

BasregDWH User Guide: Access data in Basreg

Tomas Klingström

2026-01-17

Contents

Introduction	2
Software prerequisites	2
Optional: Git + GitHub (recommended)	3
Download this guide from GitHub	3
Working with this guide	3
Install packages (only needed once)	4
Load packages (needed every session)	4
Check your R session	4
Check your current working directory	4
Step 1: Connect to BasregDWH	4
Step 2: Preview a view (download only a few rows)	5
Step 3: Filtering rows (sparse views are common)	6
Step 4: Multiple filters (ALL must be true)	7
Step 5: OR filters (either condition can be true)	8
Step 6: Select columns (download only what you need)	9
Step 7: Modifying columns	10
Step 8: Combining information from multiple views	11
Step 9: Select milking records after each calving	13
Step 10: Performing calculations on data	15

Step 11: Plot daily milk yield after calving	17
Step 12: Save results and close the database connection	19



Introduction

This guide helps you access the SLU Basreg database (**basregDWH**) using R and RStudio.

You will learn how to:

- Connect to the server using your SLU credentials
- Browse available tables and views
- Filter and handle large datasets before downloading them

If you are unfamiliar with Rstudio, Posit (the maker of Rstudio) describes the interface here: <https://docs.posit.co/ide/user/ide/guide/code/projects>

If you are reading this documentation with the intention of using it for data analysis. Please make sure you understand how R projects work: <https://docs.posit.co/ide/user/ide/guide/code/projects.html>

Important: Users have **read-only access**.

This means you can view and download data, but you are **not at risk of changing the database**.

Software prerequisites

You need:

1. RStudio

Official getting started guide:

<https://posit.co/resources/videos/getting-started-with-rstudio/>

2. Access to SLU Basreg / BasregDWH

For permissions, contact: **Eva Rundlöf**

3. Either being at a SLU campus or use the SLU VPN

How to use VPN: <https://internt.slu.se/en/support-services/administrative-support/it/support/guides-manuals/vpn-anyconnect/>

If you have a Windows-laptop issued by SLU, ODBC will automatically verify your identity when connecting. Linux and Mac users need to follow a separate instruction.

Optional: Git + GitHub (recommended)

If you want to keep your work organised, you can use Git directly in RStudio:
<https://docs.posit.co/ide/user/ide/guide/tools/version-control.html>

Download this guide from GitHub

Repository:

https://github.com/TKlingstrom/Basreg_introduction

You can download it in two ways:

Option A: Clone with Git in RStudio (recommended)

1. In RStudio: **File → New Project → Version Control → Git**
2. Paste the repo URL
3. Choose a folder and click **Create Project**

Option B: Download as a ZIP

1. Open the repository in your browser
2. Click **Code → Download ZIP**
3. Unzip and open the **.Rproj** file in RStudio

Working with this guide

Preferably you follow the guide directly in R-markdown. To do this you open basreg_userguide.Rmd with Rstudio or open Rstudio and select **File → Open project in new session**.

You can also create your own R-script (In RStudio: **File → New File → R-script**) and copy the code from the HTML file (or PDF).

The difference between R markdown and an R-script is that in markdown you can run and rerun your code in “chunks”. Chunks are grey coloured and got a small green play arrow in the upper right corner. The output from running the code is displayed right below the code. R markdown can also easily be converted into a PDF or html file. I recommend that you follow this guide directly in the R markdown file or, alternatively open the HTML file in your browser and then write the code into an R-script. In the R markdown file you can scroll down until you find this text and then begin the tutorial below. Each code chunk must be run at least once (Clicking the green play button in the upper right corner of the markdown file or by copying it into your R script, selecting it and hitting CTRL + Enter). You can modify the code and rerun it as you wish, the name to the left of the <-will be the name of the object generated by the code and can be accessed in the Environment tab (top right pane).

We use these packages which must be installed and loaded (run the code below):

- **odbc**: database driver connection
- **DBI**: database interface
- **dplyr**: data manipulation
- **dbplyr**: makes dplyr work with databases
- **ggplot2**: is used later for plotting data

Install packages (only needed once)

```
install.packages(c("odbc", "DBI", "dplyr", "dbplyr", "ggplot2"))
```

Load packages (needed every session)

```
library(odbc)
library(DBI)
library(dplyr)
library(dbplyr)
library(ggplot2)
```

Check your R session

Check your current working directory

This is where R reads/writes files (e.g. exports).

```
getwd()
```

To change folder:

```
#If necessary, remove the # below and set your working directory to the Basreg_introduction folder
#setwd("C:/path/to/your/folder")
```

Step 1: Connect to BasregDWH

This chunk creates a **connection object** called `con`.

Think of `con` as the way to tell your Rstudio session to connect to the Basreg server.

You do **not** download the whole database into your computer.

Instead, you send requests to the server to collect data from SQL views.

