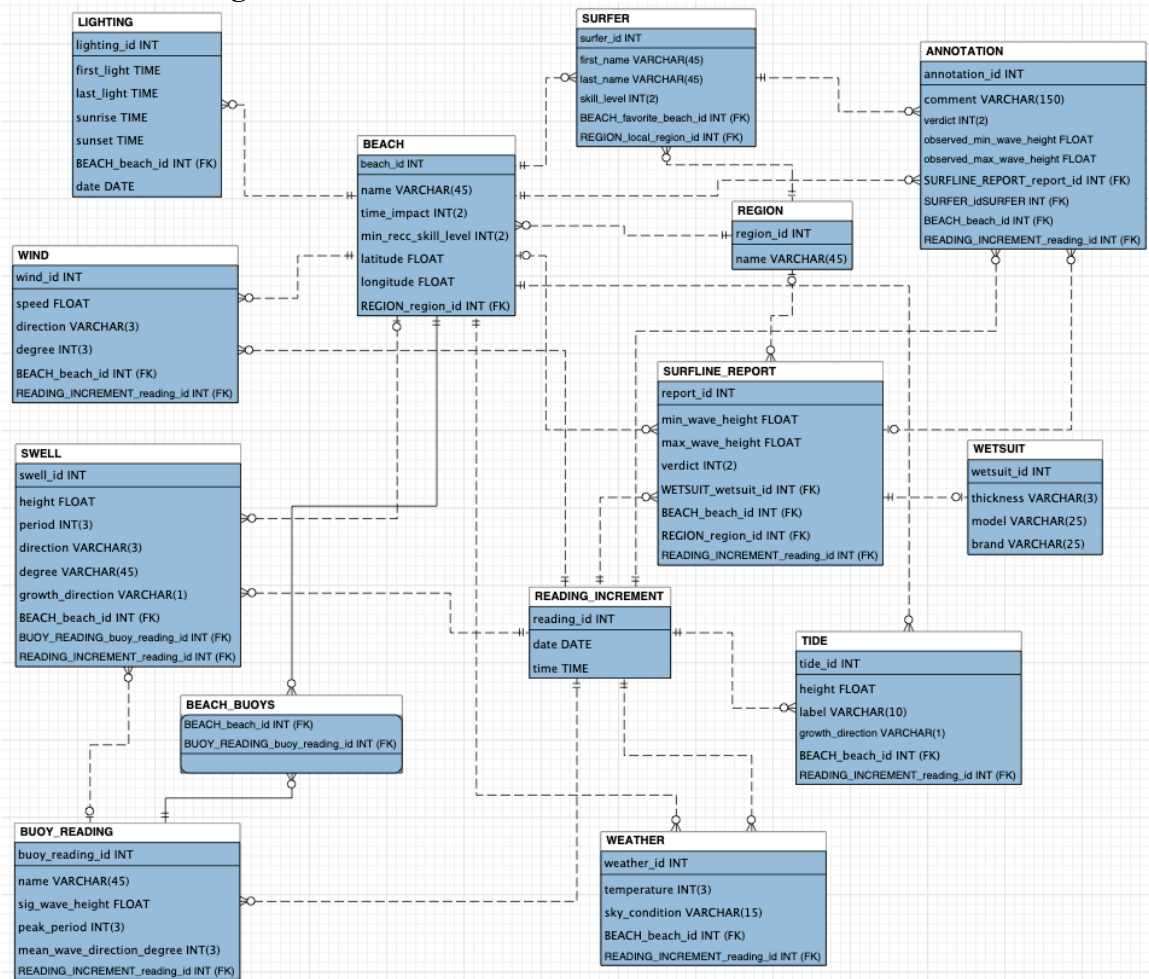
### **SURFLINE\_REPORTS of a BEACH vs a REGION**

Surfline gives reports for each beach in addition to reports for each region. As a result, a record in the SURFLINE\_REPORT table may be associated with a beach or a region. That being said, it must be associated with at least one of these via a non-null foreign key.

