CattleLog: Modernizing Cattle Record Accessibility

Camoen VanWagner

Advisor: Professor Cheryl Resch
Department of CISE

November 20, 2019

Abstract

A significant amount of data is necessary to run an effective agricultural venture like the VanWagner family farm. Since the VanWagner farm specializes in raising and breeding cattle, farm employees regularly need access to cattle health records, treatment listings, identification details, genetic/heritage data, and reproduction information. For over 20 years, a subset of the most pertinent data has been printed out on a weekly basis and distributed to farm employees. It's certainly possible to search through these printed reports to obtain necessary data, but tracking down one record out of hundreds of rows is an inefficient process that could greatly benefit from being brought into the digital age.

The project detailed in this report, an Android app called CattleLog, was developed to solve this inefficiency by providing farm employees with an intuitive GUI for record look-up—this GUI allows the user to quickly locate a specific cattle record by entering an animal's unique attributes. Beyond development of just the Android app, existing farm records were processed, stored in a database format, and uploaded to a remote repository, where they could be accessed from within CattleLog. This preprocessing functionality was packaged as a reusable executable, called CattleLog File Processor (CFP), which allows database updates to be completed as often as necessary. Finalized versions of this project's results, CFP and the CattleLog mobile app, were distributed to the VanWagner family farm, where it has already become an indispensable resource for all employees.

1. Introduction

Historically, on the VanWagner family farm, it has been standard procedure to manually create weekly reports containing all pertinent cattle records, print out several copies, and then distribute the reports to farm employees. Generally, these documents are several pages long, each of which holds a table with tens of rows and over a dozen columns—utilization of these reports is unwieldy and inefficient but necessary if farm employees are to complete their work effectively. The subject of this paper is to summarize the development of a software system, named CattleLog, which aims to provide farm employees with much more convenient record look-ups via an Android mobile application. This paper also details the design of the application's back-end database and the development of a program that can preprocess the farm's existing (and future) cattle data. The software developed during this project has been beta tested in the field by the targeted end users, whose feedback was used to create a more robust,

user-friendly system. The results of this project, CattleLog and its associated data preprocessor, have quickly been adopted by employees of the VanWagner family farm and seamlessly replaced the previous method of record distribution.

1.1 Problem Domain

An often overlooked part of farming is the extensive number of records required to manage a healthy, productive herd. On a daily basis, farmhands need access to an animal's medical records (vaccinations, for example), logs of previous health events (births, breedings, surgical procedures), and a table of reproduction specifics (heat cycles, fertility data). Beyond these necessary bits of information, it can often be helpful to have easy access to an animal's genetic data, or, in the case of lost tags, a variety of identifying characteristics for each animal in the herd. The focus of this project, CattleLog, is to efficiently process these types of existing data (and handle future data updates) for the creation and maintenance of a well-designed relational database—this database can then be leveraged by a custom-built mobile application to provide farmhands with easy access to cattle records while in the field.

2. Technical Approach (Solution)

2.1 CattleLog Backend

The primary goal of this project was to create an Android application to allow user-friendly access to cattle records, but, to get there, existing data needed to be processed and converted to a usable format. Additionally, this data processing needed to be repeatable, to ensure that future updates to the records could be propagated through to the end-user application. To this end, a relational database schema was designed to hold all necessary data for the CattleLog mobile application. Furthermore, a program was developed in Python to handle all data processing, the creation and maintenance of the SQLite CattleLog database, and the automatic upload of this database to a remote repository.

2.1.1 CattleLog Database Design

The first step in the creation of the CattleLog database schema was to gather as much data as possible from the VanWagner family farm's existing records and identify potential entities, their attributes, and the data types associated with these attributes. It became evident early on that the primary entity would be "Cattle" (with one tuple for each unique animal on the farm), and that there would be two weak entities, "HealthRecords" and "Treatments", relying on

foreign keys associated with the "Cattle" table. An E-R diagram of the CattleLog database's final design is shown in Figure 1.