A view is a way to present information in an SQL server and contain primarily observation-level data (events, measurements). Due to the way Basreg was developed it also contain some less useful columns necessary for other ways to interact with Basreg.

```
# Connecting to the database using your R/Windows user credentials
con <- dbConnect(
  odbc(),
  Driver = "SQL Server",
  Server = "basregdwh.db.slu.se",
  Database = "basregDWH"
```

```
)
# List objects available in the apiSci schema
odbcListObjects(con, catalog = "basregDWH", schema = "apiSci")
```

```
##                               name type
## 1      bridgeTrialsCattleFeedIntakeView view
## 2      bridgeTrialsCattleHealthView view
## 3      bridgeTrialsCattleHerdView view
## 4      bridgeTrialsCattleMilkingParlourView view
## 5      bridgeTrialsCattleMilkingView view
## 6      bridgeTrialsCattleReproductionView view
## 7      bridgeTrialsCattleSlaughterView view
## 8      bridgeTrialsCattleWeightsView view
## 9      bridgeTrialsPigBackFatsView view
## 10     bridgeTrialsPigFarrowingsView view
## 11     bridgeTrialsPigInseminationsView view
## 12     bridgeTrialsPigSlaughterView view
## 13     bridgeTrialsPigTransfersView view
## 14     bridgeTrialsPigWeightsView view
## 15     dimTrialsAndAnimalsView view
## 16     factCattleFeedIntakeView view
## 17     factCattleHealthView view
## 18     factCattleHerdView view
## 19     factCattleMilkingParlourView view
## 20     factCattleMilkingView view
## 21     factCattleReproductionView view
## 22     factCattleSlaughterView view
## 23     factCattleWeightsView view
## 24     factCattleView view
## 25     factPigBackFatsView view
## 26     factPigFarrowingsView view
## 27     factPigInseminationsView view
## 28     factPigSlaughterView view
## 29     factPigTransfersView view
## 30     factPigWeightsView view
## 31     factPigView view
```

Step 2: Preview a view (download only a few rows)

The function `tbl()` creates a reference to a database table/view specified in the parenthesis.
 Thanks to `dbplyr` you can filter or manipulate the data before downloading it. In this case:

- `head(10)` limits to a small preview
- `collect()` downloads those rows into R as a data frame

The %>% is how you define the different steps dplyr and dbplyr will take in a pipe when filtering or manipulating data. Writing DF.Cow after the pipe then shows the output to you. In the environment tab (upper right pane in Rstudio) you also see all objects you create. Clicking the blue circle before them shows the datatype and clicking the table far to the right will show a table with the contents of the dataframe. The Connections tab to the right of Environment also show you all the database schemas you can access, if you have access you can click the blue circle and see what is inside of it. The views we will work with are located in **basregDWH** → **apiSci** and starts with “fact”.

```
DF.Cow <- con %>%
  tbl(in_catalog("basregDWH", "apiSci", "factCattleView")) %>%
  head(10) %>%
  collect()

DF.Cow

## # A tibble: 10 x 45
##   Farm    SE_Number DelProID CeresCattleNumber CeresTransponderID Gender Species
##   <chr>    <chr>      <int>           <int>            <int64> <chr>  <chr>
## 1 Lövsta 1          1             NA              NA Female  Cattle
## 2 Lövsta 2          2             NA              NA Female  Cattle
## 3 Lövsta SE-08827~  31            NA              NA Female  Cattle
## 4 Lövsta SE-08827~  80            NA              NA Female  Cattle
## 5 Lövsta SE-08827~  228            NA              NA Female  Cattle
## 6 Lövsta SE-08827~  235            NA              NA Female  Cattle
## 7 Lövsta SE-08827~  270            NA              NA Female  Cattle
## 8 Lövsta SE-08827~  279            NA              NA Female  Cattle
## 9 Lövsta SE-08827~  282            NA              NA Female  Cattle
## 10 Lövsta SE-08827~ 286            NA              NA Female Cattle
## # i 38 more variables: BreedName <chr>, BirthDate <chr>, BirthWeight <dbl>,
## #   Mother <chr>, Father <chr>, CullDecisionDate <chr>, CullReason1 <chr>,
## #   CullReason2 <chr>, ExitDate <chr>, SerialNo <int>, BornHerdNo <int>,
## #   DeathDate <chr>, ArrivalDate <chr>, Dissected <lgl>, DissectedDate <chr>,
## #   Carrier <chr>, BreedPart1 <chr>, `BreedPart1%` <int>, BreedPart2 <chr>,
## #   `BreedPart2%` <int>, BreedPart3 <chr>, `BreedPart3%` <int>,
## #   BreedPart4 <chr>, `BreedPart4%` <int>, TwinBorn <chr>, Supplier <chr>, ...
```

Try switching the object name in the code above, for example:

- "factCattleView" to
- "factCattleReproductionView"

Step 3: Filtering rows (sparse views are common)

Many Basreg views are **sparse**.

That means most rows contain only a few filled columns and many NA.

Filtering helps you select only meaningful rows.