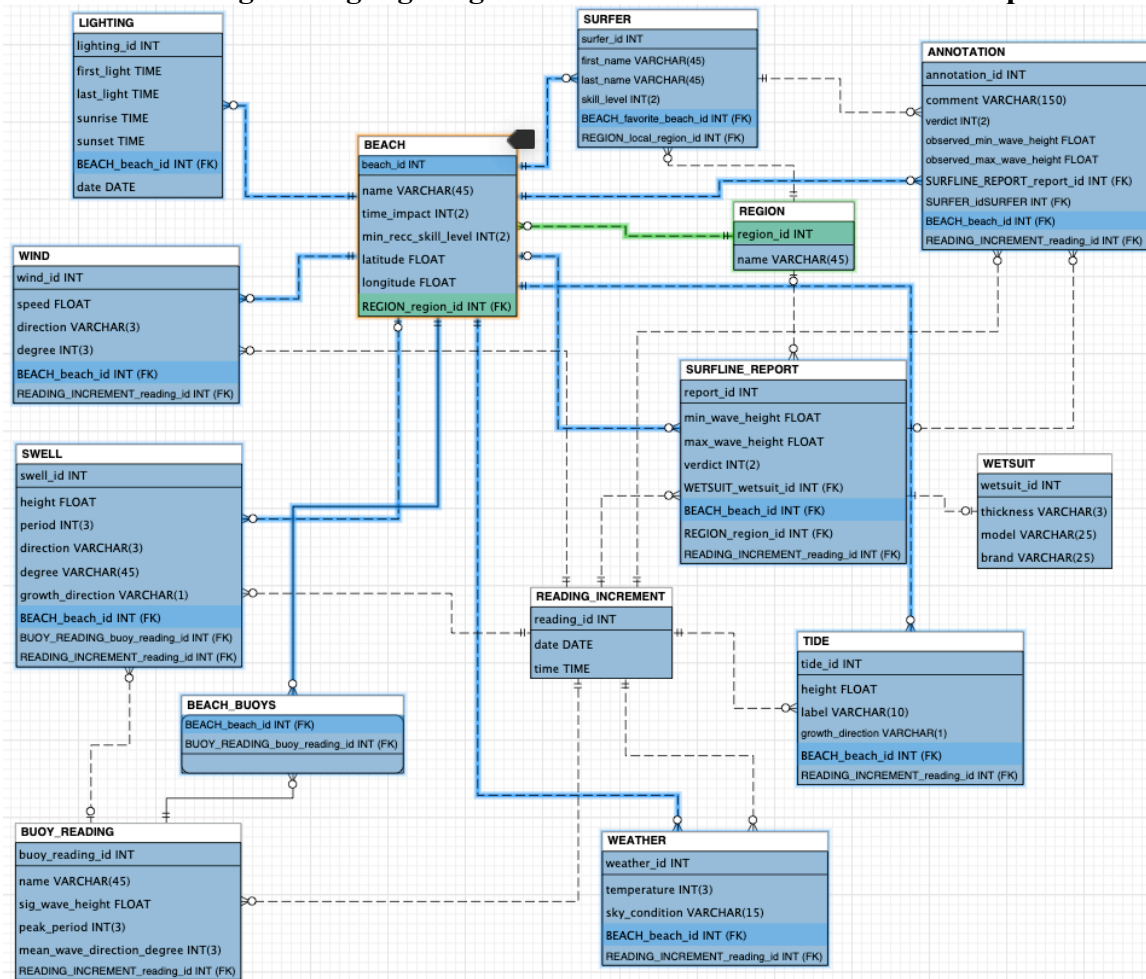
### **3.2.3 E/R Diagram**

*Continued on next page...*

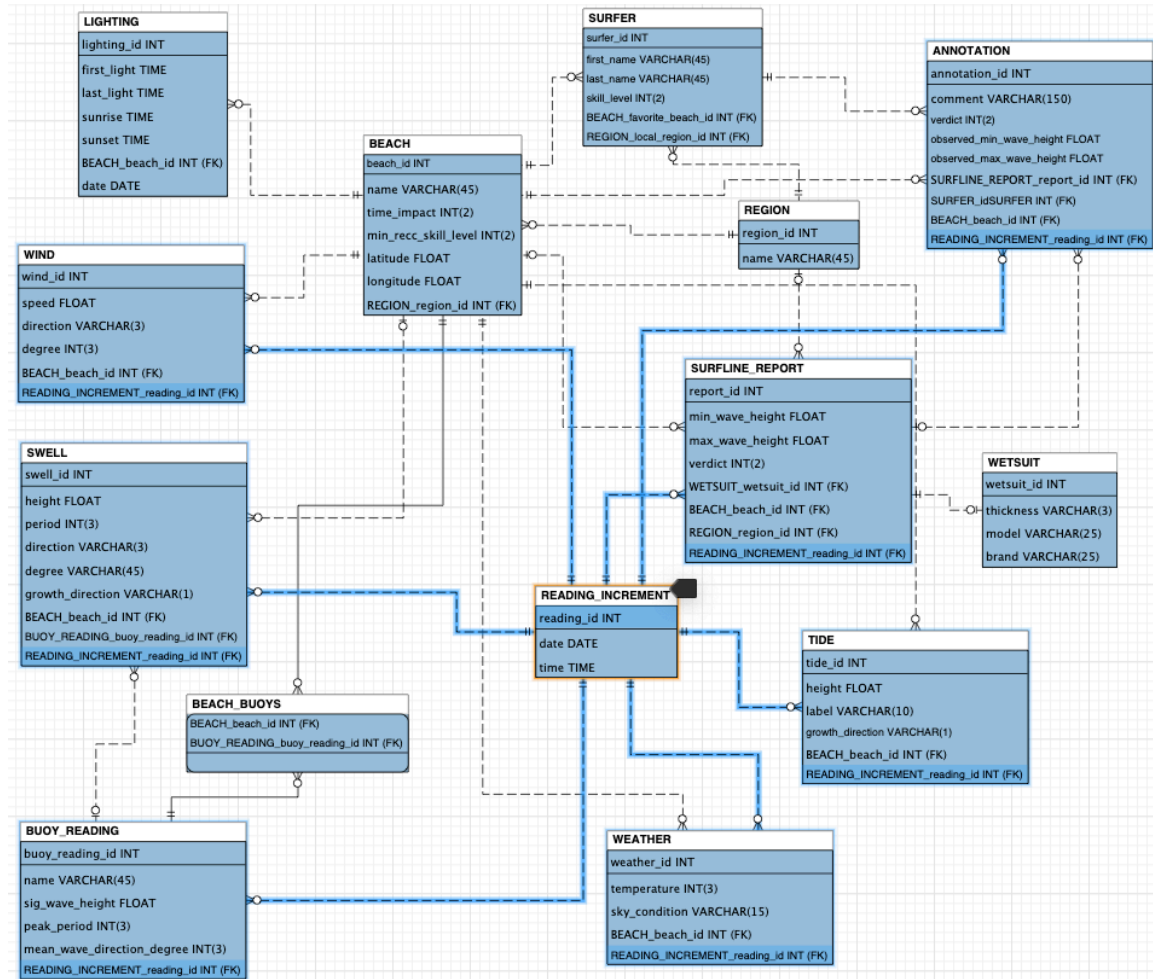
## Schema diagram



## Schema Diagram highlighting BEACH and its associated relationships



## Schema Diagram highlighting **READING\_INCREMENT** and its associated relationships



### 3.3 Relational Model

#### 3.3.1 Data Dictionary

*Continued on next page...*

Column Name(s)	Description	Data Type	Size	Constraint Type	Allow Null	Valid Values
lighting_id, beach_id, surfer_id, annotation_id, region_id, wind_id, swell_id, reading_id, report_id, wetsuit_id, tide_id, weather_id, swell_id, buoy_reading_id	Auto incrementing unique ID	INT	n/a	Primary Key	n	0-10 numeric digits
BEACH_beach_id, READING_INCREMENT_reading_id, REGION_region_id, SURFER_idSURFER	Required reference to the Primary Key of the parent record	INT	n/a	Foreign key	n	0-10 numeric digits
BUOY_READING_bouy_reading_id, WETSUIT_wetsuit_id, SURFLINE_REPORT_report_id, SURFLINE_REPORT.READING_INCREMENT_reading_id, SURFLINE_REPORT.BEACH_beach_id	Optional reference to the Primary Key of the parent record	INT	n/a	Foreign Key	y	0-10 numeric digits
date	Calendar date	DATE	5	Attribute	n	YYYY-MM-DD
time first_light last_light sunrise sunset	Clock time	TIME	8	Attribute	n	HH:MM:SS
time_impact min_recc_skill_level skill_level verdict	10 based rating scale where 0 represents no-impact/noobie/flat and 10 represents maximum impact/professional/epic.	INT	2	Attribute	n y y n	1, 2, 3, 4, 5, 6, 7, 8, 9, 10
name first_name last_name	Stores name of the associated record	VARCHAR	45	Attribute	n	45 alpha-numeric characters
latitude	Geographical coordinates	FLOAT	n/a	Attribute	n	-90 to 90 with up to 3 decimal places reported
latitude	Geographical coordinates	FLOAT	n/a	Attribute	n	-180 to 80 with up to 3 decimal places reported
min_wave_height max_wave_height observed_min_wave_height observed_max_wave_height height sig_wave_height	Measurement taken in feet	FLOAT	n/a	Attribute	n	0-250 with up to 1 decimal places reported
direction	Direction found on 16 point compass	VARCHAR	3	Attribute	n	3 character code

degree mean_wave_direction_degree	Measure of an angle	INT	3	Attribute	n	Whole number between 0-360
comment	Short text written by SURFER as their response to a SURFLINE REPORT	VARCHAR	150	Attribute	y	150 alpha-numeric characters
thickness	Thickness of a wetsuit reported in 'N/M' format where N is thickness in the arms and M is thickness in the body.	VARCHAR	3	Attribute	n	2 single digit integers separated by a '/' character
model brand	Used to record information about a wetsuit	VARCHAR	25	Attribute	n	25 alpha-numeric characters
period peak_period	Time between peaks or troughs of wave's measured by buoys	INT	3	Attribute	n	Whole number between 0 and 999.
speed	Wind speed recorded in kts	FLOAT	n/a	Attribute	n	n/a
growth_direction	Flag reporting if a value is growing (1), or shrinking (0)	INT	1	Attribute	n	0 or 1
label	String storing the state of the tide (high, low, rising, falling, mid...)	VARCHAR	10	Attribute	y	10 alpha-numeric characters
temperature	Temperature recorded in Fahrenheit with no decimals	INT	3	Attribute	n	Whole number between 0 and 999
sky_condition	Short description of the sky (clear, cloudy, partly_cloudy...)	VARCHAR	15	Attribute	n	15 alpha-numeric characters

### 3.3.2 Integrity Rules

A unique characteristic of the system which encapsulates this schema is that most of its data is lifted from an existing source. Surfline currently does not allow users to see past measurements and has yet to implement a logging feature like the one this system aims to deliver. That being said, the majority of the data that ends up in the schema has already been through Surfline's validation and the domains of expected values are well defined.