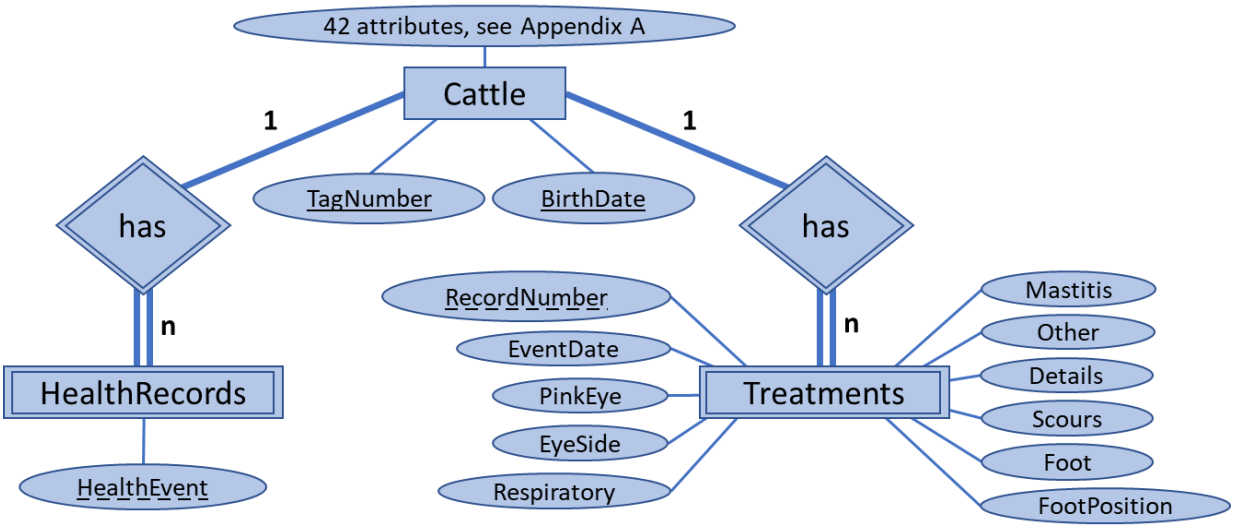

Figure 1. E-R Diagram of CattleLog Database. "Cattle" attributes are listed in Appendix A.

The identification of a unique primary key for each entity, especially the Cattle entity, was an issue during the design of the database schema. Animals on the VanWagner family farm have no truly unique identifier. On rare occasions, multiple animals may have same number displayed on their plastic ear tag, meaning that this attribute (TagNumber) couldn't be a viable primary key on its own. Though each animal does have a DHIID attribute that's guaranteed to be unique, it's possible that the small, metal tag displaying a particular animal's DHIID could be lost, which would require its replacement—unfortunately, this would also necessitate the replacement of the animal's recorded DHIID value in the database. Since the CattleLog database is abstracted away from and never directly modified by the farm's record-keeper, this type of update, while possible, would require the record-keeper to enter more information than their standard procedure usually entails.

To minimize any work required from the record-keeper, it was decided that animals would be uniquely identified within the Cattle table via a composite primary key consisting of their tag number and birthdate. While it's conceivable that two animals on the farm may share the same tag number, it's almost impossibly unlikely that they would also share the same birth date. However, if this situation were ever to arise, one of the conflicting birthdates could be modified by a day without substantially affecting the records, the duplicate tag number could be replaced, or the duplicate tag number could be recorded with an appended or prepended integer.

2.1.2 CattleLog File Processor

Cattle records on the VanWagner family farm are stored and updated in two different ways. Some record details (namely, all data required by the "Treatments" entity in Figure 1) are stored in a Microsoft Excel workbook (XLS format), but a majority of records are maintained via an outdated version of a herd management software called PCDART. Despite being more than 10 years out of date, this version of PCDART is still an enterprise-level piece of software, so development of an alternative solution for record entry was beyond the scope of a project spanning a single semester. Since the goal of this project was not to change the way records are maintained, it became necessary to take outputs from both PCDART and the mentioned Microsoft Excel workbook, process them, and generate a usable database to support the CattleLog mobile app.