Example:

- `!is.na(Calving)` means: keep only rows where Calving exists
- In this dataset, `StartDate` contains the date of the event
- Switching to `DryOff` instead gives dry-off events

```
DF.Reproduction <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
  filter(!is.na(Calving)) %>%
  collect()

DF.Reproduction

## # A tibble: 2,521 x 27
##   Farm   Place Equipment SE_Number     DelProId BiocontrolId CeresCattleNumber
##   <chr>  <chr>    <chr>      <chr>       <int>      <int>           <int>
## 1 Lövsta Lövsta Unknown SE-088270-02~     235        NA            NA
## 2 Lövsta Lövsta Unknown SE-088270-02~     270        NA            NA
## 3 Lövsta Lövsta Unknown SE-088270-03~     317        NA            NA
## 4 Lövsta Lövsta Unknown SE-088270-03~     328        NA            NA
## 5 Lövsta Lövsta Unknown SE-088270-03~     344        NA            NA
## 6 Lövsta Lövsta Unknown SE-088270-03~     355        NA            NA
## 7 Lövsta Lövsta Unknown SE-088270-03~     356        NA            NA
## 8 Lövsta Lövsta Unknown SE-088270-03~     367        NA            NA
## 9 Lövsta Lövsta Unknown SE-088270-03~     369        NA            NA
## 10 Lövsta Lövsta Unknown SE-088270-02~    270        NA            NA
## # i 2,511 more rows
## # i 20 more variables: dimCattle_Id <int>, StartDate <chr>, StartTime <chr>,
## #   InseminationMethod <chr>, SemenId <chr>, PregnancyCheckResult <chr>,
## #   Calving <int>, CalvingEase <chr>, ExpectedCalvingDate <chr>,
## #   ExpectedDryOffDate <chr>, DryOff <int>, BiometricType <chr>,
## #   SmoothedLevel <dbl>, RawLevel <dbl>, Heat <int>, InseminationNumber <int>,
## #   LactationNumber <int>, NumberOfDaysInLactation <int>, ...
```

Step 4: Multiple filters (ALL must be true)

You can combine filters.

In `filter()`, multiple conditions mean:

condition 1 **AND** condition 2 **AND** condition 3

Here we select:

- rows with Calving events (the value in the column `Calving` is not NA)
- only events between 2021-01-01 and 2021-01-31 (the value in column `StartDate` is equal or after 2021-01-01 but equal or before 2021-01-31)

```
DF.Reproduction <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
```

```

filter(
  !is.na(Calving),
  StartDate >= as.Date("2021-01-01"),
  StartDate <= as.Date("2021-01-31")
) %>%
collect()

```

DF.Reproduction

```

## # A tibble: 29 x 27
##   Farm   Place Equipment SE_Number     DelProId BiocontrolId CeresCattleNumber
##   <chr>  <chr>    <chr>      <chr>       <int>        <int>            <int>
## 1 Lövsta Lövsta Unknown SE-088270-20~    2008         NA             NA
## 2 Lövsta Lövsta Unknown SE-088270-20~    2010         NA             NA
## 3 Lövsta Lövsta Unknown SE-088270-20~    2013         NA             NA
## 4 Lövsta Lövsta Unknown SE-088270-20~    2014         NA             NA
## 5 Lövsta Lövsta Unknown SE-088270-21~    2136         NA             NA
## 6 Lövsta Lövsta Unknown SE-088270-21~    2155         NA             NA
## 7 Lövsta Lövsta Unknown SE-088270-21~    2170         NA             NA
## 8 Lövsta Lövsta Unknown SE-088270-21~    2171         NA             NA
## 9 Lövsta Lövsta Unknown SE-088270-21~    2172         NA             NA
## 10 Lövsta Lövsta Unknown SE-088270-21~   2173         NA             NA
## # i 19 more rows
## # i 20 more variables: dimCattle_Id <int>, StartDate <chr>, StartTime <chr>,
## #   InseminationMethod <chr>, SemenId <chr>, PregnancyCheckResult <chr>,
## #   Calving <int>, CalvingEase <chr>, ExpectedCalvingDate <chr>,
## #   ExpectedDryOffDate <chr>, DryOff <int>, BiometricType <chr>,
## #   SmoothedLevel <dbl>, RawLevel <dbl>, Heat <int>, InseminationNumber <int>,
## #   LactationNumber <int>, NumberOfDaysInLactation <int>, ...

```

Step 5: OR filters (either condition can be true)

Sometimes you want either:

- Calving events **OR**
- DryOff events

In R:

- | means **OR**
- & means **AND** (commas also work as you saw above)

Here we select:

- rows where Calving exists **OR** DryOff exists
- but still only within January 2021

```

DF.Reproduction <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
  filter(
    (!is.na(Calving) | !is.na(DryOff)),
    StartDate >= as.Date("2021-01-01"),
    StartDate <= as.Date("2021-01-31")
  ) %>%
  collect()

DF.Reproduction

## # A tibble: 56 x 27
##   Farm   Place Equipment SE_Number     DelProId BiocontrolId CeresCattleNumber
##   <chr>  <chr>   <chr>      <chr>       <int>      <int>           <int>
## 1 Lövsta Lövsta Unknown SE-088270-20~    2005        NA            NA
## 2 Lövsta Lövsta Unknown SE-088270-20~    2012        NA            NA
## 3 Lövsta Lövsta Unknown SE-088270-20~    2027        NA            NA
## 4 Lövsta Lövsta Unknown SE-088270-20~    2029        NA            NA
## 5 Lövsta Lövsta Unknown SE-088270-20~    2030        NA            NA
## 6 Lövsta Lövsta Unknown SE-088270-20~    2032        NA            NA
## 7 Lövsta Lövsta Unknown SE-088270-20~    2034        NA            NA
## 8 Lövsta Lövsta Unknown SE-088270-20~    2040        NA            NA
## 9 Lövsta Lövsta Unknown SE-088270-20~    2042        NA            NA
## 10 Lövsta Lövsta Unknown SE-088270-20~   2045        NA            NA
## # i 46 more rows
## # i 20 more variables: dimCattle_Id <int>, StartDate <chr>, StartTime <chr>,
## #   InseminationMethod <chr>, SemenId <chr>, PregnancyCheckResult <chr>,
## #   Calving <int>, CalvingEase <chr>, ExpectedCalvingDate <chr>,
## #   ExpectedDryOffDate <chr>, DryOff <int>, BiometricType <chr>,
## #   SmoothedLevel <dbl>, RawLevel <dbl>, Heat <int>, InseminationNumber <int>,
## #   LactationNumber <int>, NumberOfDaysInLactation <int>, ...