Given that data is provided by Surfline and added to this schema via an ETL, the design of this database is very strict on mandatory fields. For example, it doesn't make sense to create a SWELL record that would only have a height and a period. If a user knows these two pieces of information, they should have all the other information required for each attribute of the SWELL table. As a result, there are very few optional fields in this design. Most optional fields come in the form of foreign keys where certain database records might be added before others. An example being the ANNOTAITON records. A surfer could log their session before an ETL uploads Surfline's report for that day.



The domains of many of the attributes of this scheme are also very strict. Again, this loops back to the data being sourced from Surflines. For example, the direction attribute will only ever be a direction found on a 16-point compass which is at most three characters. The allowed values for this field are then only strings of this length. A few other attributes in this scheme map to domains with a tight and well-defined set of values. As a result, many of them only allow short strings or integers of few digits.

There are many relationships across all the tables in this schema as shown in section 3.2.3. For the most part, referential integrity is established by requiring that records from many different entity types have a non-null foreign key to BEACH and READING\_INCREMENT records. In the case of SURFLINE\_REPORTs, they should contain a foreign key to either a BEACH or a REGION. Almost every table in this schema (with the exception of SURFER, WETSUIT, and REGION) describes features of a beach at a specific time. So long as all these records maintain references to the time they were recorded and which beach they were recorded at, referential integrity will be maintained.

### **3.3.3 Operational Rules**

For the major users of this database, there are very few operational rules. As discussed in more detail in section 3.3.4, most of the data captured by this system should be put there automatically on a regular schedule as orchestrated by some other module.

Users for the most part are the surfers, they will be able to register an account by providing their name, favorite beach, local region, and skill level. The skill level must be an integer between 1 and 10 and their favorite beach and local region must already exist in the database. After that, they will be given a user ID for logging into the system. This means that many surfers of the same name will have unique keys to access the web-app. However, they will not be able to log their info if the beach they surf at doesn't exist in the database. In this sense, the web app will need to be rolled out to new regions with database admins taking care of creating new BEACHes and REGIONs. From there, if a surfer is able to make an account, their local region will already have been created along with all the local beaches. This is all they need to start logging their surf sessions. Outside of that, surfers will only ever interact with the database by seeing views which provide them reports on the data collected with respect to the beaches they surf at. Measurements will be added by admins and then by an automated system once it is implemented. These measurements will need to be associated with a BEACH but it is the job of the database admins to ensure that a BEACH exists prior to measurements being added. This is important because beaches require precise coordinates and grouping under a region. For data integrity concerns, there cannot be two records of the same beach with different coordinates.

