Optimierte Speicherung von Wetterdaten in einem Datenbankmanagementsystem

Friedjof Noweck

März *2021*

Inhaltsverzeichnis

Einleitung

Im Rahmen einer einjähriger Projektarbeit baute ich 2018 & 2019 ein autonomes Gewächshaus[1]. Auf dieses Projekt baut diese Projektarbeit auf. Mein Gewächshaus sammelt nicht nur Wetterdaten wie Temperatur und Luftfeuchtigkeit, sondern auch die Bodenfeuchtigkeit, sowie den Zeitpunkt einer Bewässerung und der Belüftung. Diese Daten werden teilweise für Entscheidungen und teilweise zur Dokumentation gebraucht. Zum Beispiel werden die Pflanzen bewässert, wenn die Bodenfeuchtigkeit zu niedrig ist oder es wird die Luftzirkulation verbessert, wenn es zu warm ist. Diese Aktionen werden aus dem Durchschnitt der gesammelten Daten über einen bestimmten Zeitraum ermittelt. Damit wird sichergestellt, dass einzelne Messungenauigkeiten nicht situationsabhängige unangemessene Aktionen auslösen. Daraus lässt sich ableiten, dass die Daten längerfristig gespeichert werden sollen.

Die oben beschriebenen Aufgaben der Datenspeicherung kann gut durch eine Datenbank abgedeckt werden. Die bisherige Datenbankstruktur kann optimiert werden, da beispielsweise keine vollständige Normalisierung durchgeführt wurde. Zudem ist die Netzwerkstruktur nicht für eine größere Skalierung geeignet, da die Datenbank direkt auf dem aktuellen Steuerungssystem[2] liegt. Dadurch wird eine sinnvolle Abfrage weiterer Clients, damit sind weitere Systeme gemeint mit Schreib- und Leserechten, unmöglich. Doch nicht nur weitere Zugänge zu diesen Daten werden erschwert, sondern auch die sinnvolle Administration der Daten. So muss, um die Datenbank zu verwalten, eine aktive Verbindung zu diesem Steuerungssystem aufgebaut werden. Auch kann so die Stärke von heutigen grafischen Verwaltungstools nicht genutzt werden.

Die Datenintigrität oder sogar die gesamte Datenbank ist aktuell ohne Backups in Gefahr bei einem Ausfall dieses Clients. Dies kann besonders auf diesen kleinen Steuerungssystemen der Fall sein, da diese meist nicht über eine besonders gute Fehlerkorrektur verfügen. Auch ist ein passwortgeschützter Zugriff auf die Daten nicht möglich. Dies liegt daran, dass die Daten unverschlüsselt auf dem Steuerelement liegen und diese auch nicht durch eine Berechtigungsabfrage von potenziell gefährlichen Programmen geschützt sind. Dadurch wird das Risiko einer Übernahme der kompletten Datenbank inakzeptabel hoch.

[1] Mehr zu dieser Projektarbeit unter https://automated.noweck.info.

[2] Ein Bild des aktuellen Steuerungssystems HIER oder auch auf meinem Blog [https://automated.noweck.info]

Die bisherige Struktur der Datenbank

Die in ROT angegebenen Tabellen sind die Tabellen in denen Systemwerte gespeichert werden (mehr zu diesen in Punkt eins und zwei). Die anderen vier GRÜN gekennzeichneten Tabellen sind zur Speicherung der Wetterdaten da, welche durch die verschiedenen Wettersensoren erfasst werden (mehr unter den Punkten drei bis sechs).

Die bisherige Datenbank enthält sechs Tabellen. Diese Tabellen dokumentieren folgende Datenpunkte:

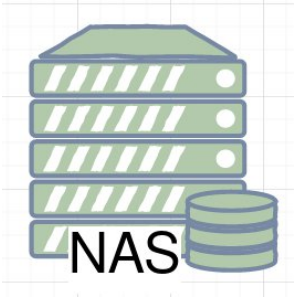
1. **Belueftung** dokumentiert die Belüftungen des Gewächshauses.
   Die ID zur Eindeutigkeit als Integer, das Datum als Text, den Öffnungszustand (1 = offen, 0 = geschlossen) als Integer und die Woche im Jahr als Integer, in welcher diese Messung vorgenommen wurde.
2. **Bewaesserung** dokumentiert die Bewässerungen.
   Die ID zur Eindeutigkeit als Integer, das Datum als Text, die Bewässerungen immer als eins bei einer Bewässerung und die Woche im Jahr als Integer, in welcher diese Messung vorgenommen wurde.
3. **Luft** dokumentiert die Luftfeuchtigkeit (der Name der Tabelle ist nicht optimal).
   Die ID zur Eindeutigkeit als Integer, das Datum als Text, der Messpunkt als Integer und die Woche im Jahr als Integer, in welcher diese Messung vorgenommen wurde.
4. **Wind** wurde benutzt, um die Windstärke zu messen. Der Windsensor ist aber nicht mehr in Betrieb.
   Die ID zur Eindeutigkeit als Integer, das Datum als Text, der Messpunkt als Umdrehungen in einer bestimmten Zeitspanne als Integer und die Woche im Jahr als Integer, in welcher diese Messung vorgenommen wurde.
5. **Temperatur** zur Dokumentation der Lufttemperatur innerhalb des Gewächshauses.
   Die ID zur Eindeutigkeit als Integer, das Datum als Text, der Messpunkt als Integer in Grad Celsius und die Woche im Jahr als Integer, in welcher diese Messung vorgenommen wurde.
6. **Feuchtigkeit** zur Dokumentation der Bodenfeuchtigkeit.
   Die ID zur Eindeutigkeit als Integer, das Datum als Text, der Messpunkt in Prozent als Integer und die Woche im Jahr als Integer, in welcher diese Messung vorgenommen wurde.