```

Step 6: Select columns (download only what you need)

In Basreg views there are often many columns, The function select() keeps only the columns you specify. Everything else is removed from the download.

Here we reuse the same query as in Step 5 (Calving or DryOff events in January 2021), but only collect the most important columns.

```

DF.Reproduction <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
  filter(
    (!is.na(Calving) | !is.na(DryOff)),
    StartDate >= as.Date("2021-01-01"),
    StartDate <= as.Date("2021-01-31")
  ) %>%

```

```
select(Farm, SE_Number, StartDate, Calving, CalvingEase, DryOff) %>%
  collect()
```

DF.Reproduction

```
## # A tibble: 56 x 6
##   Farm    SE_Number     StartDate  Calving CalvingEase DryOff
##   <chr>   <chr>        <chr>      <int>  <chr>       <int>
## 1 Lövsta SE-088270-2005 2021-01-04     NA Unknown       1
## 2 Lövsta SE-088270-2012 2021-01-08     NA Unknown       1
## 3 Lövsta SE-088270-2027 2021-01-22     NA Unknown       1
## 4 Lövsta SE-088270-2029 2021-01-04     NA Unknown       1
## 5 Lövsta SE-088270-2030 2021-01-22     NA Unknown       1
## 6 Lövsta SE-088270-2032 2021-01-15     NA Unknown       1
## 7 Lövsta SE-088270-2034 2021-01-08     NA Unknown       1
## 8 Lövsta SE-088270-2040 2021-01-22     NA Unknown       1
## 9 Lövsta SE-088270-2042 2021-01-15     NA Unknown       1
## 10 Lövsta SE-088270-2045 2021-01-04    NA Unknown       1
## # i 46 more rows
```

Step 7: Modifying columns

We will now modify the view before downloading it from the server.

In the chunk below we remove the columns `Calving`, `DryOff` and `StartDate` and with the `mutate` function instead create:

- `CalvingDate` which contains the date of the calving (from `StartDate`)
- `DryOffDate` which contains the date of the dry-off (from `StartDate`)

This makes the dataset easier to interpret after downloading it.

```
DF.Reproduction <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
  filter(
    (!is.na(Calving) | !is.na(DryOff)),
    StartDate >= as.Date("2021-01-01"),
    StartDate <= as.Date("2021-01-31")
  ) %>%
  mutate(
    # mutate() creates new columns (or modifies existing ones)
    # Read more here: https://dplyr.tidyverse.org/reference/mutate.html
    CalvingDate = if_else(!is.na(Calving), StartDate, as.Date(NA)),
    DryOffDate = if_else(!is.na(DryOff), StartDate, as.Date(NA))
  ) %>%
  select(Farm, SE_Number, CalvingDate, DryOffDate, CalvingEase) %>%
  collect()
```

DF.Reproduction

```
## # A tibble: 56 x 5
##   Farm    SE_Number     CalvingDate DryOffDate CalvingEase
##   <chr>   <chr>        <chr>       <chr>       <chr>
## 1 Lövsta SE-088270-2005 <NA>        2021-01-04 Unknown
## 2 Lövsta SE-088270-2012 <NA>        2021-01-08 Unknown
## 3 Lövsta SE-088270-2027 <NA>        2021-01-22 Unknown
## 4 Lövsta SE-088270-2029 <NA>        2021-01-04 Unknown
## 5 Lövsta SE-088270-2030 <NA>        2021-01-22 Unknown
## 6 Lövsta SE-088270-2032 <NA>        2021-01-15 Unknown
## 7 Lövsta SE-088270-2034 <NA>        2021-01-08 Unknown
## 8 Lövsta SE-088270-2040 <NA>        2021-01-22 Unknown
## 9 Lövsta SE-088270-2042 <NA>        2021-01-15 Unknown
## 10 Lövsta SE-088270-2045 <NA>       2021-01-04 Unknown
## # i 46 more rows
```

Step 8: Combining information from multiple views

The reproduction view tells us when a calving happened, but it does not directly show which offspring cow was born from that event.

To connect the calving event to the offspring identity, we will combine two views:

- `factCattleReproductionView` contains calving information and calving dates
- `factCattleView` contains the offspring cow identity and the mother's identity

In `factCattleView`, the mother's SE-number is stored in the column `Mother`.