### **3.3.4 Operations**

There are two major operations that encompass most of the interactions with the database. Surfers logging their sessions and data being added from Surflines.

Given the nature of data availability, a single set of measurements and a Surfline report for a single time increment will all be inserted into the database at once. This data is available from the Surfline UI and it would be strange for some of these measurements to be available while others aren't. That being said, in order for this data to be inserted, a BEACH must already exist in the database. Beaches should not be created at the time of measurement insertion as beaches must be created with data that is far outside the scope of what is available from the measurements. Also, the combination of READING\_INCREMENT and BEACH is how all these measurements are sorted and queried; this means a BEACH must already exist before measurements are added. Furthermore, when measurements are added to the database, there should be a validation trigger/operation that checks the READING\_INCREMENT which links them all together. The Surfline UI presents datapoints at predefined time increments. This means that so long as data is sourced from Surfline, it should always be associated with one 3-hour window.

When a surfer logs their session, they will provide a beach and a time increment. The options for these two fields must be populated from records already in the database. For example, the logging of a surf session via an ANNOTATION should never trigger the creation of a BEACH. The Beach should already exist in the database. Additionally, the log must coincide with a three-hour window as captured by the records in READING\_INCREMENT. This means that it can trigger the creation of a READING\_INCREMENT, but one with validated data. This validation will happen at the application layer. A database trigger that occurs after the insertion of a SURFLINE\_REPORT will establish foreign keys in ANNOTATION records to the SURFLINE\_REPORTS that occur at the same beach at the same time. Furthermore, upon the creation of an ANNOTATION, the foreign key to the parent SURFLINE\_REPORT record should be established if one exists with matching BEACH and READING\_INCREMENT foreign keys.

Little to no use cases will involve an update or delete operation. The goal of this system is to collect as much data as possible. Due to the structured nature of the data, if it passes validation, there should be little reason to delete or update it at a later date. The records in the database will originate from Surfline so this system will be more involved in data capture than data maintenance. Surfline has already done a lot of data maintenance by the time information is displayed in the UI and this system will take advantage of that upstream work.

### **3.4 Security**

For the most part, there will be two major categories of people interacting with the database. Surfers will be adding annotations to Surfline Reports via uploading their comments and reviews of their own session through a web app. This means they will only ever have access to creating records in the SURFER and ANNOTATION tables. Validation here will require that good data is being provided that is in line with historical trends for the same beaches and regions. Given a fully built out system, the pivoting of data from Surfline's UI to SurfUnderLine's relational database should happen programmatically. This would mean that the only people who have the ability to interact

with every table in the database are the database admins and the software that is uploading data from Surflines UI on a regular basis. It will be up to admins to create BEACH and REGION records so that the automated data upload has records to assign the metrics and reports to. The workflow should be: configure a beach and give a back end system the URL where measurements from that beach will be found. After that, an automated process should check the URL every day for new data. Eventually, surfers would get their own custom views for seeing their past surf sessions and how they agreed/disagreed with Surflines reports. This would be a presentation data layer and the goal would be to develop a system where users aren't in charge of reporting the metrics already present in the Surflines UI. In short, this should be an automated process.

### **3.5 Database Backup and Recovery**

Sqlite, in the context of Django apps, stores the database in a single file called db.sqlite3 within the directory of the app itself. Creating a database backup is as easy as copying the db.sqlite file to somewhere safe. In the production instance of the SurfUnderLine Django app hosted on a remote server, a cronjob to copy the db.sqlite3 file to a different directory with a timestamp will create snapshots of the database. A cronjob on the developer's local machine to sync the timestamped database copies to a local directory will allow for a backup incase the remote server goes down. Sqlite3 also supports dumping the contents of the db.sqlite3 file into a sql file using `sqlite3 db.sqlite .dump > $BACKUP.sql`.

For data recovery, the latest time stamped copy if the db.sqlite3 file will replace the one that was lost in the case of system failure.

### **3.6 Using Database Design or CASE Tool**

MySQLWorkbench – used in the creation of E/R diagrams.

Atom – IDE used for writing python for Django app

Django framework – For database and UI implementation in addition to database server management.

### **3.7 Other Possible E/R Relationships**

An earlier design of this schema had ANNOTATIONS saved only as children of SURFLINE\_REPORTS. The idea being that surfers would review how accurate the Surflines projection was of the surf conditions and comment directly under SURFLINE\_REPORTS. However, it is not unreasonable that surfers will want to log their sessions prior to Surflines reports getting added to the database. Before the automated process is set up to add Surflines data to this schema on a regular basis, someone will have to do that by hand. The schema was altered so that a foreign key to a SURFLINE\_REPORT was optional and that surfers only had to report the BEACH and READING\_INCREMENT of their session in creating a new ANNOTATION record. This way the association to the SURFLINE\_REPORT at that beach and time could be established later.

There was an alternate schema design which omitted the `READING_INCREMENT` table and instead had date and time attributes for each measurement table. Given the nature of this system, there are so many records that will have the same date and time that for data validation purposes, it made more sense to have them all reference a record in the `READING_INCREMENT` table with a single date and time than manage date and time attributes spread across 6 tables. The `READING_INCREMENT` table would also be easier for later queries. In the case that each measurement table got it's own date and time attribute, queries would have to manage both these attributes from each relation. Having measurements of different types taken at the same time reference the same record in the `READING_INCREMENT` table seemed to simplify both queries and validation.

## 4. Implementation Description

This section discusses the implementation of this system using Django to create user interfaces for interacting with a sqlite3 RDBMS. Additionally, section 4.1 described the the data dictionary. The DDL used for creating the schema itself is reported in Appendix B. Screenshots of the UI provided in section 4.2 are from a dev environment as this application has yet to be deployed. Section 4.2 also contains a description of how the fully built web application should behave and what functionalities it should include. Current implemented functionality supports the addition of Surfline data to the database by database admins and a front-end web interface which allows surfers to log their sessions.