Bei einer genauen Betrachtung fällt auf, dass hier einige Daten nicht benötigt werden, oder der Datentyp nicht optimal gewählt wurde. Die Anzahl der Woche im Jahr in dem die Messung

gemacht wurde ist zum Beispiel unnötig, weil diese durch das Datum herausgefunden werden kann. Auch sollte das Datum nicht als Text sondern als Zeitpunkt gespeichert werden. Dazu bietet sich der Datentyp Datetime an. Auch sollten die Messpunkte nicht als Ganzzahl (Integer) gespeichert werden werden, da die Temperatur zum Beispiel auch Kammerstellen beinhaltet. In diesem Fall sollte der Datentyp Float gewählt werden.

Auch werde ich die Tabelle „Wind“ nicht in die neue Implementierung mit aufnehmen, weil diese nur zum Testen meines Windsensors angelegt wurde und schon seit mehr als einem Jahr nicht mehr benutzt wird. Bei der Tabelle Bewässerung fällt zu der Woche, wie in allen Tabellen auch der Parameter „Bewaesserungen“ weg. Zum Einen gibt der Name nicht wirklich den Verwendungszweck wieder und zum Anderen ist dieser Parameter überflüssig, weil hier kein richtiger Wert gespeichert wird. Bei der Bewässerung ist eigentlich nur der Zeitpunkt entscheidend. Der Parameter der Tabelle „Belueftung“, welcher den Status der Belüftung speichert kann besser durch den Datentypen Boolean ausgedrückt werden. Dabei wird true gepsichert, wenn die Frontklappe offen und false, wenn sie geschlossen ist. So muss keine Zahl in einen Wahrheitswert umgewandelt werden und die Möglichkeit eines Fehlers an dieser Stelle wird minimiert.

Ein NAS als sicherer Speicherort



Ein **NAS** (**N**etwork **A**ttached **S**torage) ist ein netzwerkfähiger Speicherort. Dieser ist durch seinen Einsatzort auf die sichere Speicherung von Daten optimiert. Zum einen werden die Daten verschlüsselt und zum Anderen werden die dortigen Informationen auch gegen Schreibfehler geschützt durch zum Beispiel ein Raid[3]-System. Ein NAS ermöglicht zusätzlich, je nach Model, nicht nur einen gesicherten Zugang zu den dort liegenden Daten, sondern bietet auch weitere Funktionalitäten. In diesem Fall nutze ich ein NAS einmal zur Speicherung meiner Datenbanken und auch als Server meiner MariaDB als Datenbankmanagementsystem.

Die MariaDB als DBMS[4]

Ich habe mich bei diesem Projekt für das häufig genutzt und freie, relationale Open-Source DBMS **MariaDB** entschieden. Die MariaDB läuft auf der Containerplattform Docker[5] welche auf dem NAS installiert ist und die Verwaltung von Servern deutlich erleichtert. Durch den folgenden Befehl wird der MariaDB-Server einfach und schnell aufgesetzt.

```
$ docker run --name my_mariadb
        -p 3306:3306
        -v /pfad/zum/speicherort/der/db‘s:/var/lib/mysql
        -v /pfad/zu/den/mariadb/config‘s:/etc/mysql/conf.d
        -e MYSQL_ROOT_PASSWORD=my-secret-pw
        -d mariadb:tag
```

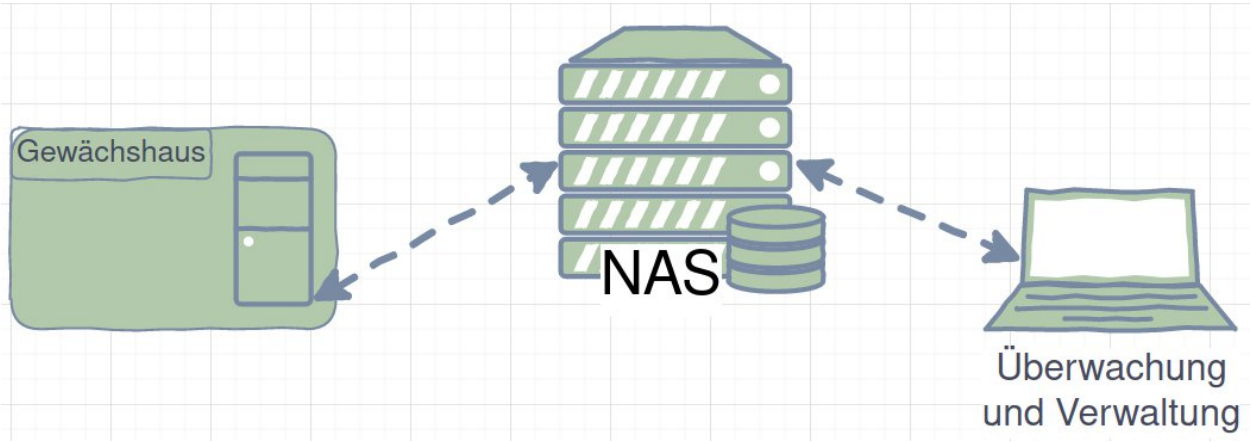
[3] Ein RAID-System dient zur Organisation mehrerer Speichermedien. Es wird häufig verwendet, um die Organisation von Daten sicherer zu gestalten.