We will:

1. Extract `Farm`, `SE_Number`, `CalvingDate`, and `CalvingEase` from `factCattleReproductionView`
2. Extract `SE_Number` (offspring), `Mother`, and `BirthDate` from `factCattleView`
3. Match calving events by joining:
 - `SE_Number` (mother) = `Mother`
 - `CalvingDate` = `BirthDate`
4. Keep the calving information and rename the offspring `SE_Number` into `Offspring`

```
# IMPORTANT:
# In this step we do NOT download (collect) the full views into R.
# Instead, dbplyr keeps the objects as "lazy" SQL queries that remain on the Basreg server.
# This means filter(), transmute() and inner_join() are translated into SQL and run server-side.
# Only when we call collect() at the end will the result be downloaded into R.
```

```

# Running computations server-side is appropriate when it reduces the amount of data you need to
# should however be avoided as it may impact other users.

#
# Read more here:
# - dbplyr basics (lazy queries): https://dbplyr.tidyverse.org/articles/dbplyr.html
# - dbplyr translation to SQL: https://dbplyr.tidyverse.org/articles/translation.html

# 1) Prepare the calving events table (still server-side, NOT collected)
DF.ReproductionEvents <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
  filter(
    !is.na(Calving),
    StartDate >= as.Date("2021-01-01"),
    StartDate <= as.Date("2021-01-31")
  ) %>%
  transmute(
    Farm = Farm,
    SE_Number = SE_Number,           # Mother identity in reproduction view
    CalvingDate = StartDate,         # Calving date is stored in StartDate
    CalvingEase = CalvingEase
  )

# 2) Prepare the cattle view with offspring information (still server-side, NOT collected)
DF.Cattle <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleView")) %>%
  transmute(
    Offspring = SE_Number,          # Offspring identity in cattle view
    Mother = Mother,                # Mother identity in cattle view
    BirthDate = BirthDate
  )

# 3) Join the two server-side objects in SQL Server and sort from first born to last born
DF.CalvingWithOffspring <- DF.ReproductionEvents %>%
  inner_join(
    DF.Cattle,
    by = c("SE_Number" = "Mother", "CalvingDate" = "BirthDate")
  ) %>%
  arrange(CalvingDate) %>% # arrange() sorts the output by date (oldest first)
  select(Farm, SE_Number, Offspring, CalvingDate, CalvingEase) %>%
  collect() # collect() runs the SQL query and downloads ONLY the final joined result into R

# 4) List any calving events that did not find an offspring match
DF.UnmatchedCalvings <- DF.ReproductionEvents %>%
  anti_join(
    DF.Cattle,
    by = c("SE_Number" = "Mother", "CalvingDate" = "BirthDate")

```

```
) %>%
arrange(SE_Number, CalvingDate) %>%
collect()
```

DF.CalvingWithOffspring *#This table will contain all cases where we now have a Mother and a SE_N*

```
## # A tibble: 26 x 5
##   Farm    SE_Number Offspring   CalvingDate CalvingEase
##   <chr>   <chr>     <chr>       <chr>      <chr>
## 1 Lövsta SE-088270-2171 SE-088270-8422 2021-01-01 11 Lätt, utan hjälp
## 2 Lövsta SE-088270-0551 SE-088270-8423 2021-01-01 11 Lätt, utan hjälp
## 3 Lövsta SE-088270-0592 SE-088270-8425 2021-01-02 12 Lätt, med hjälp
## 4 Lövsta SE-088270-0950 SE-088270-8424 2021-01-02 12 Lätt, med hjälp
## 5 Lövsta SE-088270-0790 SE-088270-8426 2021-01-04 11 Lätt, utan hjälp
## 6 Lövsta SE-088270-2177 SE-088270-2526 2021-01-04 11 Lätt, utan hjälp
## 7 Lövsta SE-088270-2013 SE-088270-2527 2021-01-05 11 Lätt, utan hjälp
## 8 Lövsta SE-088270-2010 SE-088270-8429 2021-01-07 12 Lätt, med hjälp
## 9 Lövsta SE-088270-2172 SE-088270-8428 2021-01-07 12 Lätt, med hjälp
## 10 Lövsta SE-088270-2136 SE-088270-8427 2021-01-07 12 Lätt, med hjälp
## # i 16 more rows
```

DF.UnmatchedCalvings *#This table will contain all cases where we could not find a Cow born on t*

```
## # A tibble: 3 x 4
##   Farm    SE_Number   CalvingDate CalvingEase
##   <chr>   <chr>       <chr>      <chr>
## 1 Lövsta SE-088270-0824 2021-01-22 12 Lätt, med hjälp
## 2 Lövsta SE-088270-0933 2021-01-05 13 Svår, utan veterinärhjälp
## 3 Lövsta SE-088270-2187 2021-01-06 11 Lätt, utan hjälp
```

Step 9: Select milking records after each calving

We will now use the calving table we created in Step 8 to find milking records after each calving.

For each mother (`SE_Number`) and each `CalvingDate` in the calving table, we select rows from `factCattleMilkingView` where the milking date (`StartDate`) is within **365 days after the calving date**.