PCDART is capable of outputting all required data (all "Cattle" and "HealthRecords" attributes defined in Figure 1) in comma-separated values format via the automated generation of three files. Therefore, the main technical hurdle was not obtaining the necessary data, but, instead, processing the three output files from PCDART in conjunction with an XLS file, ensuring that all processed data conformed to the defined database schema, and automatically creating or updating the CattleLog database. To make data preprocessing and database maintenance as simple and unobtrusive as possible for the end user, a Python executable, dubbed CattleLog File Processor (CFP), was created. This executable accepts the four discussed input files (for example inputs, see "Toy Database Files" in Appendix B), uses the pandas library to automatically strip out all data [2], ensures data validity and conformation to the database schema, creates a database file or updates an existing database file, and uploads the new database to a remote repository on the cloud. See Fig. 2 for a visual representation of CFP's program flow.

To create the best experience for the end-user (in this case, the VanWagner family farm's record-keeper), there is a significant amount of logic embedded within CFP's implementation to ensure data validity and handle edge cases. Due to space constraints, only the most important parts of the logic will be discussed in this report—however, a link to the full source code is available in Appendix B (see the GitHub repository for "CattleLog File Processor").

Upon startup, CattleLog File Processor provides the user with a series of textual instructions to guide them in selecting the appropriate input files. Courtesy of Python's Tkinter toolkit, CFP allows a user to input the four selected files (three CSVs generated by PCDART and

one “Treatments.xls” workbook) via a file explorer GUI [1]. As files are input, they are checked for existence and validity of format—if any of the selected files fail these preliminary checks, the program stops executing and prompts the user to try again with different input files.