[4] **D**aten**b**ank**m**anagement**s**ystem

[5] Docker ist eine freie Software zur Isolierung von Anwendungen mit Hilfe von Containervirtualisierung. Eine stichpunktartige Erklärung im Rahmen meines Referates zu Docker liegt bei.

Der „MYSQL_ROOT_PASSWORD“ setzt das Administrator Passort und sollte eben so angepasst werden, wie auch die Pfade zu den Speicherorten der Datenbanken und der zugehörigen Konfigurationen. Auch erfüllt die MariaDB meine Anforderungen an einen passwortgeschützten Zugriff auf die zu verwaltenden Datenbanken. Der Administrator kann so den einzelnen Clients einen eigenen Zugang zu den Datenbanken gewähren. Dabei sollte jeder Nutzer nur die Rechte besitzen, die unbedingt notwendig sind. Dadurch kann ein potenzieller Angreifer bei einer Übernahme eines Clients nie in Besitz aller Daten gelangen. Der Zugang des Administrators sollte daher besonderen Schutz genießen.

Die Netzwerkarchitektur



Abbild der Ziel-Netzwerkarchitektur

Durch diese Netzwerkstruktur wäre es möglich, dass weitere Steuereinheiten Zugriff zu dieser Datenbank bekommen könnten und so die Skalierbarkeit deutlich verbessert wird. Dies vereinfacht auch die Überwachung und Verwaltung des Datenbankmanagementsystems, da so der Administrator schnellen Zugriff auf die MariaDB über das Netzwerk erhalten kann. Bei der Administration ist eine grafische Benutzeroberfläche von Vorteil. Ich benutze daher die Community Edition von **DBeaver.** Mit dieser kann man sehr einfach und schnell mit der IP-Adresse, dem Port[6] der MariaDB und den Zugangsdaten eine Verbindung aufbauen.

DBeaver

DBeaver ist ein Tool zum Verwalten von vielen verschiedenen relationale Datenbanksystemen. Da DBeaver einfach zu bedienen ist und gut mit der MariaDB zusammenarbeiten kann, habe ich mich für dieses Verwaltungstool in der kostenlosen Community Variante entschieden.

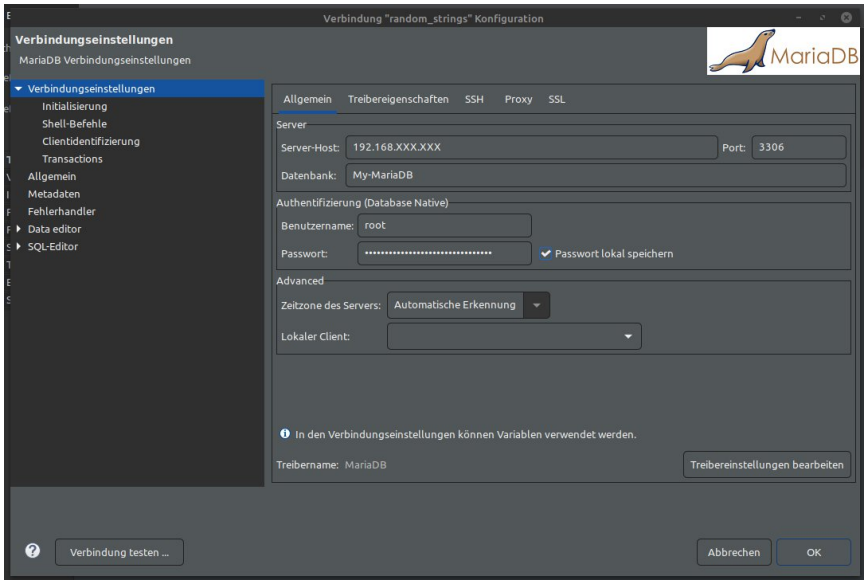
Eine Verbindung aufbauen

Eine Verbindung mit der laufenden MariaDB aufzubauen geht recht einfach. Nach dem öffnen von DBeaver kann man durch den Plus-Button oben links eine neue Verbindung hinzufügen. Im Anschluss wird nach dem DBMS[7]-Typen gefragt. Nach dieser Auswahl sollen nun die

[6] Der Standartport bei einer MariaDB ist 3306.

[7] Datenbankmanagementsystem

Verbindungsdaten eingeben. Also die IP, der Port , der Benutzername[8] und natürlich das gewählte Passwort.



Nach dem erfolgreichen Verbindungstest in der linken unteren Ecke, kann die Verbindung mit „OK“ bestätigt werden. Nun steht die Verbindung zu der MariaDB und kann zum Anlegen von Datenbaken, Tabellen, weiterer Nutzer und vielem mehr genutzt werden. Es kann auch von mehreren Rechnern zur gleichen Zeit über einen Nutzer zugegriffen werden. So lässt sich auch die Arbeit in einem Team realisieren.