### 4.1 Data Dictionary

In the following description of the schema, some attributes and tables may be named slightly different from how they were previously discussed. This is a result of integration between the sqlite3 database and the Django application layer. For example, “main\_model” is prepended to each table name because in the Django project, the schema is defined with the “main\_model” app. Additionally float fields have been registered as real fields to conform with sqlite conventions.

Following descriptions of tables take the form...

cid | name | type | notnull | df1t\_value | pk

main_model_reading_increment 0 reading_id integer 1  1 1 date date 1  0 2 time time 1  0	main_model_region 0 region_id integer 1  1 1 name varchar(45) 1  0	main_model_beach 0 beach_id integer 1  1 1 name varchar(45) 1  0 2 time_impact integer 1  0 3 min_recommended_skill_level integer 1  0 4 latitude real 1  0 5 longitude real 1  0 6 region_id integer 1  0
main_model_surfer 0 surfer_id integer 1  1 1 first_name varchar(45) 1  0 2 last_name varchar(45) 1  0 3 skill_level integer 1  0 4 favorite_beach_id integer 1  0 5 local_region_id integer 1  0	main_model_swell 0 swell_id integer 1  1 1 height real 1  0 2 period integer 1  0 3 direction varchar(3) 1  0 4 degree integer 1  0 5 growth_direction integer 1  0 6 beach_id integer 1  0 7 buoy_reading_id integer 0  0 8 reading_increment_id integer 1  0	main_model_tide 0 tide_id integer 1  1 1 height real 1  0 2 label varchar(10) 1  0 3 growth_direction integer 1  0 4 beach_id integer 1  0 5 reading_increment_id integer 1  0

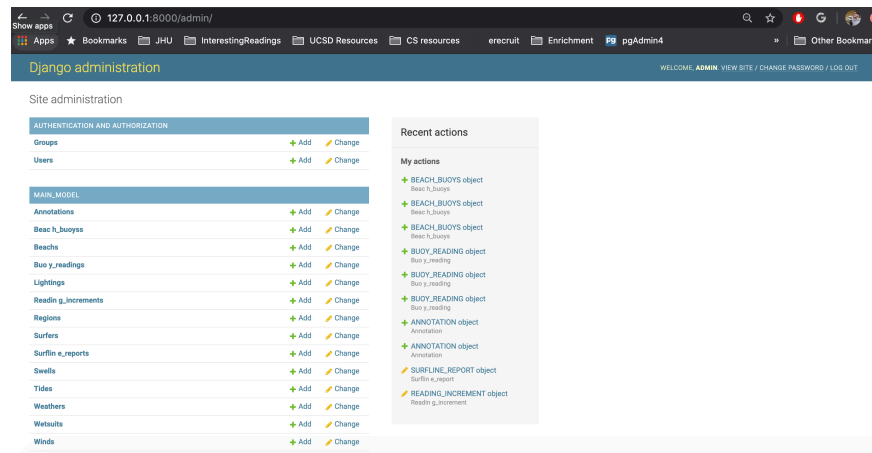
main_model_weather 0 weather_id integer 1 1 1 temperature integer 1 0 2 sky_condition varchar(15) 1 0 3 beach_id integer 1 0 4 reading_increment_id integer 1 0	main_model_wind 0 wind_id integer 1 1 1 speed real 1 0 2 direction varchar(3) 1 0 3 degree integer 1 0 4 beach_id integer 1 0 5 reading_increment_id integer 1 0	main_model_wetsuit 0 wetsuit_id integer 1 1 1 thickness varchar(3) 1 0 2 model varchar(25) 1 0 3 brand varchar(25) 1 0
main_model_buoy_reading_increment 0 reading_id integer 1 1 1 date date 1 0 2 time time 1 0	main_model_beach_buoys 0 id integer 1 1 1 beach_id integer 1 0 2 buoy_reading_id integer 1 0	main_model_surfline_report 0 report_id integer 1 1 1 min_wave_height real 1 0 2 max_wave_height real 1 0 3 beach_id integer 0 0 4 reading_increment_id integer 1 0 5 region_id integer 0 0 6 wetsuit_id integer 0 0 7 verdict integer 1 0
main_model_annotation 0 annotation_id integer 1 1 1 comment varchar(150) 1 0 2 observed_min_wave_height real 1 0 3 observed_max_wave_height real 1 0 4 beach_id integer 1 0 5 reading_increment_id integer 1 0 6 surfer_id integer 1 0 7 surfline_report_id integer 0 0 8 verdict integer 1 0	Main_model_lighting 0 lighting_id integer 1 1 1 first_light time 1 0 2 last_light time 1 0 3 sunrise time 1 0 4 sunset time 1 0 5 date date 1 0 6 beach_id integer 1 0	

## 4.2 Advanced Features

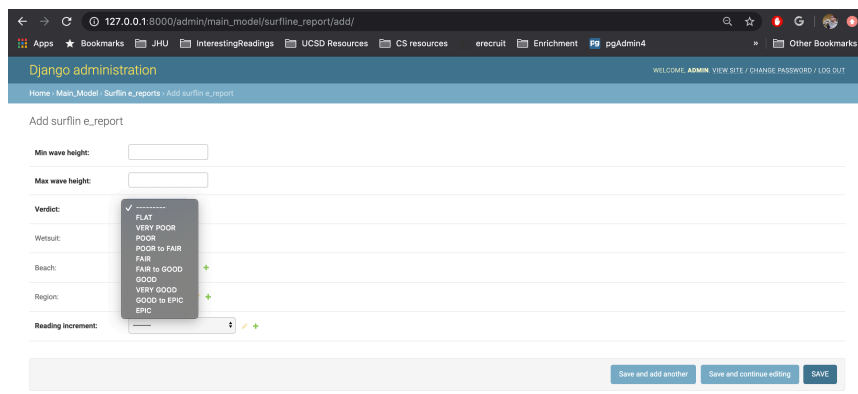
The system described up until this point was implemented as a Django web app. The code for this web app can be found here (<https://github.com/notmaurox/SurfUnderLine>). The Django implementation of this system allows for interacting with the data base via web pages. While the app has yet to be deployed to a publicly available server, screenshots are provided to demonstrate how surfers will log their sessions and how database admins will add Surfline data to the database prior to an automated system being implemented. Some initial investigation revealed that Surfline has publicly accessible REST API's, one for retrieving each of the surf metrics defined in this schema for a given beach.