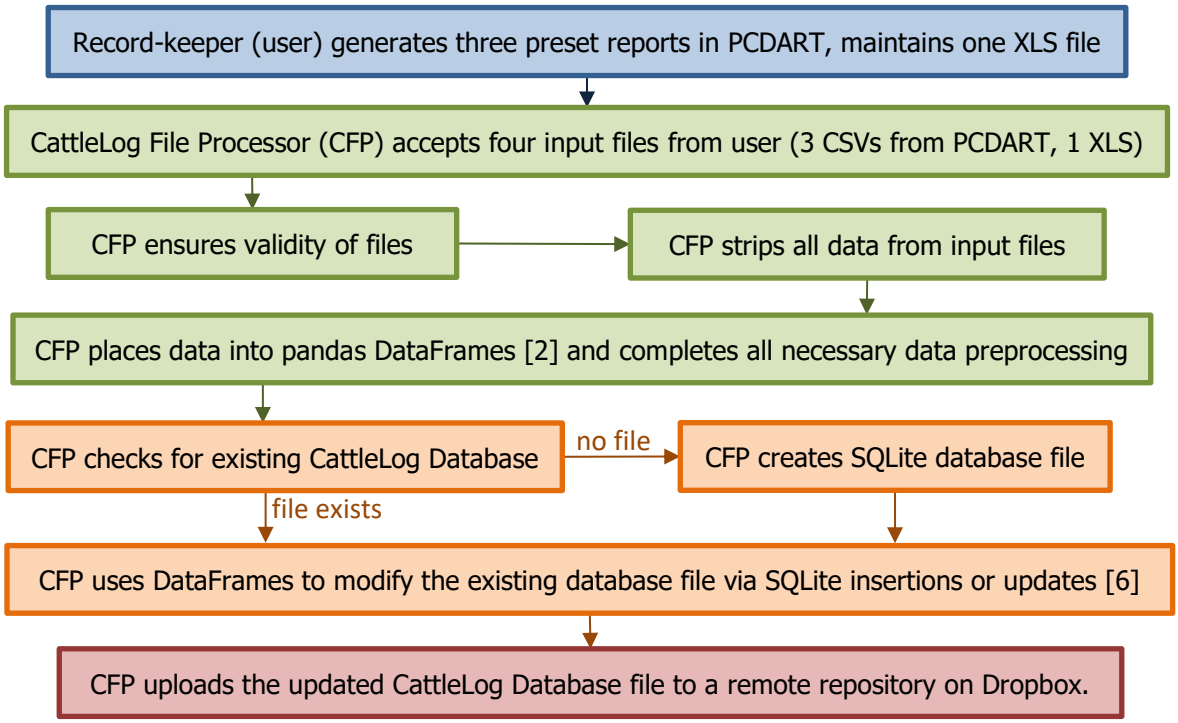

Figure 2. CattleLog File Processor program flow.

One of the input files, the manually updated “Treatments.xls” file, references cattle by tag number only (not by the combination of both tag number and birth date, as mandated by the database schema). To prevent creating additional work for the record-keeper, birthdates are associated with these tag numbers automatically. First, tag number and birth date combinations are recorded from the one of the PCDART input files. These combinations are stored in a Python dictionary, with each tag number serving as a key for its associated birth date value. Then, as data is stripped from the “Treatments.xls” file, the dictionary is used to link each tag number in the workbook with its associated birth date—this linkage is necessary to ensure that the stripped data can, eventually, be inserted into the “Treatments” table of the CattleLog database. Of course, since the “Treatments.xls” file is manually updated, implementing the appropriate logic to check for errors was important. If a duplicate tag number (a tag number that was already associated with a birthdate during CFP’s current execution) or invalid tag number (a tag number that has no associated birth date in the generated dictionary) is found, CFP stops executing, outputs a meaningful error message, and prompts the user to retry once the error is

fixed. For example, in the case of a duplicate tag number, the displayed error message reads, "{TagNumber} is a duplicate number. Add birthdate manually."

Out of all the details provided by PCDART's output files, the single most useful attribute in day-to-day work on the farm is probably "NextExpHeat", which is short for "Next Expected Heat Date". A cow is most fertile while in heat, so it's in the farm's best interests to breed an animal during this period of increased fertility. A cow exhibits many tell-tale signs when in heat, and having an animal's next expected heat date readily available allows a farmhand to verify whether or not an animal that's exhibiting signs is actually in heat.

Unfortunately, PCDART's output files only include the month and day, but not the year, of the next expected heat date. Furthermore, all of the NextExpHeat dates output by PCDART occur either on the present day or on a future date (never on a date prior to the day on which the output file was generated). For ease of querying within the CattleLog mobile app, CFP appends the current year to each of these dates during preprocessing. If the resulting date is prior to the current day, but within the past 4 weeks, the NextExpHeat date is adjusted by the addition of one or two heat cycles (21 days or 42 days, whichever is required). If, instead, the resulting date is prior to the current day, but not within the past 4 weeks, the NextExpHeat date is adjusted by the addition of one year. This logic assumes that CFP will be utilized within 4 weeks of generating the input files on PCDART (standard procedure is to generate these files on a weekly basis—three additional weeks were added as a user-friendly buffer). The implementation logic is better illustrated by an example, as shown in Table 1. Further implications of the addition of years to the "NextExpHeat" attribute will be discussed in section 2.2.2 of the report.

Date PCDART generated output file	Dec 15, 2019	Dec 15, 2019	Dec 15, 2019
Date CFP was utilized	Dec 19, **2019**	Dec 19, **2019**	Dec 19, **2019**
NextExpHeat (NEH) from PCDART	Dec-18	Dec-24	Jan-03
NEH with appended current year	Dec-18**-2019**	Dec-24**-2019**	Jan-03**-2019**
NEH adjustment	Dec-18-2019 is *before* Dec 19, 2019 but within previous 4 weeks. **Add 21 days.**	Dec-24-2019 is *after* Dec 19, 2019 **No change needed.**	Jan-03-2019 is *before* Dec 19, 2019 but outside previous 4 weeks. **Add 1 year.**
Final Result	Jan-08-2020	Dec-24-2019	Jan-03-2020

