Humbug Database Documentation

Lawrence Wang
OMI / Lawrence1994@gmail.com
August 27th 2019

1 Infrastructure

1.1 Software

The software of choice is PSQL v9.3.13.
The choice has been made for two reasons:
1) A relational database was preferred

A relational (i.e. SQL) database was advantageous to a NoSQL database (e.g. MongoDB) since it provides capability for complex queries; strict data schema; ACID compliance; cross-platform support; and a wide pool of existing UI options.

The main disadvantage in speed is mitigated with careful indexing upon construction of database.

2) Existing software

PSQL v9.3.13 has been used for the Humbug.ac.uk website. Software tools were already in place which made development easy.

1.2 Storage

There are two components to the Humbug Database - the PSQL database and the Audio Files.

The Audio Files are stored on humbug.ac.uk:/data/psql/ which is a data partition from the Florence drive. Steve Roberts believes that there are others ways to access the Florence drive - details TBC with Robots IT.
The PSQL database is running on humbug.ac.uk on localhost:5432.

2 Schema

The database schema was designed with a large number of tables to maintain tidiness and improve ease of querying.
The main division is between Label and Audio entities, where we expect a many-to-one mapping from Label to Audio, especially as the number of Labels from Zooniverse and Algorithms grow.
It should be noted that Location, Environment, and Device are sub-tables of Audio, whereas Mosquito and Labeller are sub-tables of Label.

2.1 SQL code reference

The code used to generate the Schemas and define constraints is as follows. A copy-able version can be found in the documentation folder with "Schema.txt".

```
   CREATE TYPE loc_type AS ENUM
('culture', 'cup', 'field',
'lab');

CREATE TABLE location(
   id serial PRIMARY KEY,
   country VARCHAR (50) NOT NULL
DEFAULT 'world',
   district VARCHAR (50),
   province VARCHAR (50),
   place VARCHAR (255),
   location_type loc_type,
   MAP_id VARCHAR(50),
   lat FLOAT,
   long FLOAT );

CREATE TABLE environment(
   id serial PRIMARY KEY,
   temperature FLOAT,
   humidity FLOAT,
   has_livestock BOOLEAN,
   has_rice BOOLEAN,
   has_forest BOOLEAN,
   has_irrigation BOOLEAN
);

CREATE TYPE record_method_enum
AS ENUM ('Audio', 'ABN', 'HBN',
'LT');

CREATE TABLE device(
   id serial PRIMARY KEY,
   method record_method_enum,
```

```
   mic_type VARCHAR(255),
   device_type VARCHAR(255)
);

CREATE TABLE audio(
   id serial PRIMARY KEY,
   path VARCHAR(255) NOT NULL,
   parent INTEGER REFERENCES
audio(id),
   record_datetime TIMESTAMP,
   record_entity VARCHAR(50),
   doc_path VARCHAR(255),
   env_id INTEGER REFERENCES
environment(id) NOT NULL DEFAULT
1,
   loc_id INTEGER REFERENCES
location(id) NOT NULL DEFAULT
1,
   dev_id INTEGER REFERENCES
device(id) NOT NULL DEFAULT 1,
   dashboard_id VARCHAR(50),
   zooniverse_id VARCHAR(50),
   name VARCHAR(255) NOT NULL
UNIQUE,
   legacy_path VARCHAR(255),
   sample_rate INT,
   length FLOAT
);

CREATE TYPE gender_enum AS
ENUM('Male', 'Female');

CREATE TYPE plurality_enum AS
ENUM('Single', 'Plural');

CREATE TABLE mosquito(
   id serial PRIMARY KEY,
   species VARCHAR(50),
   gender gender_enum,
   age INT,
   fed BOOLEAN,
   plurality plurality_enum,
   sound_type VARCHAR(50) NOT NULL
);

CREATE TYPE labeller_type
AS ENUM('Algorithm',
'Expert_Morphological',
'Expert_Molecular', 'Engineer',
'Zooniverse');

CREATE TABLE labeller(
   id serial PRIMARY KEY,
   name VARCHAR(50),
   type labeller_type
);

CREATE TYPE label_type AS
ENUM('Fine', 'Coarse');

CREATE TABLE label(
   id BIGSERIAL PRIMARY KEY,
   audio_id INTEGER REFERENCES
audio(id) NOT NULL,
   type label_type NOT NULL,
   mosquito_id INTEGER REFERENCES
mosquito(id) NOT NULL,
   labeller_id INTEGER REFERENCES
labeller(id) NOT NULL,
   fine_start_time float
CHECK((type != 'Fine') OR
(fine_start_time IS NOT NULL)),
   fine_end_time float CHECK((type
!= 'Fine') OR (fine_end_time IS
NOT NULL)),
   zooniverse_id VARCHAR(50)
UNIQUE,
   coarse_label BOOLEAN
);
```