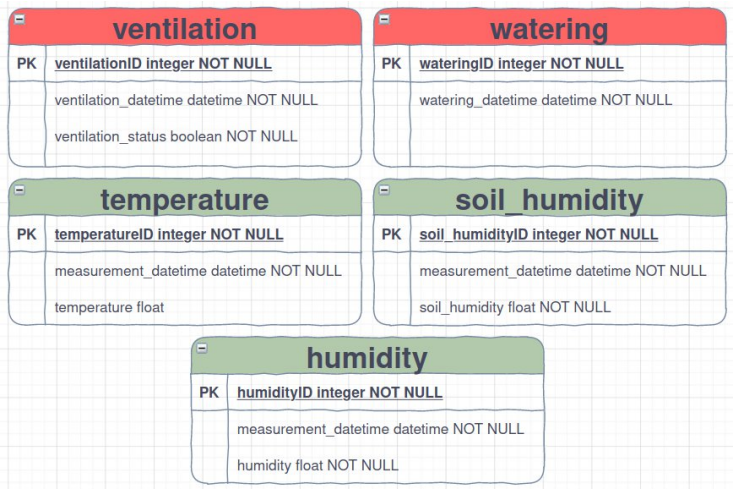
Ein kurzer Einblick

Über einen Rechtsklick auf die Verbindungsanzeige können sofort mehrere Aktionen ausgeführt. Der SQL-Editor ermöglicht die Verwaltung per SQL-Skript oder auch direkt über die SQL-Konsole. Zudem können einige Verbindungseinstellung geändert werden. Zur visuelle Darstellung gelangt man durch das Ausklappen der Verbindung. Dort werden dann weitere Punkte angezeigt. Unter „Databases“ befinden sich zum Beispiel nicht nur die eigenen Datenbanken, sondern auch Datenbanken des Systems der MariaDB. Diese Datenbaken können vom Administrator eingesehen werden, sollten aber nicht einfach so geändert werden. Unter „User“ finden sich die einzelnen Nutzer und die Option weitere anzulegen und noch einiges mehr. Unter „Administer“ finden sich die einzelnen aufgebauten Sitzungen „Sessions“. Unter dem letzten Punkt liegen, wie der Name von sagt, ausschließlich Systeminformationen der MariaDB.

[8] In diesem Fall den Superuser [Administrator] „root“.

Der Aufbau dieser neuen Datenbank

Die MariaDB kann, wie schon beschrieben, mehrere Datenbanken und Nutzer verwalten. Im Folgenden beschreibe ich den Aufbau der Datenbank „wetterdaten“, in welcher die gesammelten Daten des Gewächshauses abgespeichert werden sollen. Natürlich kann die Datenbank noch um weitere Tabellen erweitert werden.



Auf dem Schaubild habe ich bei der Darstellung der fünf Tabellen in zwei Arten unterschieden.

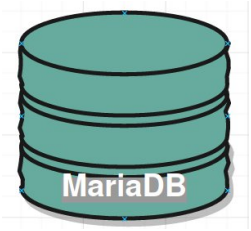
In **ROT** die Tabellen, welche mit Systemdaten beschrieben werden. Die Tabelle „ventilation“ speichert das Datum und die Uhrzeit der Aktion, sowie eine genaue ID des Ereignisses und den Status der Belüftung. Dazu muss man wissen, dass die Luftzirkulation durch eine große Frontklappe veranlasst wird. durch den Datentypen Boolean sind genau zwei Zustände möglich. „true“ ist bedeutet dabei, dass die Frontklappe geöffnet und „false“, dass diese geschlossen ist. Die Tabelle „watering“ speichert dagegen nur die ID der Bewässerung, sowie den Zeitpunkt der Bewässerung als Datentyp „datetime“, welcher natürlich wie auch schon bei der vorhergegangenen Tabelle angegeben werden muss. Da das Wasserventil nach einer definierten Zeit von selber ausgeht, ist kein Status nötig.

In **GRÜN** die Tabellen, welche direkt mit aktuellen Daten der Sensoren beschrieben werden. In der Tabelle „temperature“ wird die aktuelle Temperatur in Abständen von 10 min gespeichert. Diese Tabelle enthält ebenfalls eine ID als Primärschlüssel, sowie den Zeitpunkt der Messung und natürlich auch der ermittelten Temperatur als float, um die Zahl auch mit Kammerstellen angeben zu können. Aufgrund dieser Daten wird die Belüftung gesteuert. Die Tabelle „soil_humidity“ speichert neben der ID als Primärschlüssel ebenfalls den Messzeitpunkt und den Durchschnitt der ermittelten Bodenfeuchtigkeit aller Sensoren. Auf Grundlage dieser Daten wird die Bewässerung gesteuert. Eine mögliche Erweiterung dieser Tabelle wäre es nicht nur den Durchschnitt, sondern alle Daten der vier Sensoren in zusätzlichen Spalten zu speichern. Dadurch bekäme man einen genaueren Überblick über die einzelnen Messwerte. Dadurch würde auch die Spalte des Durchschnitts überflüssig. In der fünften Tabelle wird zusätzlich noch die ermittelte Luftfeuchtigkeit gespeichert. Diese Tabelle beinhaltet wie auch schon die beiden vorherigen Tabellen eine ID, einen Zeitpunkt und einen Messwert, welcher die Luftfeuchtigkeit des angegebenen Zeitpunktes repräsentiert. Dieser Datenpunkt triggert vorerst keine Aktion, könnte aber in der Zukunft zur Berechnung einer gefühlte Innentemperatur dienen und somit für die Kontrolle der Luftzirkulation genutzt werden.