To do this efficiently, we keep the work **server-side** and join the views in SQL Server. Only the final result is downloaded when we use `collect()`.

```
# IMPORTANT:
# We recreate the calving table as a server-side query (not collected),
# then join it to the milking view on the SQL server.
# This avoids looping in R and avoids downloading large views.
#
# Read more here:
```

```

# - dbplyr joins: https://dbplyr.tidyverse.org/articles/two-table.html
# - dbplyr SQL translation: https://dbplyr.tidyverse.org/articles/translation.html

# 1) Calving table (server-side, NOT collected)
DF.CalvingWithOffspring_db <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
  filter(
    !is.na(Calving),
    StartDate >= as.Date("2021-01-01"),
    StartDate <= as.Date("2021-01-31")
  ) %>%
  transmute(
    Farm = Farm,
    SE_Number = SE_Number,      # Mother
    CalvingDate = StartDate,
    CalvingEase = CalvingEase
  ) %>%
  inner_join(
    con %>% tbl(in_catalog("basregDWH", "apiSci", "factCattleView")) %>%
    transmute(
      Offspring = SE_Number,
      Mother = Mother,
      BirthDate = BirthDate
    ),
    by = c("SE_Number" = "Mother", "CalvingDate" = "BirthDate")
  ) %>%
  select(Farm, SE_Number, Offspring, CalvingDate, CalvingEase)

# 2) Milking view (server-side, NOT collected)
DF.Milking <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleMilkingView"))

# 3) Join calvings to milking records and keep only milking within 365 days after each calving
DF.MilkingAfterCalving <- DF.CalvingWithOffspring_db %>%
  inner_join(DF.Milking, by = "SE_Number") %>% # match milking records to the mother
  filter(
    StartDate >= CalvingDate,
    StartDate < sql("DATEADD(day, 365, CalvingDate)") #This is where the 365 day limit is kept
  ) %>%
  arrange(SE_Number, CalvingDate, StartDate) %>% # sort from earliest to latest
  collect() # download only the filtered result to R

DF.MilkingAfterCalving

## # A tibble: 15,571 x 45
##   Farm.x SE_Number     Offspring CalvingDate CalvingEase Farm.y Place Equipment
##   <chr>  <chr>        <chr>      <chr>       <chr>      <chr>  <chr>
## 1 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~

```

```

## 2 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 3 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 4 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 5 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 6 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 7 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 8 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 9 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## 10 Lövsta SE-088270-05~ SE-08827~ 2021-01-15 12 Lätt, m~ Lövsta Lövs~ Milking ~
## # i 15,561 more rows
## # i 37 more variables: DelProId <int>, BiocontrolId <int>,
## #   CeresCattleNumber <int>, dimCattle_Id <int>, StartDate <chr>,
## #   StartTime <chr>, Incompletes <chr>, KickOffs <chr>, NotMilked <chr>,
## #   MilkingDuration_sec <int>, MilkFlowDuration_sec <int>,
## #   MilkingInterval_sec <int>, SessionNumber <int>, TotalYieldLF <dbl>,
## #   TotalYieldRF <dbl>, TotalYieldLR <dbl>, TotalYieldRR <dbl>, ...

```

Step 10: Performing calculations on data

We will now summarise the milking records we selected in Step 9.

We group the dataset by `CalvingEase` and calculate:

- number of cows in each group
- the number of milking days per cow (based on `StartDate`)
- average daily milk yield (based on `TotalYield`)

Because cows can be milked several times per day, the table `factCattleMilkingView` can contain multiple rows per cow and day. In that case `TotalYield` is typically the yield per milking, not the total daily yield. To get a correct daily yield we first sum all milkings within the same day for each cow, and then calculate the average daily yield.

To better describe variation, we also report the median and quartiles (25% and 75%).

```

# This step uses group_by() and summarise() to calculate statistics.
# Read more here:
# - group_by(): https://dplyr.tidyverse.org/reference/group\_by.html
# - summarise(): https://dplyr.tidyverse.org/reference/summarise.html
#
# IMPORTANT:
# In factCattleMilkingView there can be multiple rows per day (multiple milkings).
# TotalYield is then usually the yield per milking, not per day.
# To calculate DAILY yield, we first sum TotalYield within each day for each cow.

# 1) Calculate daily total yield per cow (sums multiple milkings on the same date)
DF.DailyMilkPerCow <- DF.MilkingAfterCalving %>%
  group_by(CalvingEase, SE_Number, StartDate) %>%
  summarise(
    DailyYield = sum(TotalYield),
    MedianYield = median(TotalYield),
    Q1Yield = quantile(TotalYield, 0.25),
    Q3Yield = quantile(TotalYield, 0.75)
  )

```

```

DailyTotalYield = sum(TotalYield, na.rm = TRUE), # sum yield across milkings that day
MilkingPerDay = n(), # number of milkings that day
.groups = "drop"
)

DF.DailyMilkPerCow

```

```

## # A tibble: 6,349 x 5
##   CalvingEase      SE_Number    StartDate DailyTotalYield MilkingPerDay
##   <chr>            <chr>        <chr>          <dbl>             <int>
## 1 11 Lätt, utan hjälp SE-088270-0551 2021-01-09     6.03             1
## 2 11 Lätt, utan hjälp SE-088270-0551 2021-01-10    16.4              2
## 3 11 Lätt, utan hjälp SE-088270-0551 2021-01-11    21.0              2
## 4 11 Lätt, utan hjälp SE-088270-0551 2021-01-12    21.9              2
## 5 11 Lätt, utan hjälp SE-088270-0551 2021-01-13    23.6              2
## 6 11 Lätt, utan hjälp SE-088270-0551 2021-01-14    30.7              3
## 7 11 Lätt, utan hjälp SE-088270-0551 2021-01-15    24.1              2
## 8 11 Lätt, utan hjälp SE-088270-0551 2021-01-16    29.0              2
## 9 11 Lätt, utan hjälp SE-088270-0551 2021-01-17    30.4              2
## 10 11 Lätt, utan hjälp SE-088270-0551 2021-01-18   36.6              2
## # i 6,339 more rows