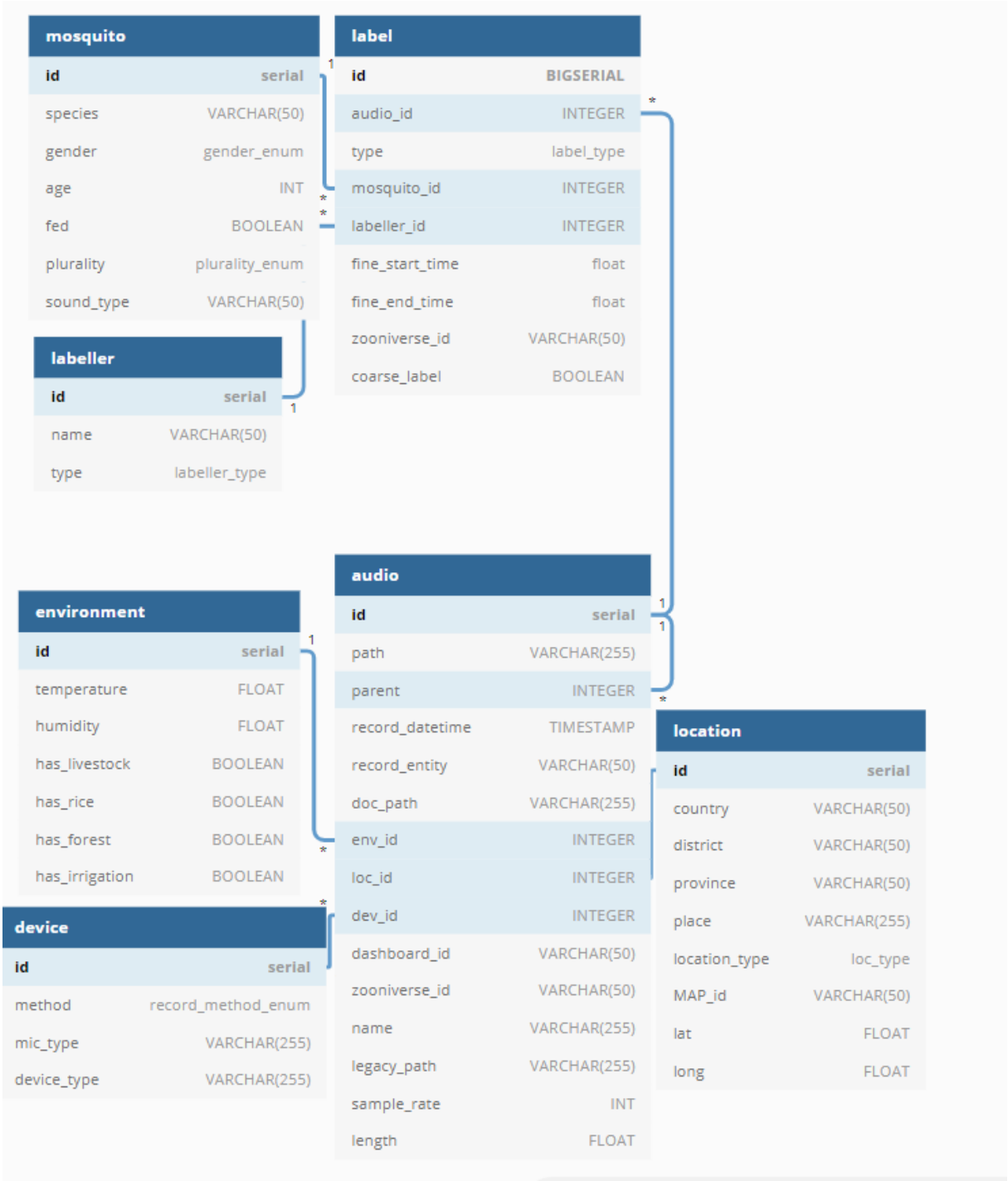

Figure 1: Schema Diagram, @dbdiagrams.io

3 Querying the Database

Humbug team members are all provided with read access to the database. In this method of access, the database can be queried with SQL commands. To access the database, one would need robots.ox.ac.uk and humbug.ac.uk accounts (please speak to Kit@robots.ox.ac.uk).
There is a Python tool where SQL commands can be sent to the database. This tool can be accessed after SSH-ing onto humbug.ac.uk with the relevant accounts and then using 'python3 /data/access/access.py *'.
Access.py takes one argument. In the form of *"text"* the text command is to be executed.
An example command could be *'python3 /data/access/access.py "Select * From audio Where id = 1;"'*. This command will show you the row of the audio file whose id is 1.
Another example command could be *'python3 /data/access/access.py "Select path from audio;" > paths.txt'*. This command will output all the paths of the audio files to *paths.txt*

3.1 SQL Query Examples

Copy-able versions of the following examples can be found in the documentation folder with "Query_Examples.txt".
1) Show frequency of audio files with varying number of fine labels

```
SELECT t.number_of_labels,
COUNT(t.id) FROM (SELECT
audio.id, COUNT(label.audio_id)
AS number_of_labels FROM audio
LEFT JOIN label ON (audio.id =
label.audio_id AND label.type =
'Fine' AND label.zooniverse_id IS
NULL) GROUP BY audio.id) t GROUP
BY t.number_of_labels ORDER BY
t.number_of_labels DESC;
```

2) Show total length of audio files whose location is Thailand

```
SELECT SUM(length) FROM
audio LEFT JOIN location ON
(audio.loc_id = location.id)