Table 1. NextExpHeat date adjustment logic. *Final result **must** be current date or a future date.*

After CattleLog File Processor has finished all data preprocessing, it creates a new SQLite database file if no such file already exists. Otherwise, it updates the local, existing database file. Finally, once the CattleLog Database file has been successfully updated, CFP uses a built-in API key to upload the database file (.db) to Dropbox [5], where it overwrites the previous version. This upload also results in the automatic update of the file's associated "last modified date" timestamp, which, as will be discussed in section 2.2.1, allows the CattleLog mobile app to detect that a new version of the database exists.

2.2 CattleLog Mobile Application

Once the CattleLog File Processor was capable of taking in input files, preprocessing their data, and generating or updating a usable SQLite database, development of the mobile Android app could commence. The main goals of the Android app, dubbed CattleLog, were to provide farm employees with an efficient method of record lookup via an intuitive user interface and display the requested records in an aesthetically pleasing, organized manner.

2.2.1 Remote Database Download

The first issue to resolve in CattleLog's development was the automatic download of the remote database during a fresh install and on subsequent database updates (due to the database file's small size, barely over 1 megabyte, mobile data usage was not a concern). To solve this issue, an initial "splash screen" activity was developed that compares the locally stored database timestamp to the timestamp of the remote database. The remote database timestamp is obtained via a Volley request sent to the Dropbox API—this requires the use of a Dropbox API key, which is built into the CattleLog Android application package [4,5]. If there is no locally stored timestamp, or the locally stored timestamp differs from the remote database's timestamp, a database download is triggered. When a download is triggered, if this is a user's first time using CattleLog, the app requests permission to access the device's storage. Once the user has granted permissions, the remote database file is downloaded to the mobile device's temporary storage. After this download is complete, the database file is copied into the app's internal storage, and the temporary file is deleted. Finally, the locally stored timestamp is set equal to the updated database file's timestamp.

2.2.2 Frontend and Backend Connection

To leverage data from the downloaded SQLite database file and expose it to CattleLog's users, the Room persistence library was utilized. Room allows the use of standard SQL queries

to obtain requested data; this is a significant advantage over the syntax heavy implementation that's required to directly manipulate an SQLite database [4,6]. To display data to the user, the results of SQL queries are attached to RecyclerView objects throughout the mobile application—these RecyclerViews display returned query results as a dynamically generated, scrollable list. This method of graphical display is used for all three entities in the discussed database schema.

In general, all of the SQL queries required by CattleLog are quite straightforward, aside from the "getNextExpectedHeatsPreset" query, which is used to generate a "Heats" list (see Figure 4 in section 3 for an example of the results). Since heat cycles are not exact, it's helpful to show all expected heats within a few days' range of the current day (before and after). The date range specified by the app's end users was "3 days before and after". However, recall that PCDART only provides NextExpHeat (NEH) dates that are on the current day or in the future. As a result, whenever the database is updated, all NEH dates prior to the current day are lost. To account for this, the "getNextExpectedHeatsPreset" query returns all NEH dates between 3 days before and after the current day, but also includes all NEH dates between 18 and 21 days from the current day (an offset of one heat cycle, to account for any "before" dates that were lost in the database update). See Table 2 for an illustrative example of this query's logic.

1	2	3	4	5	6	7	**Dates Returned by getNEHPreset Query**
8	9	10	11	12	13	14	A. Current day
15	16	17	18	19	20	21	B. 3 days before current day
22	23	24	25	26	27	28	C. 3 days after current day
29	30	31					D. A and B offset by 21 days (one heat cycle)

Table 2. Example of result dates returned by getNextExpectedHeatsPreset query.

