Uppsala Universitet

Project in Computational Science

Constructing a database System

UPPSALA UNIVERSITET

Authors:
Gustaf Andersson
Li Ju
Linus Kanestad

Date:
2021-01-22

Abstract

This report covers the structure of a constructed database for data collected in two farms, containing information about cow health, milking and barn position. Each farm holds over 200 cows. Position data from each cow is collected each second resulting in around 17 million records, corresponding to approximately 1 Gb of data, every day. This information is used by a research group to, for instance, investigate social behaviours and track disease transmission. The database was based on MySQL to provide stability and maintainability. The table design choices were based on efficiency and in cooperation with researchers from SLU. The design revolves around a central cow information table connecting the other tables using cow id and timestamps as a primary key. Fetching position data uses a mapping table to connect a requested cow id to the correct tag, containing the position data. The resulting database system has been proven to insert 47.4 million records in under 30 minutes and query 271 thousand records in under three seconds.

Contents

1 Introduction

The common denominator for almost all research, is that it is based upon data. Data in the form of measurement values, compilations of information and more. In some cases the size of the data is small, e.g. a collection of numbers that can be stored as writing on a sheet of paper or as cells in a spreadsheet. In other cases the data consist of larger quantities, perhaps millions or billions of measurement records increasing with each passing day. In this case the data can no longer be stored on a sheet of paper. The data can still be kept within spreadsheets, but as the number of records increases, the data gets more demanding to view, process and analyze. Storing all records locally in this manner would also imply that all users must store a local copy of the data. To solve this problem, the data can be stored in a database. The database can provide data structure and storage whilst allowing the users to fetch a selection of data. This report will cover the construction of such a database and a system providing requested services.

1.1 Background

A project group from the Swedish University of Agricultural Sciences (SLU) conduct research on cows. The research project is in collaboration with a cow farm in Sweden and one in the Netherlands, where data from the cows is collected. The collected data consists of milking information, health information and position data. This information is used by the research group to investigate social behaviour such as dominance and avoidance or physical phenomenons such as disease transmission and milk production. The amount of collected data is increasing and the research group has decided a database is to be constructed.

1.2 Problem description

The research project at SLU involves capturing position data from cows within barns, as well as storing information health and milking information of cows. The farms hold between 200 and 300 cows. The position data is sampled each second resulting in large data files. The purpose of the project, addressed in this report, is to manage this data and to effectively store it in a database. The data must then also be accessed by the involved researchers using effective queries, returning requested data. The current methods of handling the data is to download and store files locally.

The challenges in this construction revolve around two issues. The varying structure of the different data types and matching a measuring tag to the correct cow. Information data is uploaded once every week and each file type is structured differently. This requires pre-processing and sorting in collaboration with project researchers. Matching tags to each corresponding cow will have to be done over a time interval to allow for position data queries. The challenge is to consider removal and switching of tags.

The data from Swedish farms consists of eight file types, making up three categories; position data, info data and milk data. The files are illustrated in Figure 1.

Position data consists of the position information of each cow for each second, recorded from their tag. There exists four types of position data, filtered position (FA), clustered position (PC), accumulated hourly activites (PAA) and concluded activities (PA), all containing different types of data. The position data is the largest data type that will be inserted into the database.

Milk data consists of two files, one containing milk time and milking station and the other

containing daily produced volume. The difficulty in storing the milking data is to match each milking to the right corresponding date of the milking and produced volume. Furthermore the data contains deviations, redundant and missing data due to farm routines that need to be handled.

Info data Info data consists of health information and general information for cows. General information of all cows are tracked. Treatment information are stored about cows which have been or are being treated for disease or injury. The difficulties in managing the info data involve tracking references between cows and tags whilst navigating through messy file structures containing unexpected indents and blank spaces.

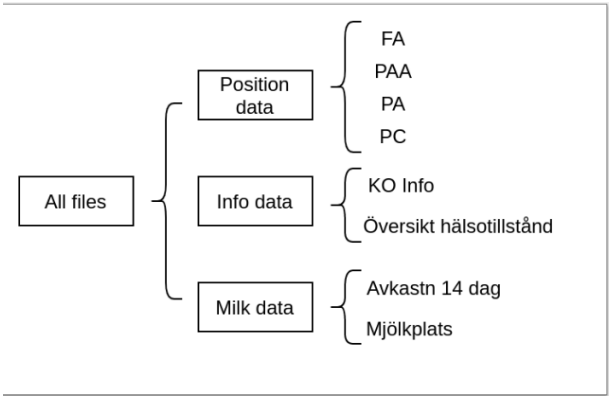

Figure 1: The files provided by the Swedish farm are displayed and split into categories.