Alle IDs werden automatisch aufsteigent generiert und sind die Primärschlüssel der einzelnen Tabellen. Aufgrund der automatischen generierung ist der Datentyp der IDs als Integer zu setzen.

Die Tabellen können nicht wirklich direkt in eine Beziehung gesetzt werden. Deshalb sind auf diesem ERD keine Beziehungen angegeben. Die Verbindungen der einzelnen Tabellen werden eher im Hintergrund durch die zuständige Systeme geregelt.

Die Implementierung



Nach dem starten der MariaDB als DBMS und dem kurzen Überblick über das Verwaltungstool DBeaver, ist das anlegen der Datenbanken und der fünf Tabellen der logische nächste Schritt.

Die neue Datenbank anlegen

Dazu kann wie oben beschrieben die SQL-Konsole geöffnet werden und der folgende Befehl ausgeführt werden:

```
CREATE DATABASE wetterdaten
```

Nach dem erstellen, kann durch einen Rechtsklick auf den Menüpunkt „Databases“ die vorhandenen Datenbankenanzeige Aktualisiert werden. Nun sollte dort die neue Datenbank „wetterdaten“ angezeigt werden.

Durch einen doppelklick kann man nun einen Blick in die leere Datenbank werfen. Im Folgenden zeige ich, wie diese mit den oben beschriebenen Tabellen gefüllt werden kann.

Anlegen der Tabellen

Folgende Befehle legen die einzelnen Tabellen wie oben beschrieben an:

Tabelle „ventilation“

```
CREATE TABLE wetterdaten.ventilation (
        ventilationID integer auto_increment NOT NULL,
        ventilation_datetime DATETIME NOT NULL,
        ventilation_status BOOL NOT NULL,
        PRIMARY KEY (ventilationID)
)
ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

Da wir uns auf der obersten Ebene befinden beim Ausführen des Befehls, wird vor dem Tabellennamen der Name der Datenbank geschrieben. Die beiden Namen werden durch einen Punkt von einander getrennt.

Im Anschluss folgen die einzelnen Spalten der Tabelle mit dem zugehörigen Datentypen. Auch müssen diese Werte gesetzt werden „NOT NULL“. Die „ventilationID“ ist eine Zahl und wird

automatisch generiert. Damit das klappt, muss die „ventilationID“ als Primärschlüsseln festgelegt werden.

Auch erkennt man, dass hier neben dem Anlegen der Spalten einzelne Parameter mitgegeben werden müssen. Zum Einen ist das der zu verwendende Zeichensatz und zum Anderen das System in dem die Daten gespeichert werden sollen. Als alternative zur InnoDB können zum Beispiel folgende Formate angegeben werden: CSV (Einfaches Tabellenformat lesbar im dem Texteditor oder auch mit Excel), Aria, Memory und einiges mehr.

Diese Erklärungen der einzelnen Parameter dieses SQL-Befehls lassen sich auf die folgenden Aufrufe Übertragen.

Tabelle „watering“

```
CREATE TABLE wetterdaten.watering (
        wateringID integer auto_increment NOT NULL,
        watering_datetime DATETIME NOT NULL,
        PRIMARY KEY (wateringID)
)
ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

Tabelle „temperature“

```
CREATE TABLE wetterdaten.temperature (
        temperatureID integer auto_increment NOT NULL,
        measurement_datetime DATETIME NOT NULL,
        temperature FLOAT NOT NULL,
        PRIMARY KEY (temperatureID)
)
ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

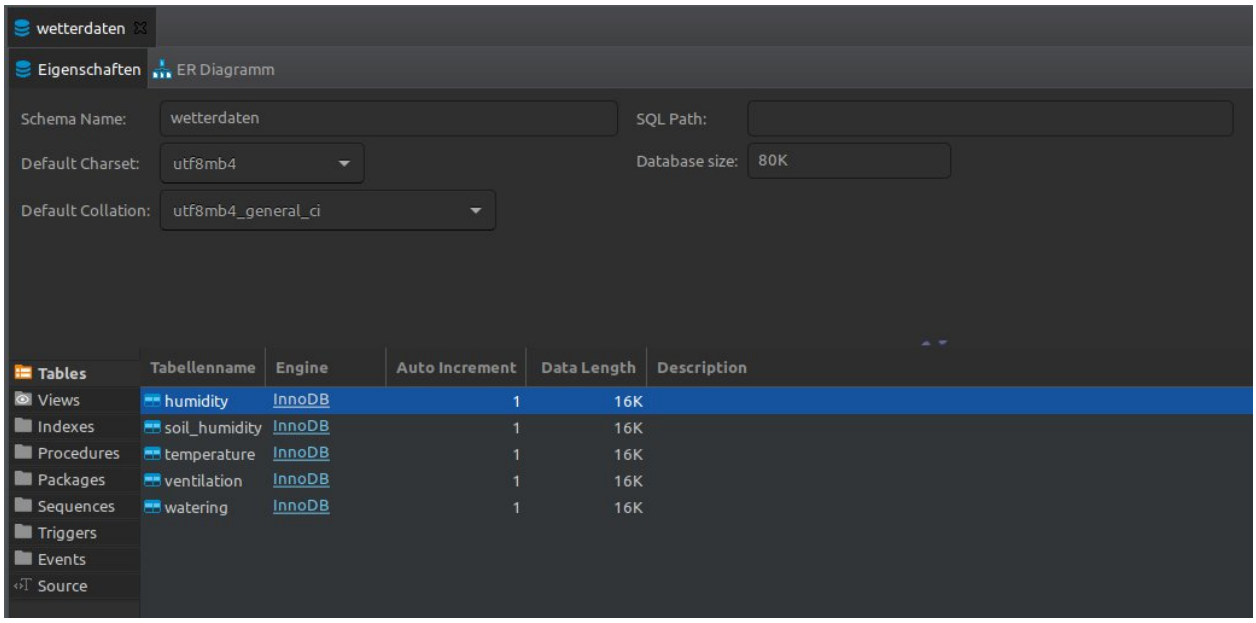
Tabelle „soil_humidity“