3. Results

CattleLog's main features are presented in this section—see Appendix C for additional screenshots and a video demonstration of CattleLog's complete functionality. The main screen after startup (Fig. 3) displays a list of all cattle on the VanWagner family farm, ordered by tag number (denoted "Number"). Specific animals can be searched for by tag number, metal tag number, or barn name. Another list view (Fig. 4, screenshot taken on Nov. 15th), available under the Heats tab, displays all cattle with "next expected heat" dates returned by the getNEHPreset query discussed in section 2.2.2—in this view, results are ordered by next expected heat date, denoted "Heat Date", and animals can be searched for by tag number or service sire. Tapping on any record in either list opens a detailed overview for the specific animal (Fig. 5). The overview tab is scrollable, and displays all attributes stored in the "Cattle" entity table. The other two tabs,

"Health" and "Treatment", hold all of the animal's associated records from the "Health" and "Treatment" entities in the database (Fig. 6-7 in Appendix C).

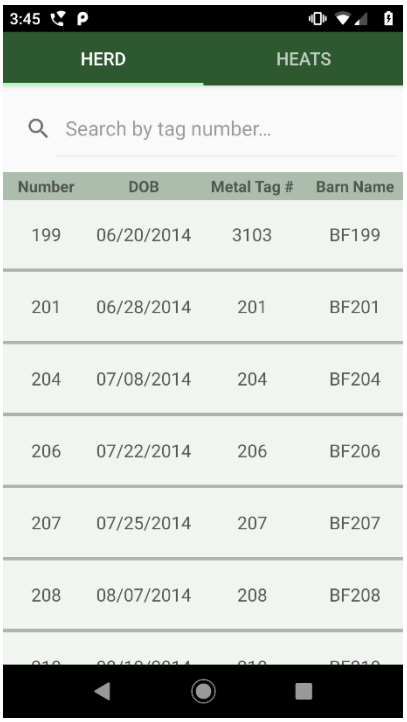

Figure 3. Herd Tab

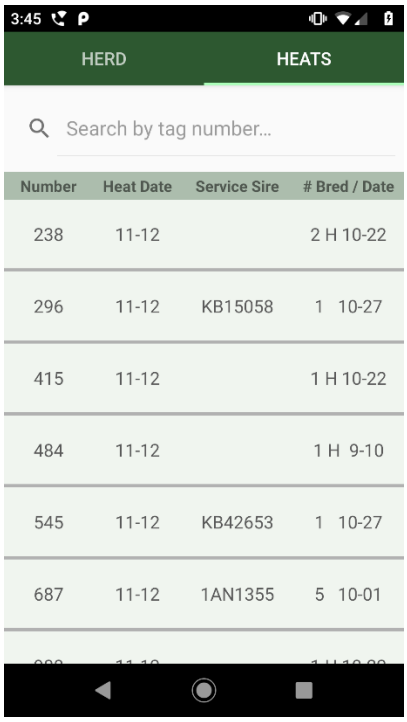

Figure 4. Heats Tab

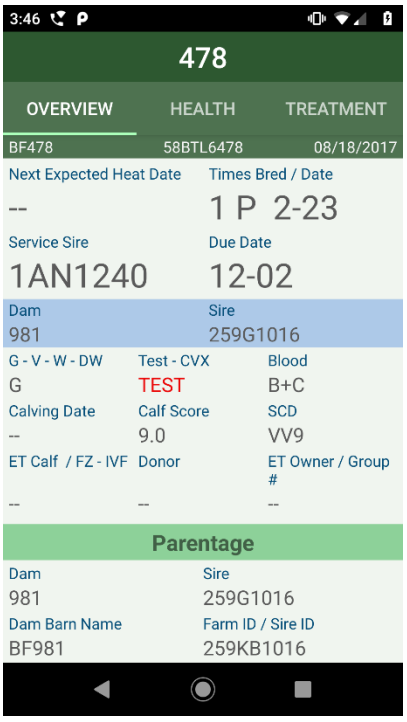

Figure 5. Animal Overview

As development neared completion, the application's eventual end users participated in testing a beta version of CattleLog. After minor modifications based on user feedback, the finalized CattleLog File Processor and CattleLog mobile app were distributed to the appropriate parties on the VanWagner family farm. The software system was readily adopted, quickly replacing the old method of record distribution. CattleLog is now utilized by farm employees on a daily basis to conveniently and efficiently review cattle records.