1.3 Goals

In collaboration with SLU, the following set of goals set to be achieved within the project:

- Storage:
  - Store data from both Swedish and Dutch farms separately
  - Positioning data, identified by tag ID
  - Milking and cow info data, identified by cow id
  - Mapping table, to keep tracking mapping relation between cow and tag
- Querying:
  - Web App: querying data from a webpage to get text files with sufficient user interface choices
  - Functional API: querying data with python APIs and to get dataframe directly
- Maintenance:
  - Regular check to find the broken tags
  - Regular check to show meta data on web app

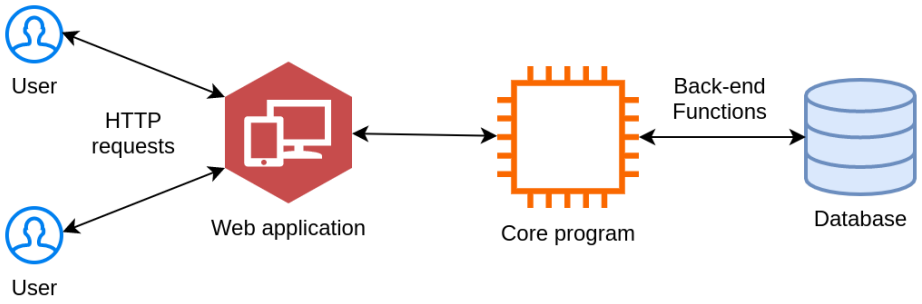

Figure 2: An overall structure for the database system.

2 Design and Workflow

2.1 Overall structure of the system

The structure of the database system is designed based on the set of goals, presented in Section 1.3. The architecture consists of a database, a running python-based server for a web application and a series of back-end python-based functions to communicate with database. Users can access the web application to insert data files, view the status of the database and request data. The database is also potential to be accessed through a python API. The structure is displayed in Figure 2.

2.2 Front-end

The front-end checks for illegal entries in user inputs on the web application and formulate instructions for the back-end. To make the application flow seamlessly, the front end provides detailed user feedback to the user in order to ease the use of each application.

2.3 Back-end

The back-end is python-based and its main functionality is to handle data preprocessing, insertions into and extractions from the database. The back-end provides a set of functions for communicating with the database. The functions are called from the front-end or from the API and will run on a separate thread.

For the data from the Swedish farm, there are three main processing types to be made: 1). sorting out unstructured health information data, 2). matching milking data to the correct dates and 3). matching cows to its corresponding tag.

The data from Dutch farm require tag matching and extracting columns to the correct database tables. All that data, besides position data, is uploaded once each month and is well structured. This simplifies the processing since the informative data in the same structure as it is uploaded.

3 Implementation

3.1 Database

3.1.1 SQL vs NoSQL

One of the first and central design choice is whether to use Structured Query Language (SQL) or Non-Structured Query Language (NoSQL). SQL is used for handling Relational database management system, where data is stored with relations in tables. NoSQL is a non-relational database management system, allowing storage of unstructured data. Unstructured in the sense of not having the relational table structure. NoSQL might then be faster and offer a more dynamic database. We considered SQL to be more robust and similar to the structure of the raw data file. Another aspect, is maintenance of the database. SQL, in constrast to NoSQL, is established and well documented. Since the database is to be maintained by the SLU research group, SQL might be the better choice. NoSQL can be faster, but does not provide the security of the robust structure of SQL[1]. We chose to use SQL due to its robustness and maintainability.

3.1.2 Table design

The second issue for the database design is to construct the tables for efficient storage. Efficient storage in the sense of reducing redundant data and allowing fast querying. The final table design is shown in Figure 3. The process of designing is described below.

The central part of the design is to decide on primary keys to use as identifiers to link information to each respective cow. The main identifier is decided to be the cow id together with a timestamp. We construct a central *cow info* table containing information such as status, lactation number and group affiliation with cow id and update timestamp as primary keys. It is then used as a central node for querying, connecting cows to tables containing health information, milking information and position data.

The information data from the Swedish farm consists of health information, calving status and insemination information. This data is currently saved in various files with some overlapping redundant information. After interpretation and communication with the SLU research group representatives, two tables are constructed separating active (i.e. milkable) cows and cows that are either dried off or chosen for slaughter.

To store milking data, we decided to save each milking as a record. Each record then contains a cow id, a timestamp, at what milking station the milking occurred. The initial plan was to include daily volume produced, but due to uncertainty in what volume record corresponds to what day it was not included. The research group on SLU has been notified of this problem and is communicating with the farmers to sort out how the milking records are collected.

The position data are collected from tags attached to the cows. Each tag is identified by its tag id and is removed and switched throughout the life of a cow. This needs to keep tracking on which cow is wearing what tag. We then construct a *mapping* table, containing information about when and what tag is put on. When the user queries position data from one or more cows, the *mapping* table is used as a middle hand connecting each cow to one or more tags. The information regarding which tag a cow is wearing is updated weekly and if a cow were to switch tag or have it removed, the *mapping* table is updated. If a cow is dried off, the tag is

Figure 3: The complete schema containing the data from the Swedish farm. Fields in bold are keys of the table. Fields in light blue are foreign keys referring to other tables and blue lines indicate which fields are foreign keys referring to. All foreign keys are one to one mapping, noted around blue lines.