### Admin Portal

The admin portal will be used by database administrators to register beaches, regions, and add Surfline reports and their associated metrics to the database. Admin accounts must be created manually via Django's createsuperuser command. Admin credentials are then used to log into the admin portal. Prior to an automated system which polls Surfline's REST APIs to populate the database, admins will need to register data for currently supported beaches manually.



*Admin portal homepage showing all tables in database schema*

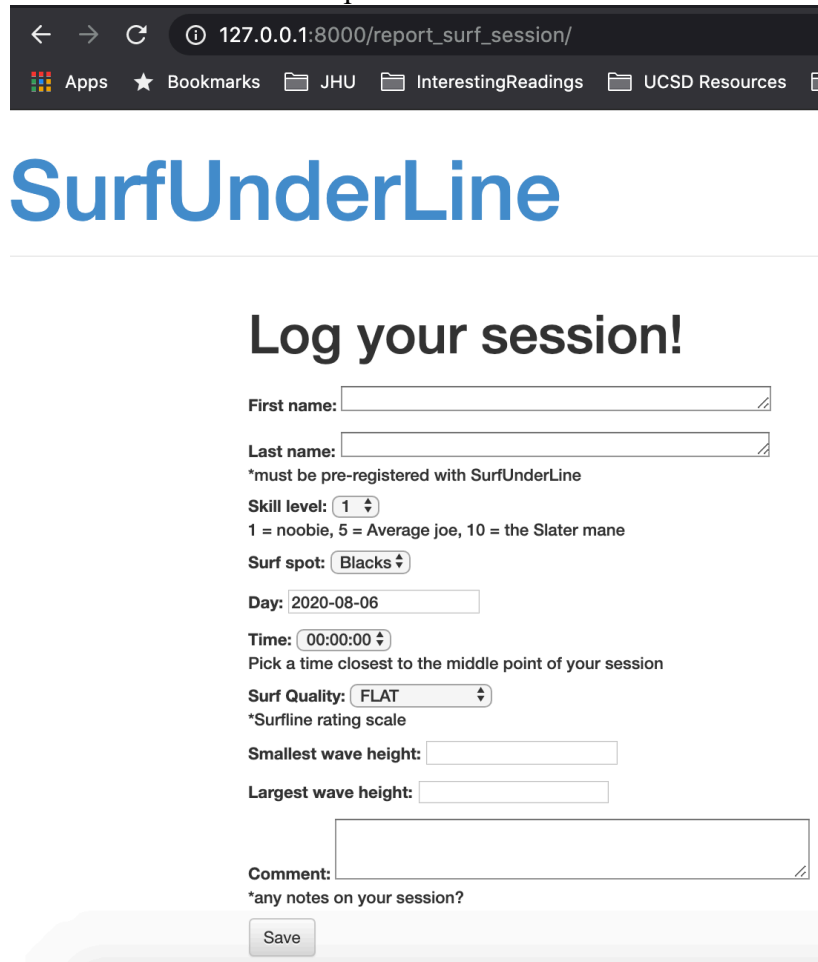


*Admin page for adding SURFLINE\_REPORT records to the database*

## Surfer Portal

Surfers have a non-password protected Django form for logging their surf sessions. This form is implemented in the Django application layer on top of the SQLite database. A strength of this approach is that fields from this form are validated by Django so SQL injection is not possible. Surfers will provide their first name, last name, and skill level. Additionally, they will be reporting a beach which they are logging the surf session at. The beach is selected from a drop down menu which is populated by the names of beaches already in the database. Database admins must add a beach to the database before a surfer can log sessions there. The combination of these fields will be used to look up existing records in the SURFER table. If there are no matches, one will be created. The default for the new SURFER record will be to set their favorite beach to the beach they are logging in the report, and their region to the region that beach belongs to. The rest of the fields on the page are the attributes required to create a new record in the ANNOTATION table. The user will provide the date and time of their session. This information will be used to lookup an existing READING\_INCREMENT record or create a new one. This UI provides the bare minimum to allow surfers to log their data. Back end triggers will be used to look up associated Surflin reports and add newly created ANNOATIONS to them as children. Furthermore, another trigger upon the addition of a new SURFLINE\_REPORT should look up the ANNOTATIONS logged for that beach at that time and attach them as children. These triggers should most likely be

implemented in the Django application layer so that better logging and error handling can be included. A screenshot of this UI is provided below.



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/report_surf_session/`. The browser's bookmark bar includes 'Apps', 'Bookmarks', 'JHU', 'InterestingReadings', and 'UCSD Resources'. The main content area features the 'SurfUnderLine' logo in blue, followed by a horizontal line and the heading 'Log your session!'. The form contains the following fields and controls:

- First name:** A text input field.
- Last name:** A text input field.
- \*must be pre-registered with SurfUnderLine**: A text label.
- Skill level:** A dropdown menu with '1' selected.
- 1 = noobie, 5 = Average joe, 10 = the Slater mane**: A text label.
- Surf spot:** A dropdown menu with 'Blacks' selected.
- Day:** A text input field containing '2020-08-06'.
- Time:** A dropdown menu with '00:00:00' selected.
- Pick a time closest to the middle point of your session**: A text label.
- Surf Quality:** A dropdown menu with 'FLAT' selected.
- \*Surfline rating scale**: A text label.
- Smallest wave height:** A text input field.
- Largest wave height:** A text input field.
- Comment:** A large text area.
- \*any notes on your session?**: A text label.
- Save**: A button at the bottom of the form.