```

2) Summarise per cow (within each CalvingEase group)

```

DF.MilkingPerCow <- DF.DailyMilkPerCow %>%
  group_by(CalvingEase, SE_Number) %>%
  summarise(
    MilkingDays = n_distinct(StartDate), # number of days with milking records
    AvgDailyYieldCow = mean(DailyTotalYield, na.rm = TRUE), # average DAILY yield per cow
    .groups = "drop"
  )

```

```
DF.MilkingPerCow
```

```

## # A tibble: 26 x 4
##   CalvingEase      SE_Number    MilkingDays AvgDailyYieldCow
##   <chr>            <chr>          <int>             <dbl>
## 1 11 Lätt, utan hjälp SE-088270-0551     291             41.9
## 2 11 Lätt, utan hjälp SE-088270-0571      61              29.3
## 3 11 Lätt, utan hjälp SE-088270-0790     289             44.0
## 4 11 Lätt, utan hjälp SE-088270-0806     325             41.2
## 5 11 Lätt, utan hjälp SE-088270-0808     145             48.4
## 6 11 Lätt, utan hjälp SE-088270-0823     335             38.4
## 7 11 Lätt, utan hjälp SE-088270-0930     126             53.3
## 8 11 Lätt, utan hjälp SE-088270-0946     303             40.0
## 9 11 Lätt, utan hjälp SE-088270-2008     314             42.2
## 10 11 Lätt, utan hjälp SE-088270-2013    104             38.6
## # i 16 more rows

```

```

# 3) Summarise per CalvingEase group (mean, median, quartiles)
DF.SummaryByCalvingEase <- DF.MilkingPerCow %>%
  group_by(CalvingEase) %>%
  summarise(
    Cows = n_distinct(SE_Number),
    AvgMilkingDaysPerCow = mean(MilkingDays, na.rm = TRUE),
    MedianMilkingDaysPerCow = median(MilkingDays, na.rm = TRUE),
    MilkingDays_Q25 = quantile(MilkingDays, 0.25, na.rm = TRUE),
    MilkingDays_Q75 = quantile(MilkingDays, 0.75, na.rm = TRUE),
    AvgDailyYield = mean(AvgDailyYieldCow, na.rm = TRUE),
    MedianDailyYield = median(AvgDailyYieldCow, na.rm = TRUE),
    DailyYield_Q25 = quantile(AvgDailyYieldCow, 0.25, na.rm = TRUE),
    DailyYield_Q75 = quantile(AvgDailyYieldCow, 0.75, na.rm = TRUE),
    .groups = "drop"
  )

```

DF.SummaryByCalvingEase

```

## # A tibble: 2 x 10
##   CalvingEase   Cows AvgMilkingDaysPerCow MedianMilkingDaysPer~1 MilkingDays_Q25
##   <chr>      <int>             <dbl>                  <int>            <dbl>
## 1 11 Lätt, ut~     15              236.                 291            136.
## 2 12 Lätt, me~     11              255.                 298            262
## # i abbreviated name: 1: MedianMilkingDaysPerCow
## # i 5 more variables: MilkingDays_Q75 <dbl>, AvgDailyYield <dbl>,
## #   MedianDailyYield <dbl>, DailyYield_Q25 <dbl>, DailyYield_Q75 <dbl>

```

Step 11: Plot daily milk yield after calving

We will now plot daily milk yield for each cow in the dataset.

Instead of using lines (which can become difficult to interpret when many cows overlap), we plot each milking day as a dot.

The dots are coloured by CalvingEase.

Note that step 1-3) Are rather repetitive examples of joining while step 4 is the new step with plotting.

```

# We plot milk yield by Days in Milk (DIM) instead of calendar date.
# ggplot2 is used for plotting
# Read more here: https://ggplot2.tidyverse.org/

```

```

# DIM = number of days since calving for each cow.
# This makes cows comparable even if they calved on different dates.