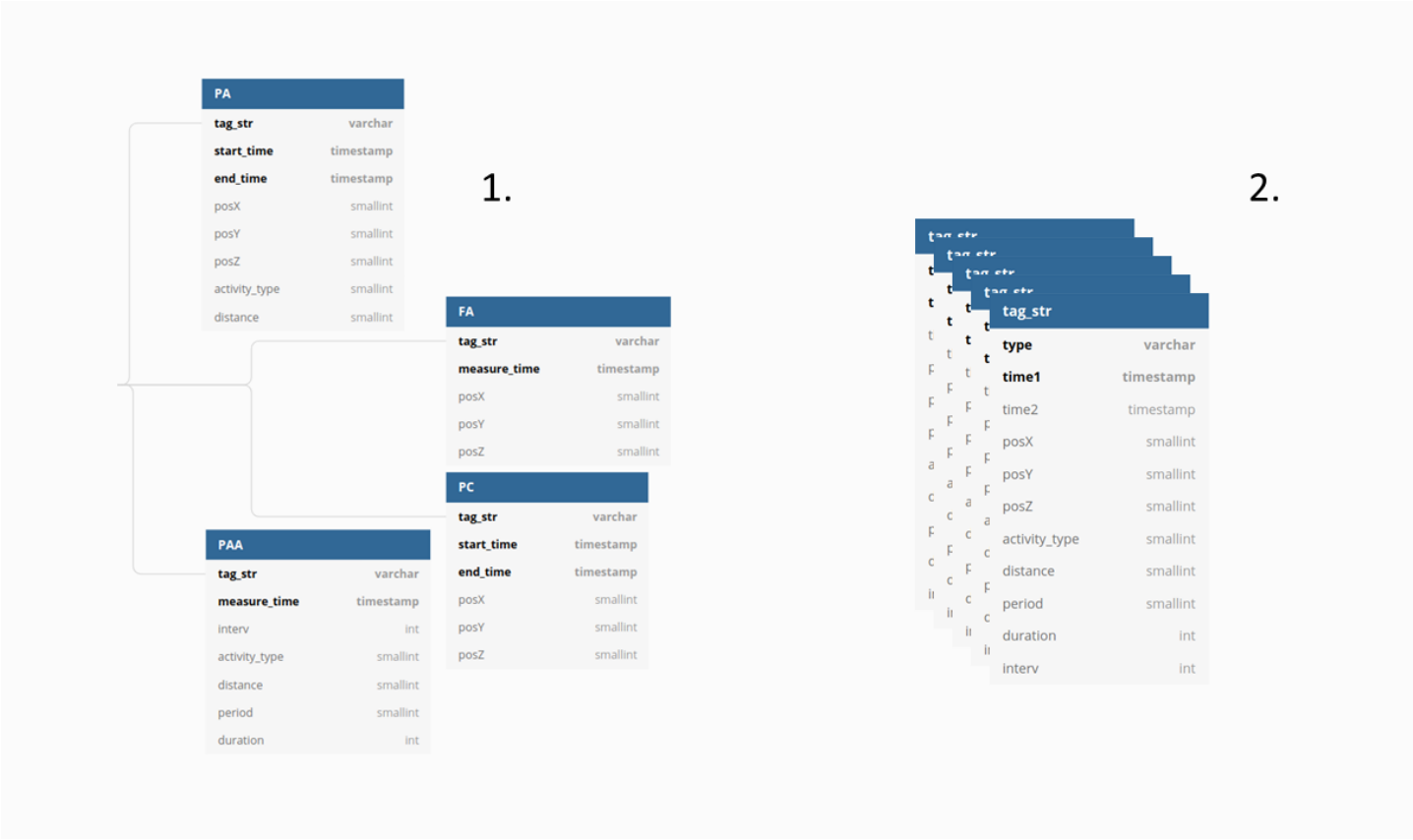

Figure 4: The two schema alternatives for position data storage.

considered as removed.

Position data are split into 4 types: FA, PA, PAA and PC data. The different types contain various information, such as raw position data, noise reduced data and tag interactions. Position data make up the majority of the storage that the database requires. which take approximately 1 Gb for each day. To store this data efficiently, two alternatives are designed. The database schema of a database is its structure described in a formal language supported by the database management system.[2]. Schema for both designs are shown in Figure 4. In the first design, position data are stored in four tables, according to their type. In the second one, data with same tag ids are stored in a separated table regardless of their types. The first design is expected to gain more space efficiency while the second one should be able speed up querying and inserting. All tables for position data have their primary keys as tag ids and timestamps.

To decide which schema to use, we conducted performance benchmark tests, querying position data of various quantity. The tests were in favour for the first schema, separating the four file types. The results from the measurement are shown in Figure 8.

3.2 Back-end functions

3.2.1 Data Preprocessing

The data from the farms is uploaded in csv-, txt-, xml- and xlsx- files. These files vary in structure. Some offer duplicate information and others need to be interpreted and processed.

Position data The position data is by far the largest datatype in our system. The position data consists of 3D-coordinates for every tag for each second. Files containing position data is uploaded to the system every day. The position information is extracted from filetypes FA, PA, PAA and PA, providing different kinds of position data, such as raw position data, interaction data and noise reduced data. Each record in the files is inserted into the database, separating the four filetypes. Each record contains a tag ID and a timestamp, working as a primary key for finding and selecting requested data. The tag ID corresponds to a tag. The information of which cow is wearing what tag is extracted from the file "KO info" (see Figure 1) and stored in the mapping table. The mapping table is updated weekly when the KO info file is uploaded.
Position data files are well-formatted and neat so that reading and preprocessing part is simple and straight forward. However due to the great number of records, to ensure the performance of the function over millions of records, Numpy functions are used instead of plain loop.