```
CREATE TABLE wetterdaten.soil_humidity (
        soil_humidityID integer auto_increment NOT NULL,
        measurement_datetime DATETIME NOT NULL,
        soil_humidity FLOAT NOT NULL,
        PRIMARY KEY (soil_humidityID)
)
ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

Tabelle „humidity“

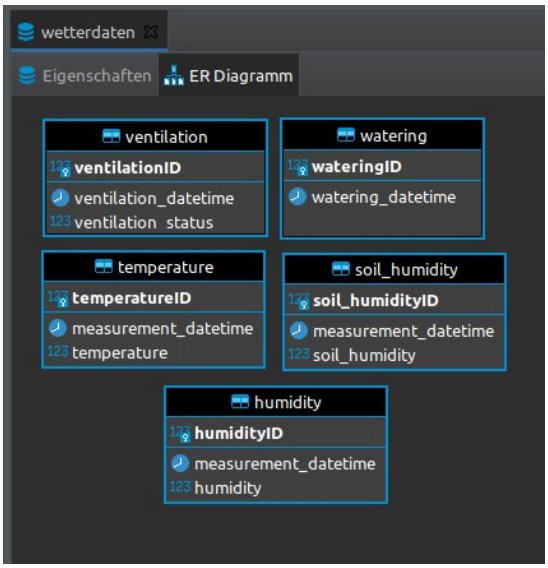
```
CREATE TABLE wetterdaten.humidity (
        humidityID integer auto_increment NOT NULL,
        measurement_datetime DATETIME NOT NULL,
        humidity FLOAT NOT NULL,
        PRIMARY KEY (humidityID)
)
ENGINE=InnoDB
DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_general_ci;
```

Nun werden die angelegten Tabellen in der Wetterdaten Datenbank angezeigt. Sollte dies nicht der Fall sein, kann die Ansicht nochmals aktualisiert werden.



Unter dem Menüpunkt „ER Diagramm“ lassen sich auch die einzelnen Tabellen ggf. mit Beziehungen anzeigen.

Wie schon erwähnt gibt es in dieser Modellierung keine Beziehungen zwischen den einzelnen Tabellen.



Anmerkung zu „*auto_increment*“

Wenn einzelne oder sogar alle Datenpunkte aus einer Tabelle gelöscht werden, wird trotzdem der Zähler (auto_increment) weiter hoch zählen. Wenn also zum Beispiel alle 1000 Messwerte gelöscht werden, würde der Zähler beim ersten Wert die ID 1001 wählen. Um diesen Zähler wieder zurück zu setzen, ist folgender Befehl notwendig.

```
ALTER TABLE {Datenbank}.{Tabelle} AUTO_INCREMENT = 1
```

Das Befüllen der Datenbank

Das Befüllen der Datenbank übernimmt nun mein Gewächshaus. Dafür muss die aktuelle Software angepasst werden. Hier für werden für jede Tabelle INSERT Befehle benötigt.

INSERT

Zum Eintragen von neuen Daten kann folgende Vorlage genutzt werden.

```
INSERT INTO wetterdaten.{Tabellen Name} ({Parameter Name, Parameter Name})
VALUES ({Parameter Daten, Parameter Daten});
```

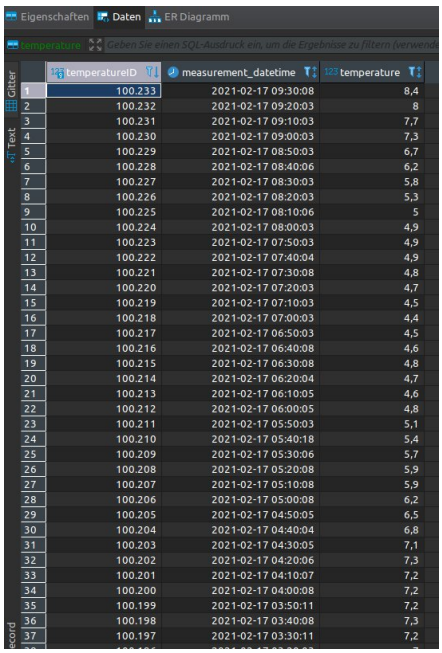
Durch das automatische generieren des PK[9] (der ID) muss dieser Parameter nicht angegeben werden.