*UI which enables surfers to log their sessions*

## 4.3 Queries

The following describes some common queries that will be of interest to database admins and surfers once the database has enough historical information to enable the study of surf trends and patterns. The guiding question to answer once enough data is captured in this system is as follows; how often do the surfers going to the beach agree/disagree with the surf quality that Surfline reported, and what types of measurements are common when these moments of disagreement arise?

### 4.3.1 Get all of a surfer's logged surf sessions

Input:

Surfer's first name and last name.

*SQL example where `SURFER.first_name` = "Nick" and `Surfer.last_name` = "Dante"*

```
SELECT * FROM main_model_annotation WHERE surfer_id = (
  SELECT surfer_id FROM main_model_surfer WHERE first_name = "Nick" and last_name = "Dante"
);
```

Output:

```
1|Occasional 4 foot sets were rare but there|2.0|4.0|1|1|1|1|5
```

#### 4.3.2 Find all wetsuits recommended for a region

Input:

Name of a REGION in the database

*SQL example with REGION.name = "South San Diego"*

```
SELECT * FROM main_model_wetsuit WHERE wetsuit_id IN (
  SELECT wetsuit_id FROM main_model_surflinereport WHERE (
    region_id = (SELECT region_id FROM main_model_region WHERE name = "South San Diego")
    or beach_id IN (SELECT beach_id FROM main_model_beach WHERE region_id = (
      SELECT region_id FROM main_model_region WHERE name = "South San Diego"
    ))
  )
);
```

Output:

```
1|3/2|BILLABONG|Revolution Pro Fullsuit
```

#### 4.3.3 Find the most popular beach for a region

Input:

Name of a REGION in the database

*SQL example with REGION.name = "South San Diego"*

```
SELECT name FROM main_model_beach WHERE beach_id = (
  SELECT favorite_beach_id FROM main_model_surfer WHERE (
    local_region_id = (
      SELECT region_id FROM main_model_region WHERE name = "South San Diego"
    )
  ) GROUP BY favorite_beach_id LIMIT 1
);
```

Output:

```
Blacks
```

#### 4.3.4 Find the average height for a specific beach by name

Input:

Name of a BEACH in the database

*SQL example with BEACH.name = "Blacks"*

```
SELECT AVG((max_wave_height+min_wave_height)/2) FROM
  main_model_beach LEFT JOIN main_model_surflinereport ON
    main_model_beach.beach_id = main_model_surflinereport.beach_id
  WHERE main_model_beach.name = "Blacks";
```

Output:

```
3.3
```



### 4.3.5 List all the swells associated with good surf for a beach

Input:

Name of a BEACH in the database

*SQL example with BEACH.name = "Blacks". It is important to note that a verdict with a score of 5 or better means the surf was good. Additionally, surfers are mostly concerned with height, period, and direction when they compare swells to each other. Two swells being identical except for their degrees isn't enough to classify them as two different swells. The count and group by operations are important to include in the SQL statement to get the count of a unique swell across good surf conditions. The more a swell shows up on good days, the more likely it is to be a driving force behind the good surf.*

```
SELECT COUNT(*), height, period, direction FROM main_model_swell JOIN(
  SELECT reading_id, verdict FROM (
    main_model_reading_increment JOIN main_model_surflinereport
    ON main_model_reading_increment.reading_id =
    main_model_surflinereport.reading_increment_id
  ) WHERE verdict > 4
) verdicts ON main_model_swell.reading_increment_id = verdicts.reading_id
WHERE main_model_swell.beach_id = (
  SELECT beach_id FROM main_model_beach WHERE main_model_beach.name = "Blacks"
) GROUP BY height, period, direction;
```

Output:

```
2|0.3|10|SSW
1|0.5|8|WSW
2|0.7|8|WSW
3|0.7|12|SW
1|1.2|6|WSW
1|1.8|6|WNW
2|2.6|7|W
```

### 4.3.6 Find all the instances when a surfer disagreed with the Surflinereported wave height by more than 1 foot.

Input:

Surfer's first name and last name.

*SQL example where SURFER.first\_name = "Nick" and Surfer.last\_name = "Dante"*

```
SELECT * FROM main_model_annotation JOIN
  main_model_surflinereport ON
  main_model_annotation.surflinereport_id = main_model_surflinereport.report_id
WHERE (-1 >= (
  main_model_annotation.observed_min_wave_height
+ main_model_annotation.observed_min_wave_height
- main_model_surflinereport.min_wave_height
- main_model_surflinereport.max_wave_height)
Or 1>= (
  main_model_annotation.observed_min_wave_height
+ main_model_annotation.observed_min_wave_height
- main_model_surflinereport.min_wave_height
- main_model_surflinereport.max_wave_height))
AND main_model_annotation.surfer_id
= (SELECT surfer_id FROM main_model_surfer WHERE first_name = "Nick" and last_name =
"Dante");
```

Output:

1|Occasional 4 foot sets were rare but there|2.0|4.0|1|1|1|1|5|1|4.0|4.0|1|1|||5

## 5. CRUD Matrix

Each column represents an entity defined in section 5.1

Each row represents a function defined in section 5.2

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14
F1		u	c											
F2				u	c	c,r								
F3				c	u	c,r								
F4		d	u	d	d		d	d	d	u	d	u	d	d
F5		r				c,r		c	c	c	c	c	c	c
F6	d	d	u	d	d		d	d	d		u	u	d	d
F7	r			r		r		r	r	r	r	r	r	r
F8		r	c,r	r	c	c,r								