Milking data Milking data files are uploaded once a week, containing information of the past week or two. The Milking data is divided into two files; *Avkastn 14 dag* (daily volume production) and *Mjölkplats* (time of milking and milking station). Both files are uploaded weekly. The milk production file contains milk volume produced by each cow on 15 previous days. Although these files does not fulfill any identified pattern, making it impossible to accurately match each record to its corresponding date. This issue has been brought to the attention of the research group on SLU and needs to be resolved before inserting the data into the database.

For the milking station file, the data is easy to interpret, but there are some errors and deviation caused by missing data. This has to be handled. If an abnormality was identified, such as two evening milkings in a row, the abnormal record and every subsequent record was ignored. This was done to avoid shifted data causing mismatching. An exception was made for cows about to be dried off. The routine, performed at the farms, for drying off cows include only milking the cows on Monday, Wednesday and Friday mornings until it is considered dried off. According to disease transmission researchers in the project, cows in the process off being dried off are interesting and therefor its data is important to include [3]. Therefor cows milking records displaying consecutive morning milkings are kept and matched to the corresponding date.

Info data Info data of cows are updated weekly. Basic information about all cows is recorded in files with prefix "KO Info". Only cows which receive / have received some health treatment have their health data recorded in files with prefix of "Översikt hälsotillstånd". Only cows which are dried off or to be slaughtered have insemination information, and these data are recorded in "Översikt hälsotillstånd" files as well. The two info files are messy, with unexpected redundant spaces, unformatted nested table and redundant information, which is the main challenge for us to well format them. To solve this, we read files line by line and format information word by word. Fortunately info data files are of small sizes, so this does not lead to much performance loss.
Moreover, besides info data, the mapping table is maintained by Info data as well. The info files are scanned for two specific events; change of tag id and/or a cow being dried off. This results in the mapping table being updated with a new record containing the cow id, the new tag id and a start time (when the new tag is put on). The previous record of the updated is given an end time when the cow is dried of or when a tag is switched.

3.2.2 Data insertion

Insertion component of the system is designed to be modular-based structured with more maintainability and flexibility. An abstract class named insertor is defined. To implement an insertor, the class should include two class variables, table name and table fields, and two class methods, convert and insert. Method convert is responsible to convert preprocessed and structured data to proper format that can be inserted into database directly and method insert is responsible to connect with database, generate SQL statements and run them to insert input data into database.
Specially, for position data, the challenge is that the number of records are enormous (up to tens millions per file). Converting and inserting records by looping records one by one is low efficient and time consuming. To speed up converting, vectorized functions[4] are used so that the conversion can be done with a single operation by mapping the convert function to the entire data using Numpy functor. To speed up inserting, MySQL bulk insertion[5] is used to insert records into database in batch.

3.2.3 Data querying

Three query functions are offered, for information query, position query and milking data query. All constraints will be passed via arguments from front-end server (e.g. status, cow id, group id, data type, etc).
For information query, three types of data can be requested including basic information, health information and insemination information. Because of the update frequency of information data is low, and each record present the information about a cow of last entire week. If doing query with requested start date and end date directly from database, some records may be ignored. To get all records that cover the requested time interval, we use the start date as the date of a week before requested start date, and end date as requested end date.
For position data, the start date/time and end date/time are precise so they can be used to do query directly. The challenge of position query lies on key converting. When users querying position data, cow ids will be specified. However, unlike information tables and milking tables, cow ids are not keys nor fields of position table. Instead, tag id is the key of position tables. A conversion must be made between keys. What makes problem more complicated is that a cow may carry different tags on different time period and vice versa. Therefore, for each cow id requested, we maintain a list of tag the particular cow carries with its valid time interval. The intersection of requested time interval and each element in the list is finally the valid time intervals and tag ids for each cow that are actually used for querying. Multiple query statements are generated and all results are merged. Using this approach, all requested cows are traversed and queried.
Milk data are simple and well structured in database. Requested query and actual query arguments are basically the same.
All queries result in a .csv file containing all requested records. The header of the .csv files are specified by users when requesting query. If no records are fetched, a plain text file is generated with the text "No record is fetched. ". The file can be copied directly from server, or be downloaded via a browser.

3.2.4 Meta data management

To keep tracking the status of the database, files that have been inserted and avoid duplicated records in the database, meta data management is done as well by maintaining a log file. Each time input files are requested to be inserted into the database, the manage function checks if these files exist already in the log file, if yes, then these files are not inserted to avoid duplication of records and if no, insertion functions are called. Only if files are successfully inserted, their file names are saved in log file. With the log file, users can also detect which files are missing and remain to be inserted. To check the status and storage of database, a function is defined to return sizes and number of records of all tables.
With these meta data, users can easily manage files and check system status.

3.3 Front-end

The choice of environment to develop the front-end in were a pragmatic decision. Since the team have minor experience in front-end development, using a framework that is easy to get started with was paramount. Therefore, we decide to use the Django web development framework, which uses the familiar Python language to develop its applications. Also, Django has well written documentation to guide i its development [6]. Since Python also have a vast support for extension such as data processing and even SQL communication, it became a natural choice to use Django to create the web application.
When a Django project is initiated, it comes with a light-weight web-server for fast, early development. Django creates a skeleton of Python scripts when the project is created that is then extended by the developer, which convenient for a structured start when developing larger projects.