WHERE country = 'Thailand';
```

3) Show paths of audio files where a fine label exists

```
SELECT t.id, t.path FROM
(SELECT audio.id, audio.path,
COUNT(label.audio_id) AS
number_of_labels FROM audio
LEFT JOIN label ON (audio.id =
label.audio_id AND label.type =
'Fine' AND label.zooniverse_id IS
NULL) GROUP BY audio.id) t WHERE
t.number_of_labels > 0;
```

4) Show fine label start and end times where there is a mosquito found

```
SELECT label.id, audio_id,
fine_start_time, fine_end_time
FROM label LEFT JOIN mosquito ON
(label.mosquito_id = mosquito.id)
WHERE sound_type = 'mosquito' and
type = 'Fine';
```

4 Updating the Database

4.1 Write Access

Currently, write access of the database is kept only by Lawrence Wang. This will be passed on to the Post-Doctoral hire once the role is filled.
Administration of the database can be accessed through UI with PGAdmin or via PSQL/SQL terminal commands. These operations are only available to the administrators. Please contact Lawrence Wang for an admin account.

4.2 Recommended Upload Procedure

The recommended procedure for database upload is as follows:

A) Upload audio entries
    1) Insert location, device, and environment entries if does not already exist
    2) Upload database entries for audio files
B) Upload label entries
    1) Insert mosquito and labeller entries if does not already exist
    2) Upload database entries for labels

It is recommended to upload from csv files.
Resources:
https://www.trineo.co.nz/blog/2018/08/using-copy-in-postgres-for-importing-large-csvs
http://www.postgresqltutorial.com/import-csv-file-into-posgresql-table/
An example can also be found in the documentation folder with "Example_Update_loc.txt".