### 5.1 List of Entity Types

E1: REGION

E2: BEACH

E3: SURFER

E4: SURFLINE\_REPORT

E5: ANNOTATION

E6: READING\_INCREMENT

E7: WETSUIT

E8: TIDE

E9: WEATHER

E10: BUOY\_READING

E11: BEACH\_BUOYS

E12: SWELL

E13: WIND

E14: LIGHTING

### 5.2 List of Functions

F1: Insert a SURFER

F2: Insert an ANNOTATION

F3: Insert a SURFLINE\_REPORT

F4: Delete a BEACH

F5: Add a set of measurements for a given time increment

F6: Delete a Region

F7: Study correspondence between Surfline Reports and Measurements for a beach

F8: Data added through /report\_surf\_session/

## 6. Concluding Remarks

In summary, this project was an exercise in iterative design. This schema went through many cycles of attempting to capture information from Surfline's UI, only to discover that something was missing which required revising the entire schema. Additionally,

relationships had to be altered so that data available at different times could be added and not get held up by the absence of what were once required parent entities. Also, this project uncovered the depth and seemingly endless detail one can spiral into when describing databases. It may be a consequence of choosing a complicated real-world system to model with this schema, but there were always more edge cases and alternate considerations which could have been expanded upon.

There is great potential for future development of this system. Some immediate action items being, deploying the Django app to the web, automating the data retrieval from Surflin REST APIs, creating a real system for users to register accounts with an email address, and writing front end data visualization apps so that historical data can be studied for trends.

Time will tell if any of these things actually get implemented but there is great value in such a tool which delivers all the features described above. If the surf stays flat like it has for the past month, one day users might be able to take advantage of a fully developed SurfUnderLine.

## Appendices

### Appendix A - Surfline UI screen grabs

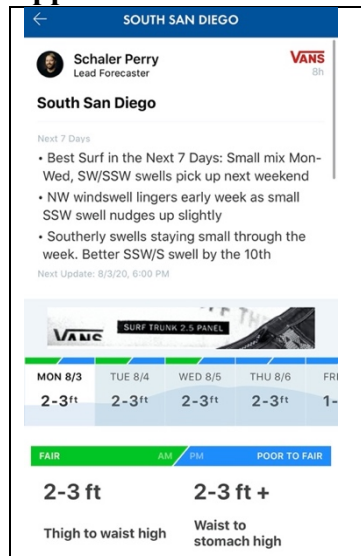


Figure 1: Region Report

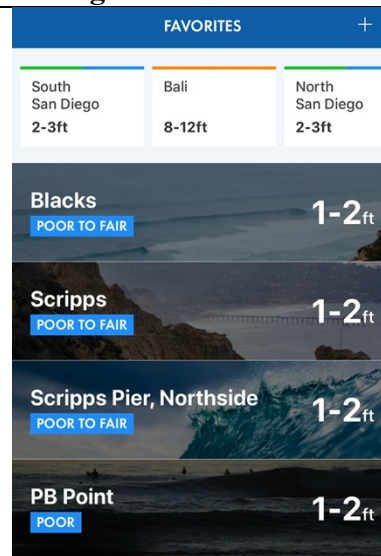


Figure 2: Beach Reports

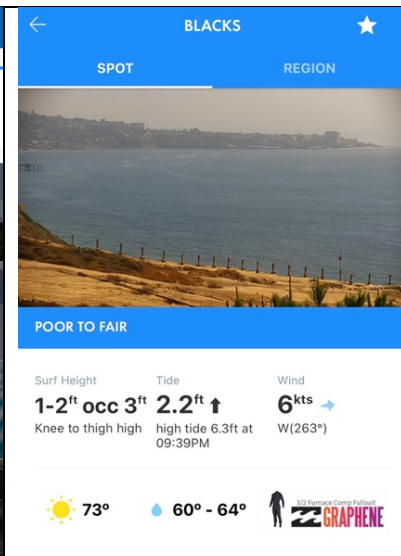


Figure 3: Beach report 2

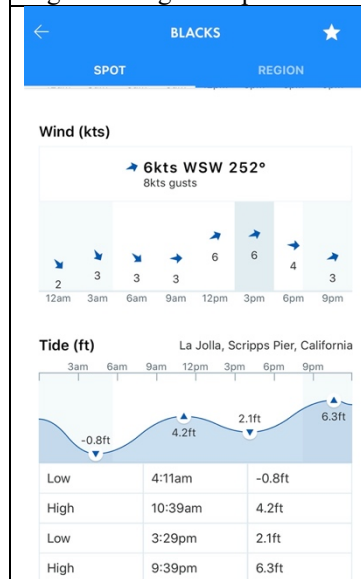


Figure 4: Report Metrics 1

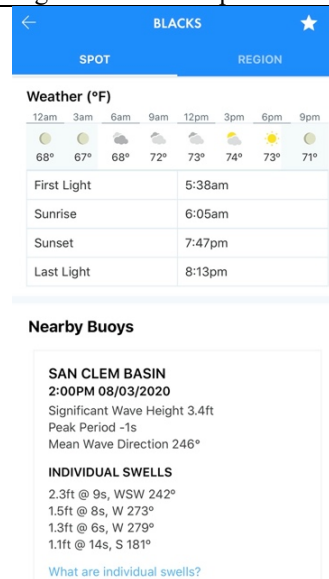


Figure 5: Report Metrics 2

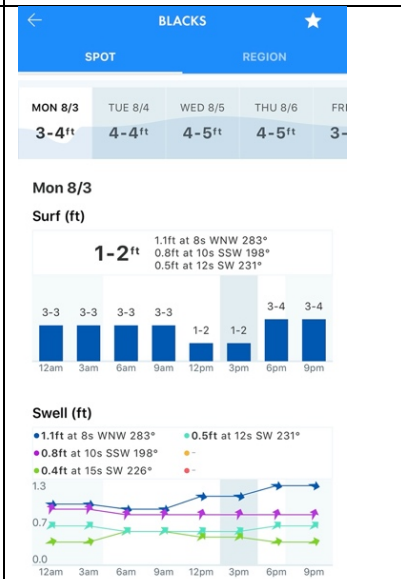


Figure 6: Report Metrics 3