The main goal for the front end part of the project is to pass user input arguments to the database in the back end. To do this, each HTML-template that is made for user engagement includes at least one button that can issue a Http-POST request. Since the web server listens for such requests, we can use the *views.py* python script from the django framework to catch this POST-request which contains the arguments from the user that has been selected. Examples are shown below. These user inputs are then passed to the back end part of the project and handled accordingly.

3.3.1 Front-end: General structure

The base of the web application is made from a bootstrap (CSS and JavaScript) framework which in this case uses a menu bar located to the left for navigation throughout the site. The site also uses a top-bar for more navigational options. The left menu and the top-bar is static throughout the whole site, only the "container", the main part of the site, changes when navigating.
When one enters the site, the users is welcomed by "Welcome to Cowsite" and short informational text on how to navigate and use the APIs from Section 3.4.

3.3.2 Front-end: Storing data

The data can be inserted into the database in two ways. Either manually on the web application by selecting and uploading a file or automatically by copying files to the directory */input_files/*.

For both methods the files are processed in the same way. The filename is read to identify what kind of information is uploaded. Faulting files, duplicates or already processed files will not be inserted. The different filetypes are then processed as described in section 3.2.2.
The user selects what kind of data that is to be inserted by clicking the respective button. A POST-request is then created containing instructions for the back-end for what files to look for by calling the respective file scan functions.
The user receives a feedback message on the page confirming that the initiation of uploading data to the database was successful.

3.3.3 Front-end: Query data

To retrieve data from the database, the user can chose the *Swedish Data* tab in the web application. The user then selects what kind of data that is to be requested. These are divided into three parts, *Position data*, *Milking data* and *Cow information data*. The different categories of data are described in section 3.2.1.
When type of data has been selected, the user is presented with several options that can be queried for in the selected part of the database. These choices differ depending on what kind of data that is to be queried for.
Querying is to be done by inserting requested information in the fields on the web application or through the API. For the web application the fields, filled by the user, are structured into a function call, connecting the front-end and the back-end by the "Fetch data" button which initiates a POST-request with all the user input. The back-end in turn, extracts the requested information from the database and writes the information into one or more files. The files and some meta information about file content is returned to the web application page. This workflow is displayed in Figure 5. The user interface for querying is displayed in Figure 6 for Swedish farm data and in Figure 7 for Dutch data. When finished, a file is created in the file folder labelled */result/*.
The user can navigate to this folder to use the new, generated file that contains the information specified for.
The different user inputs for querying data are described below. For more details on the query function of the back-end, see section 3.2.3.

3.3.4 Front-end: Overview

To give the users a basic overview of the database in manner of storage size and records, one can use the *Overview* tab in the left menu. Here the user will find three tables with information: **Database size**, which contains information about the amount of records stored for each file type and its corresponding sizes in MBs. **Cow Information data** lists the stored and available files for cow information data. **Position data** lists the stored and available position data files from the database.

For this information to be listed, the front-end calls the *overview.py* script that calls the backend function to check the statistics of records. This information is the passed to the *overview.html* template to create the tables. See section 3.2.4 for more technical detail regarding the backend functions.

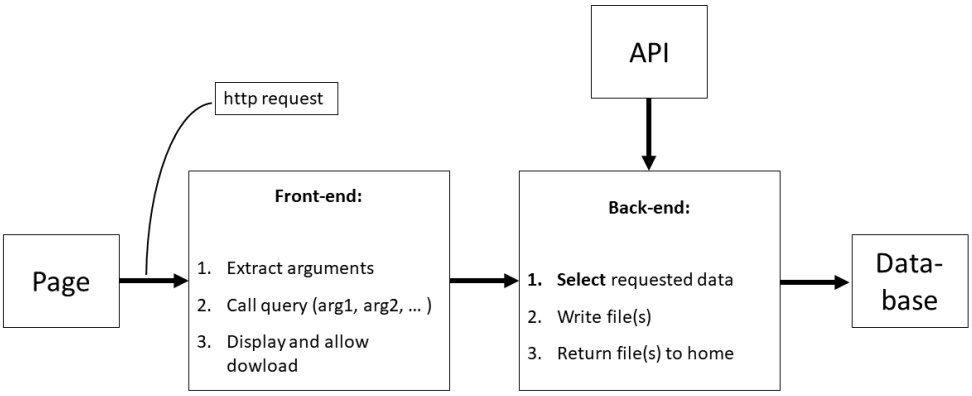

Figure 5: Workflow for querying from the web application.