The developers of this solution, as well as CattleLog's end users, are satisfied with the outcome of this project, but the app portion, specifically, could benefit from additional testing on a larger range of devices and Android operating systems. The CattleLog mobile app is currently stable on user devices that were targeted specifically (via emulation) during development, but there were occasional hiccups when the app was first tested on newer phones with higher versions of the Android operating system (for example, "holding and waiting" for user-granted permissions was handled in a different manner). Looking forward, functionality of the mobile app will be monitored to ensure that CattleLog remains a viable solution for the long term.

4. Acknowledgements

The author would like to extend his gratitude to his advisor, Professor Cheryl Resch, for her agreement to oversee this (somewhat niche) project and her enthusiasm for the problem being solved. Additional thanks to Aleksandr Hovhannisyan (the author's group member in the completion of this project) and the members of the VanWagner family, for putting up with many questions about how records are logged, distributed, and utilized on the family farm.

5. References

[1] Anon. "Python 3.7.4 documentation", https://docs.python.org/3/, Python Software Foundation (as-of 27 Nov 2019).

[2] Anon. "pandas: powerful Python data analysis toolkit — pandas 0.25.1 documentation", https://pandas.pydata.org/pandas-docs/version/0.25/, pandas (as-of 27 Nov 2019).

[3] Anon. "Build Your First Android App in Kotlin", https://codelabs.developers.google.com/codelabs/build-your-first-android-app-kotlin/, Google Developers Codelabs (as-of 27 Nov 2019).

[4] Anon. "Documentation | Android Developers", https://developer.android.com/docs, Google Developers (as-of 27 Nov 2019).

[5] Anon. "Dropbox for HTTP Developers - Dropbox API v2", https://www.dropbox.com/developers/documentation/http/documentation, Dropbox (as-of 27 Nov 2019)

[6] Anon. "SQLite Commands", https://www.sqlitetutorial.net/sqlite-commands/, SQLite Tutorial (as-of 27 Nov 2019)

Appendix A – 42 Attributes Belonging to the Cattle Entity in the CattleLog Database

DaysSinceBredHeat	DaysTilDue	GroupNumber
TempGroupNumber	ReproCode	TimesBred
Breed	UsrDef1	UsrDef2
UsrDef3	UsrDef4	UsrDef5
UsrDef6	UsrDef7	UsrDef8
UsrDef9	UsrDef10	DaysTilNextHeat
BarnName	DHIID	DamIndex
DamName	SireNameCode	TimesBredDate
DateDue	ServiceSireNameCode	NextExpHeat
AgeInMonthsAtCalving	DonorDamID	FarmID

DamDHI_ID	PrevBredHeat1	PrevBredHeat2
PrevBredHeat3	WeightBirth	WeightWean
WeightBred	WeightPuberty	WeightCalving
DaysinCurGroup	DateLeft	Reason

Appendix B – Source Code

These repositories hold the complete code for both the backend and frontend aspects of the project, albeit with the redaction of all API keys, credentials, and private cattle records. The CattleLog File Processor repository includes sample files for a representative toy database. The mobile app repository, if compiled and executed, is linked to the same toy database.

- CattleLog File Processor: https://github.com/Camoen/CattleLog-File-Processor-Public
    - Toy Database Files: https://github.com/Camoen/CattleLog-File-Processor-Public/tree/master/Project%20Files
- CattleLog Mobile App: https://github.com/Camoen/CattleLog

Appendix C – CattleLog Mobile Application Demo, Additional Screenshots

CattleLog mobile application video demonstration: https://youtu.be/n9yGcRzhoyQ

Figure 6. Health Tab

Figure 7. Treatment Tab