```

```

# IMPORTANT:
# DF.DailyMilkPerCow currently only contains StartDate (milking date), not CalvingDate.
# We therefore re-create a calving table server-side and join it to the daily milk table,
# then calculate DIM in R after collect().

# 1) Calving table (server-side, not collected)
DF.CalvingWithOffspring_db <- con %>%
 tbl(in_catalog("basregDWH", "apiSci", "factCattleReproductionView")) %>%
  filter(
    !is.na(Calving),
    StartDate >= as.Date("2021-01-01"),
    StartDate <= as.Date("2021-01-31")
  ) %>%
  transmute(
    Farm = Farm,
    SE_Number = SE_Number,
    CalvingDate = StartDate,
    CalvingEase = CalvingEase
  )

# 2) Daily milk per cow per date (R object from Step 10)
# Make sure StartDate is Date class
DF.DailyMilkPerCow <- DF.DailyMilkPerCow %>%
  mutate(StartDate = as.Date(StartDate))

# 3) Join daily milk to calving table and calculate Days in Milk (DIM)
DF.DailyMilkWithDIM <- DF.DailyMilkPerCow %>%
  inner_join(
    DF.CalvingWithOffspring_db %>% collect(),
    by = c("SE_Number", "CalvingEase")
  ) %>%
  mutate(
    CalvingDate = as.Date(CalvingDate),
    DIM = as.integer(StartDate - CalvingDate) # days since calving
  ) %>%
  filter(DIM >= 0, DIM <= 365) # keep the first 365 days after calving

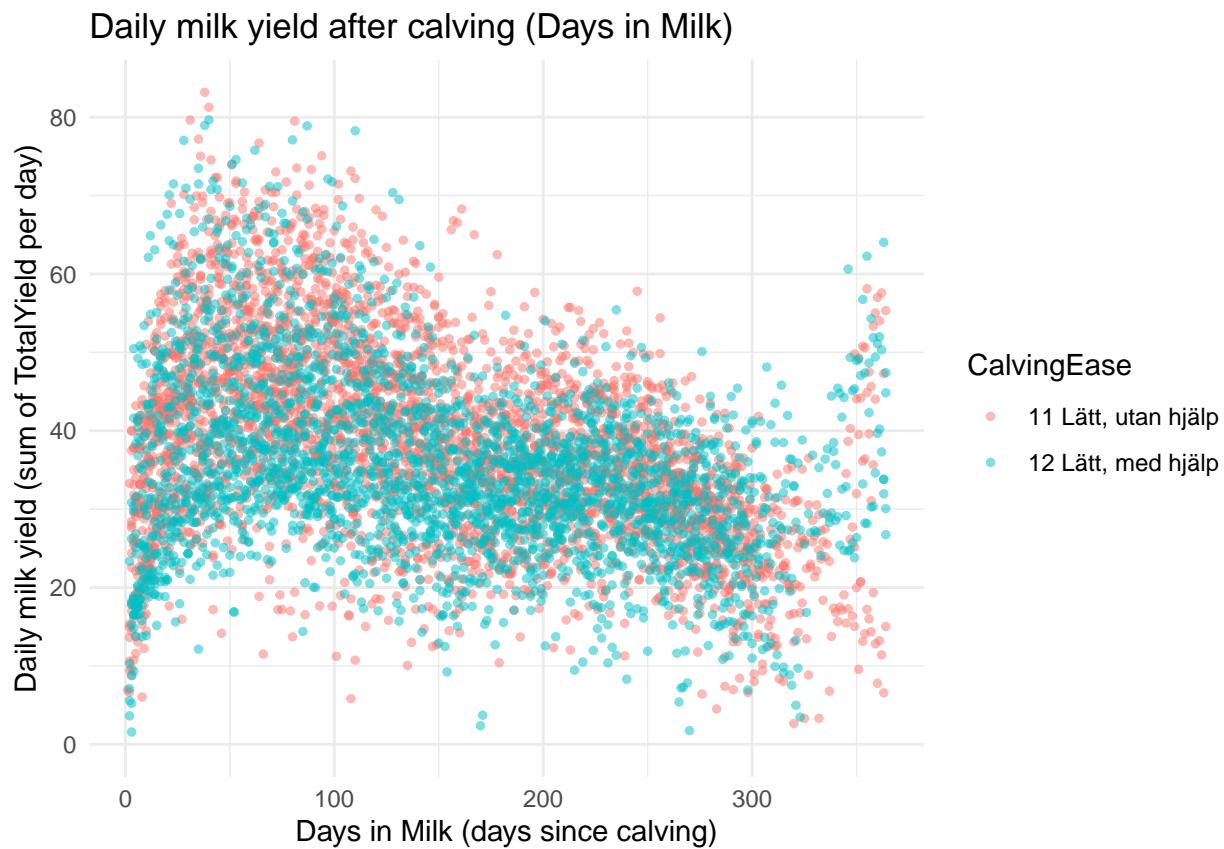
# 4) Plot dots (easier to interpret than lines when many cows overlap)
ggplot(
  DF.DailyMilkWithDIM,
  aes(x = DIM, y = DailyTotalYield, colour = CalvingEase)
) +
  geom_point(alpha = 0.5, size = 1) +
  scale_y_continuous(limits = c(0, NA)) + # Y axis starts at 0
  labs(
    title = "Daily milk yield after calving (Days in Milk)",
    x = "Days in Milk (days since calving)",
    y = "Daily milk yield (sum of TotalYield per day)",
  )

```

```

    colour = "CalvingEase"
) +
theme_minimal()

```



Step 12: Save results and close the database connection

We will now save the milking dataset to your computer as a CSV file. You can imagine that we later will investigate the appropriate end of the lactation as we see several animals having started their next lactation in the plot.

After saving, we close the database connection. This is good practice and releases resources on both your computer and the Basreg server.

```

# Save the milking data as a CSV file in your working directory
# The file will be saved in the folder returned by getwd()
# Read more here: https://www.rdocumentation.org/packages/utils/topics/write.csv
write.csv(DF.MilkingAfterCalving, "DF_MilkingAfterCalving.csv", row.names = FALSE)

# Close the database connection when you are done
# Using dbIsValid() avoids errors if the connection is already closed
# Read more here:
# - dbDisconnect(): https://dbi.r-dbi.org/reference/dbDisconnect.html

```

```
# - dbIsValid():    https://dbi.r-dbi.org/reference/dbIsValid.html
if (DBI::dbIsValid(con)) DBI::dbDisconnect(con)
```