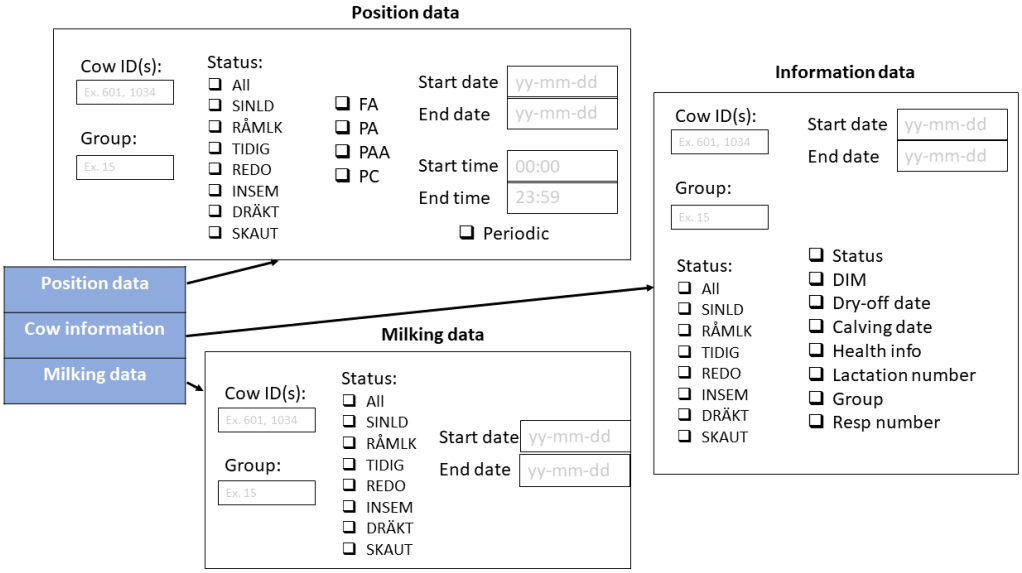

Figure 6: UI workflow for querying Swedish data.

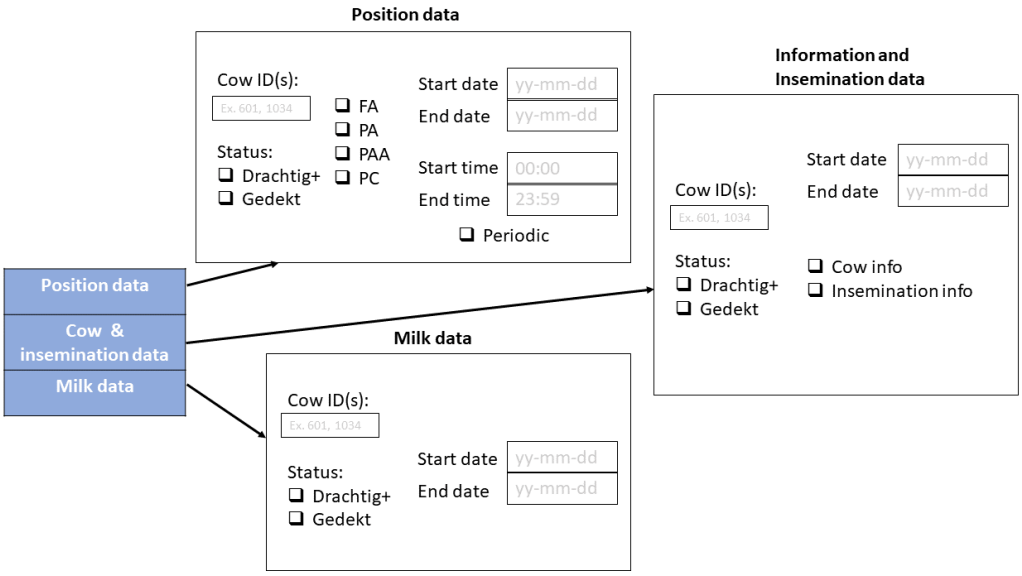

Figure 7: UI workflow for querying Dutch data.

3.3.5 Position Data Query

Position data is queried by choosing what cow(s) to query, what filetype(s) to query and what time interval to query. Cows are chosen by either entering comma-separated cow IDs, comma-separated group numbers and/or a requested status of said cow(s). The file type is selected by checking one or more boxes corresponding to FA, PA, PAA and PC data. Choosing time interval is done firstly selecting start and end dates. Then the user chooses the start and end time (if left empty, the default time is 00:00:00-23:59:59). The user must decide if the time is to be considered periodic by checking the box marked *Periodic*. Periodic time denotes that the chosen time interval will be used every day. E.g. if the user wants to query position data between 8:00 and 16:00 each day, then the periodic box must be checked.

When fetching data, the following function call is made from the front-end to the back-end: **positionQuery(arg1, ... ,arg9)**, where

- arg1 = $cow\_id : [int]$
- arg2 = $group\_no : [int]$
- arg3 = $status : [string]$
- arg4 = $position\_type : [string]$
- arg5 = $start\_date : string(yy - mm - dd)$
- arg6 = $end\_date : string(yy - mm - dd)$
- arg7 = $start\_time : string(hour : min : sec)$
- arg8 = $end\_time : string(hour : min : sec)$
- arg9 = $periodic : bool$

Issuing a position data request returns one or more csv files, depending on the number of selected position file types (FA, PA, PAA and PC), for downloading or locally copying.

3.3.6 Milking Data Query

When fetching data, the following function call is made from the front-end to the back-end: **milkingQuery(arg1, ... ,arg5)**, where

- arg1 = $cow\_id : [int]$
- arg2 = $group\_no : [int]$
- arg3 = $status : [string]$
- arg4 = $start\_date : string(yy - mm - dd)$
- arg5 = $end\_date : string(yy - mm - dd)$

Issuing a milking data request returns a csv file containing records of all requested milkings for downloading or locally copying.