Ein Beispiel

```
INSERT INTO wetterdaten.temperature (measurement_datetime,temperature)
VALUES ('2021-02-25 15:38:47',17.4);
```

[9] Primary Key => Primärschlüssel

Da ich die alten bisher gesammelten Daten übertragen wollte, habe ich mir ein kleines Python Script gebastelt, welches die mehreren 100000 Messwerte automatisiert für mich überträgt. Im Anschluss daran kann man sich die Daten natürlich in DBeaver anschauen. Links ist nun die Tabelle aller Temperatur Messdaten zu sehen. Insgesamt komme ich auf 253.000 Messwerte über einen Zeitraum von mehr als zweieinhalb Jahren.



	temperatureID	measurement_datetime	temperature
1	100.233	2021-02-17 09:30:08	8,4
2	100.232	2021-02-17 09:20:03	8
3	100.231	2021-02-17 09:10:03	7,7
4	100.230	2021-02-17 09:00:03	7,3
5	100.229	2021-02-17 08:50:03	6,7
6	100.228	2021-02-17 08:40:06	6,2
7	100.227	2021-02-17 08:30:03	5,8
8	100.226	2021-02-17 08:20:03	5,3
9	100.225	2021-02-17 08:10:06	5
10	100.224	2021-02-17 08:00:03	4,9
11	100.223	2021-02-17 07:50:03	4,9
12	100.222	2021-02-17 07:40:04	4,9
13	100.221	2021-02-17 07:30:08	4,8
14	100.220	2021-02-17 07:20:03	4,7
15	100.219	2021-02-17 07:10:03	4,5
16	100.218	2021-02-17 07:00:03	4,4
17	100.217	2021-02-17 06:50:03	4,5
18	100.216	2021-02-17 06:40:08	4,6
19	100.215	2021-02-17 06:30:08	4,8
20	100.214	2021-02-17 06:20:04	4,7
21	100.213	2021-02-17 06:10:05	4,6
22	100.212	2021-02-17 06:00:05	4,8
23	100.211	2021-02-17 05:50:03	5,1
24	100.210	2021-02-17 05:40:18	5,4
25	100.209	2021-02-17 05:30:06	5,7
26	100.208	2021-02-17 05:20:08	5,9
27	100.207	2021-02-17 05:10:08	5,9
28	100.206	2021-02-17 05:00:08	6,2
29	100.205	2021-02-17 04:50:05	6,5
30	100.204	2021-02-17 04:40:04	6,8
31	100.203	2021-02-17 04:30:05	7,1
32	100.202	2021-02-17 04:20:06	7,3
33	100.201	2021-02-17 04:10:07	7,2
34	100.200	2021-02-17 04:00:08	7,2
35	100.199	2021-02-17 03:50:11	7,2
36	100.198	2021-02-17 03:40:08	7,3
37	100.197	2021-02-17 03:30:11	7,2

Durch die Optimierung der Datenbank konnte ich den Speicherverbrauch um 30,9% senken im Vergleich zur alten Implementierung.

Tabellen Optimierung

Gegen Ende dieses Projektes habe ich Kommentare zu den einzelnen Tabellen hinzugefügt. Dadurch sind die einzelnen Tabellen besser verständlich. Dafür habe ich folgenden Befehl verwendet:

```
ALTER TABLE wetterdaten.humidity
COMMENT='Die Luftfeuchtigkeit in meinem Gewächshaus';

ALTER TABLE wetterdaten.soil_humidity
COMMENT='Die Bodenfeuchtigkeit in meinem Gewächshaus';

ALTER TABLE wetterdaten.temperature
COMMENT='Die Lufttemperatur in meinem Gewächhaus';

ALTER TABLE wetterdaten.ventilation
COMMENT='Die Belüftung in meinem Gewächshaus';

ALTER TABLE wetterdaten.watering
COMMENT='Die Bewässerung in meinem Gewächshaus';
```

Abschließend bekommt man folgenden Überblick über die Tabellen und deren Inhalt.

Schema Name: wetterdaten SQL Path:

Default Charset: utf8mb4 Database size: 9,8M

Default Collation: utf8mb4_general_ci

Tables
- Views
- Indexes
- Procedures
- Packages
- Sequences
- Triggers
- Events
- Source

Tabellenname	Engine	Auto Increment	Data Length	Description
humidity	InnoDB	91.325	3,5M	Die Luftfeuchtigkeit in meinem Gewächshaus
soil_humidity	InnoDB	55.930	2,5M	Die Bodenfeuchtigkeit in meinem Gewächshaus
temperature	InnoDB	100.234	3,5M	Die Lufttemperatur in meinem Gewächhaus
ventilation	InnoDB	774	48K	Die Belüftung in meinem Gewächshaus
watering	InnoDB	4.496	160K	Die Bewässerung in meinem Gewächshaus

Bonus

Durch die genaue Archivierung kann man nun verschiedene Abfragen tätigen.