3.3.7 Info Data Query

When fetching data, the following function call is made from the front-end to the back-end: **infoQuery(arg1, ... ,arg6)**, where

- arg1 = $cow\_id : [int]$
- arg2 = $group\_no : [int]$
- arg3 = $status : [string]$
- arg4 = $start\_date : string(yy - mm - dd)$
- arg5 = $end\_date : string(yy - mm - dd)$
- arg6 = $information : [string]$

Issuing a milking data request returns a csv file containing all requested health records for downloading or locally copying.

3.4 APIs

The APIs provide python functions necessary to allow users access database conveniently via other programs. Overall, this includes following functionalities:

- File scan: both blocking IO and non blocking IO functions to scan and insert files to database are provided;
- Query data: both blocking and non blocking functions to query data from database are provided;
- Overview: two overview functions are provided, to check meta data of the database, one is to check status of the database and another is to provide log of inserted files in database.

4 Results and Discussion

4.1 Fulfilled goals

Swedish farm data is successfully uploaded and stored in a well thought through design. The position data is stored in four tables, separating the position data file types FA, PA, PAA and PC. The position data is reached through its primary; key tag id and timestamp.

A mapping table is used to link a cow id to a tag id for position data querying. The mapping table is updated when a cow is dried off or given a new tag, given in the "KO info" files.

Milking place data is stored, representing each milking with a record. Each record is matched to the corresponding day of milking, containing milking station and time of milking.

The user is able to query data from a web page, returning a text file in the local directory. The same kind of queries can be made using an API.

The database system performs regular checks to provide meta data and statistics. This information is displayed on the web page.

4.2 Unfulfilled goals

The dutch data is not yet stored within the database due to time limitations. A structural design for how the data is to be stored has been created. The web page is equipped with a tab for querying dutch data, see Figure 7. To include the dutch data, the front-end must be expanded with functions to formulate input arguments to call the back-end functions. The back-end function set has to be expanded with insertion and extracting functions. The system is built modular-wise, making it scalable and relatively easy to add these functions.

Milk production data, uploaded in "Avkastn 14 dag" files has not yet been included in the database system. This is due to the difficulties in matching a produced volume record to a date. The dates displayed some overlapping patterns that were not consistent throughout the data set. Front- and back-end functions for inserting and extracting this data are in place, but on hold until the data can be sorted out.

Querying currently requires access to the local machine. This might have to be expanded to allow download through the web page to fulfill user requirements.

The database system does not perform regular checks to identify broken tags.

4.3 Testing

4.3.1 User testing

The web application was tested by allowing a user, not familiar with the code, navigate the application, trying to fulfill a set of tasks. The tasks were:

- Insert one position data file and an information data file into the database.
- Query position data and information data.
- Navigate the web page freely.

The testing session was recorded, with permission from the user, and produced the following summarized results.

– The home page contained detailed instructions of how to use the web page. The text was hard to understand and was considered too much. The user suggested to move the detailed instructions to where they were to be used and to rewrite them in a more clear way.

– The user had had some difficulties in understanding how to use the scanning functions to insert files into the database. The instructions needed improvement. The user thought the process of copying files to a local directory was tedious and would prefer being able to upload files through the web page.

– In the scanning and querying tab, the tester experienced a lack of user feedback. There were no interactive text telling the tester if an invalid entry was inserted. Nor was the tester told when a scan or query was finished.

– The user found it hard to see the home button in the top navigational bar of the interface.

The result of the user test lead to the following changes to be implemented:

– The user instructions were restructured and rewritten to facilitate the user experience. The introductory instructions were shortened and specific task instructions were rewritten and moved to be available whilst performing the specific tasks.

– When a query is issued, a message is displayed on the interface informing the user if the query is successful. If not, the system displays a message with the correct error. If the query succeeds, such a message is displayed together with the parameters that are used in the query as a confirmation.

– The home button was moved to the left navigational menu for easier access for the user.

– The other problems were documented and added to our future plan, since the system needs to be fully implemented after the course project is done.

4.3.2 Performance testing

All performance tests were conducted using Ubuntu 18.04.5 LTS on *Intel Core i5-4460 CPU @ 3.20GHz × 4*.

The results from the benchmark measurements, used as decision basis in Section 3.1.2 is shown in Figure 8. The second schema, using four tables, separating the different file types, outperforms the first schema where each tag is given a separate table.

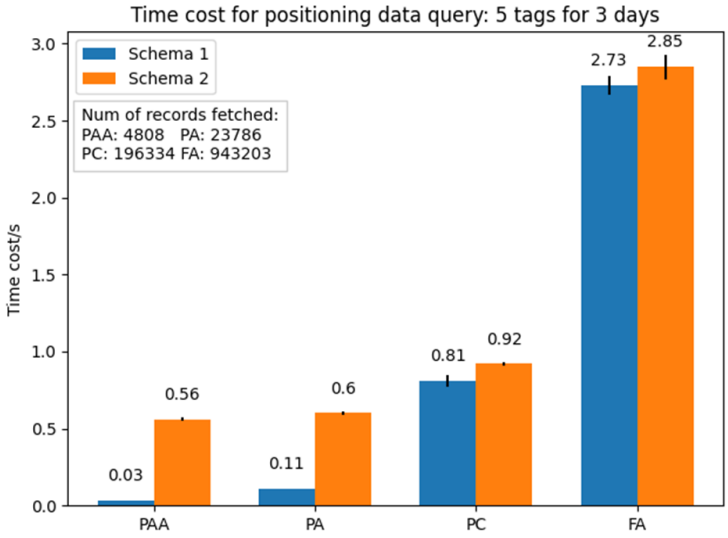

Figure 8: Benchmark performance measurement of extracting position data from two kinds of position data schemas (Section 3.1.2).

To test whether the back-end functions performed acceptably efficient, performance tests were conducted. Acceptable was defined as uploading a day's worth of position data (25M records) in less than 30 minutes and querying a day's worth of position data from one cow (86k records) in under 5 seconds. We inserted 15.8, 31.6 and 47.4 million position data records, resulting in Figure 9. The results are considered acceptable. We queried 68, 138 and 271 thousand position data records for continuous records and periodic records (between a set time each day). The results are shown in Figure 10 and displayed acceptable perfomance.

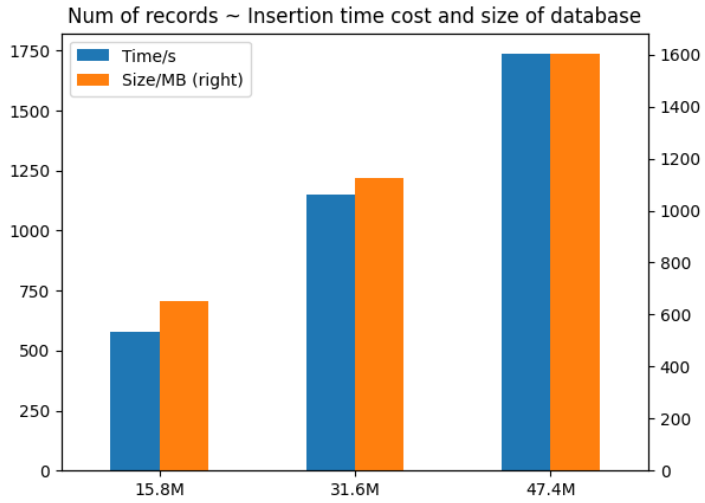

Figure 9: Performance tests, displaying the insertion time [s] (left) and storage [Mb] (right) for three quantities of position data.

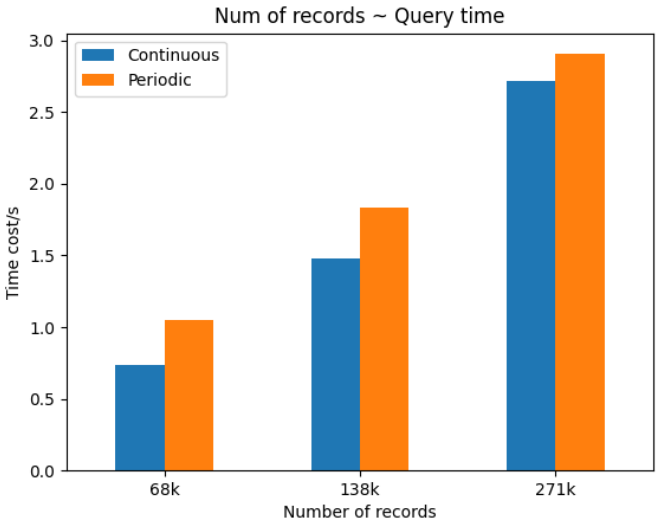

Figure 10: Performance tests, displaying the query time [s] for three quantities of position data.

5 Conclusions

The development has gone according to plan with some drawbacks regarding the milk data and the fact that the dutch farm data is not yet included in the database system. The finished parts of the database system work sufficiently well. The remaining unfinished components are the milk production data and adding the data from the Dutch farm. In conclusion, the major tasks in the project are successfully finalized.

6 Contributions

The workload was distributed evenly. Li and Gustaf were mainly focused on processing the data and back-end functions. Linus was mainly focused on the Front-end, developing the web page and transforming user input to function arguments.

7 Acknowledgments

We would also like to thank the members of the SLU research group associated with the project. Svenja Woudstra and Ida Hansson helped us understand the meaning of the information and facilitated the data processing. Mikhail Churakov, Freddy Fikse and Lars Rönnegård for input and discussions regarding database structure. Per Peetz for making contact to the farmers regarding the deviating milk data.

We would also like to thank Edvin Ekstrand and Gustav Aldevik for assisting the web development using HTML and CSS.

Most of all we would like to thank Keni Ren for supervising and mentoring us through the entire project.

References

[1] Stonebraker, M., 2010. SQL databases v. NoSQL databases. Communications of the ACM, 53(4), pp.10-11.

[2] Köhler, H. and Link, S., 2018. SQL schema design: foundations, normal forms, and normalization. Information Systems, 76, pp.88-113.

[3] Hansson, I. SLU, Uppsala, *ida.hansson@slu.se*.

[4] The SciPy community, 2020, *numpy.vectorize — NumPy v1.19 Manual*, viewed 28th December 2020, *https://numpy.org/doc/stable/reference/generated/numpy.vectorize.html*

[5] Oracle Corporation and/or its affiliates, 2021, *MySQL Connector/Python Developer Guide*, viewed 20th December 2020, *https://dev.mysql.com/doc/connector-python/en/*

[6] Django Software Foundation, 2020, *Django Documentation*, viewed 20th December 2020, *https://docs.djangoproject.com/en/3.1/*