Messpunkte in einem bestimmten Zeitraum

```
select * from wetterdaten.temperature
where measurement_datetime between '2019-07-26 07:30:00' and '2019-07-27 00:00:00'
```

```
select * from wetterdaten.temperature
where measurement_datetime between '2019-07-26 07:30:00' and '2019-07-27 00:00:00'
```

temperature 1

select * from wetterdaten.temperature where measurement Geben Sie einen SQL-Ausdruck ein, um die Ergebnisse

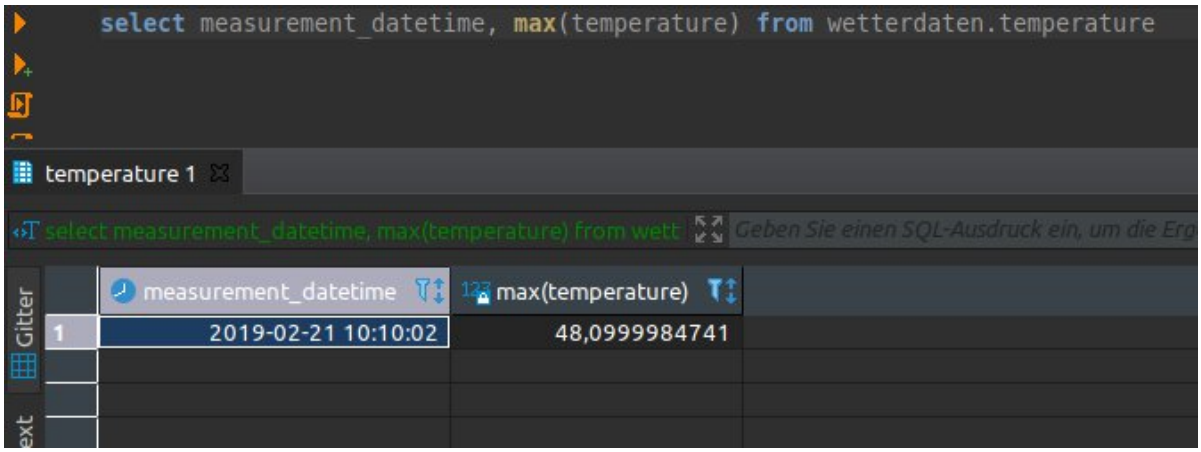
Gitter

Text

	temperatureID	measurement_datetime	temperature
1	20.539	2019-07-26 07:30:10	30,3
2	20.540	2019-07-26 07:40:03	35,8
3	20.541	2019-07-26 07:50:08	36
4	20.542	2019-07-26 08:00:24	37,1
5	20.543	2019-07-26 08:10:03	38,5
6	20.544	2019-07-26 08:20:11	39
7	20.545	2019-07-26 08:30:08	35,4
8	20.546	2019-07-26 08:40:08	31,1
9	20.547	2019-07-26 08:50:14	30,1
10	20.548	2019-07-26 09:00:06	29,9
11	20.549	2019-07-26 09:10:06	32,1
12	20.550	2019-07-26 09:20:03	34,2
13	20.551	2019-07-26 09:30:06	36,2
14	20.552	2019-07-26 09:40:09	40,9
15	20.553	2019-07-26 09:50:03	44,7
16	20.554	2019-07-26 10:00:06	46,3
17	20.555	2019-07-26 10:10:13	46,9
18	20.556	2019-07-26 10:20:08	43,9
19	20.557	2019-07-26 10:30:03	41
20	20.558	2019-07-26 10:40:18	39,4
21	20.559	2019-07-26 10:50:11	39,5
22	20.560	2019-07-26 11:00:19	39,5
23	20.561	2019-07-26 11:10:03	39,7
24	20.562	2019-07-26 11:20:06	39,9
25	20.563	2019-07-26 11:30:09	40,1
26	20.564	2019-07-26 11:40:11	40,7
27	20.565	2019-07-26 11:50:11	40,9
28	20.566	2019-07-26 12:00:08	41
29	20.567	2019-07-26 12:10:04	40,9
30	20.568	2019-07-26 12:20:04	41
31	20.569	2019-07-26 12:30:09	41,1
32	20.570	2019-07-26 12:40:06	41,6

Maximalwert aller Messpunkte

```
select measurement_datetime, max(temperature) from wetterdaten.temperature
```



(Vermutlich ein Messfehler)

Quellen

1. Informationen über das DBMS MariaDB [17.02.2021]
   https://de.wikipedia.org/wiki/MariaDB
2. Informationen über das System NAS [17.02.2021]
   https://de.wikipedia.org/wiki/Network_Attached_Storage
3. Die Funktionalität eines RAID-Systems als sichere Verwaltung von Daten [17.02.2021]
   https://de.wikipedia.org/wiki/RAID
4. Genauere Informationen zu Docker auf Wikipedia [17.02.2021]
   https://de.wikipedia.org/wiki/Docker_%28Software%29
5. MariaDB als Docker-Container auf dem Docker Hub [17.02.2021]
   https://hub.docker.com/_/mariadb
6. Genauere Informationen zu DBeaver auf Wikipedia [20.02.2021]
   https://en.wikipedia.org/wiki/DBeaver
7. Mein Internetblog und mehr zu meiner Projektarbeit des autonomen Gewächshauses [17.02.2021]
   https://automated.noweck.info/
8. Das seltzen von Primärschlüsseln unter SQL [20.02.2021]
   https://www.w3schools.com/sql/sql_primarykey.asp
9. Das erschaffen einer neuen Datenbank unter SQL [20.02.2021]
   https://www.mysqltutorial.org/mysql-create-database/
10. Auto Increment zurücksetzen [21.02.2021]
    https://stackoverflow.com/questions/8923114/how-to-reset-auto-increment-in-mysql