HSLU Hochschule Luzern

Projektbericht Semesterprojekt DBS

Frühlingssemester 20 / Gruppe 3

Abgabedatum: 15. Juni 2020

Projektteam:

Joshua Heller	joshua.heller@stud.hslu.ch
Julien Grüter	julien.grueter@stud.hslu.ch
Maurizio Hostettler	maurizio.hostettler@stud.hslu.ch
Stephan Stofer	stephan.stofer@hslu.ch

Abstract

Der vorliegende Bericht beschreibt die Analyse von Luftfremdstoffen im Zusammenhang mit Feuerwerken. Das Projekt wurde im Rahmen des Moduls DBS von der Gruppe 3 im Frühlingssemester 20 erarbeitet.

Die auf einem MySQL Server gespeicherten Messdaten über Luftfremdstoffe lassen sich über ein GUI grafisch anzeigen und auswerten. Der messbare Einfluss von Feuerwerk auf den Feinstaubgehalt in der Luft lässt sich so klar aufzeigen.

Im Visualisierungsgraph lassen sich zwei Messwerte gegenüberstellen und erlauben eine Analyse über den gegenseitigen Einfluss aufeinander.

Inhaltsverzeichnis

Abbildungsverzeichnis

SQL-Statements

Tabellenverzeichnis

1 Einführung

Mit unserem Semesterprojekt und den damit verarbeiteten Daten, möchten wir den direkten Einfluss von Feuerwerk auf die Luftqualität in der Schweiz untersuchen. Hierzu werden vorwiegend Messdaten für Luftfremdstoffe um den 1. August und 31.Dezember analysiert.

Durch den Vergleich der Messdaten in einem Zeitbereich rund um die Feiertage möchten wir in der Lage sein, den direkten Einfluss auf die Luftqualität zu identifizieren und auf Grund dessen eine Empfehlung abzugeben.

2 Datenmanagement

2.1 Datenquelle

Als Datenquelle für unsere Untersuchung dient uns das Nationale Beobachtungsnetz für Luftfremdstoffe «NABEL». Dieses betreibt 16 Messstationen in der Schweiz, welche 13 versch. Luftfremdstoffe, die Temperatur und den Niederschlag messen. Messungen sind ab dem Jahr 2000 im Stundentakt online verfügbar.

Die Daten können als Grafik, HTML- oder CSV-Datei abgefragt und heruntergeladen werden.

https://www.bafu.admin.ch/bafu/de/home/themen/luft/zustand/daten/datenabfrage-nabel.html

```
Station: Jungfraujoch
Hochgebirge
Stundenmittelwerte, PREC: Stundensummen
MEZ/CET
Quelle: NABEL/MeteoSchweiz
Datum/Zeit;O3 [ug/m3]
01.01.2000 01:00;63.8
01.01.2000 02:00;64.9;
01.01.2000 03:00;65.3;
...
...
```

Abbildung 1 Beispiel einer Messung des Niederschlags auf dem Jungfraujoch im CSV-Format.

Die Rohdaten laden wir im CSV-Format. Für den Import in die Datenbank müssen folgende Schritte vorgenommen werden:

1. Download der Daten.
2. Anpassen der Daten via Script.
3. Rohdaten mit zusätzlichen Attributnamen und Attributwerten anreichern.

Die detaillierten Schritte wie die Daten aufbereitet und geladen wurden sind im Anhang 9.2 aus dem Skript zu entnehmen.

2.2 Datenbanktechnologie

Das Team hat sich entschieden für das Projekt einen MySQL Server einzusetzen. Diese Technologie passt zu den Anforderungen und lässt eine sinnvolle und einfache Handhabung der Daten zu.

Folgende Punkte haben zu dem Entscheid beigetragen:

- Die Auswertungen basieren auf rein statischen Daten und müssen nicht hoch performant abgefragt werden.
- Keine Echtzeit Daten
- Überschaubares Datenvolumen
- Die SQL-Queries können direkt zu Auswertungszwecken benutzt werden

Im Gegensatz zu einer NoSQL Lösung ist die Skalierung mit SQL schwieriger. Da im Rahmen des Projektes mit einem initialen Datenbestand gearbeitet wird, erachten wir SQL als die passende Lösung für das vorliegende Unterfangen.

Auch konnten wir im Rahmen des DBS Moduls mit SQL am meisten Erfahrungen sammeln und nutzen die Gelegenheit, auf diese Weise das erlernte Wissen zu verankern.

2.3 Import der Daten

Es gibt zwei verschiedene Arten von Imports für unsere Datenbank. Ein Bulk Load für den initialen Datenimport und ein täglicher Import der aktuellen Daten. Für beide Varianten existiert ein Script. Das Script ruft bei beiden Varianten am Ende das MySQL Statement `LOAD DATA` auf.

LOAD DATA LOCAL INFILE

Für MySQL bietet sich das `LOAD DATA` Statement für Imports von grossen CSV-Dateien an. Die CSV-Datei muss in einem spezifischen Ordner auf dem Serversystem abgelegt werden. Alternativ kann das Keyword `LOCAL` dem Statement hinzugefügt werden, um die Datei vom Clientsystem hochladen zu können. Die Funktion muss jedoch clientseitig in der «my.cnf»-Datei und serverseitig im «msql.cnf» aktiviert werden, indem das Key-Value Attribute «local-infile=1» hinzugefügt wird.

Performance Optimierung des initialen Imports:

Mit folgenden Massnahmen konnte die Performance optimiert werden. Die Importdauer konnte von initial 16 Stunden auf 2 Stunden verkürzt werden.

- Eine grosse CSV-Datei statt viele kleine CSV-Dateien oder INSERTS ist performanter.
- Die «buffer size» wurde von 128MB auf 2GB erhöht, um die vielen Daten importieren zu können.
- Die «unique_check» und «constraint_check» wurden temporär für den Import deaktiviert.
- Das «binary log» wurde temporär für den Import deaktiviert.

```
mysql --login-path=${login_path} -D ${database} --local-infile=1 -N -e
"SET unique_checks = 0;
 SET foreign_key_checks = 0;
 SET sql_log_bin=0;
    LOAD DATA LOCAL INFILE '${folder}/master_import.csv' INTO TABLE messung
        FIELDS TERMINATED BY ';'
        LINES TERMINATED BY '\n'
        (datum, wert, ort_id, groesse_id);
 SET unique_checks = 1;
 SET foreign_key_checks = 1;
 SET sql_log_bin=1;"
```

SQL-Statement 1 LOAD DATA Query für den Datenimport

Referenzdokumentation von MySQL für `LOAD DATA`:
https://dev.mysql.com/doc/refman/8.0/en/load-data.html

3 Datenmodellierung

3.1 Datenmodell (ER)

Nach der Analyse der Daten von «NABEL» haben wir mit Hilfe der dritten Normalform folgendes Entity-Relationship-Modell entworfen. Das Modell wurde während mehreren Inkrementen laufend angepasst und verbessert.

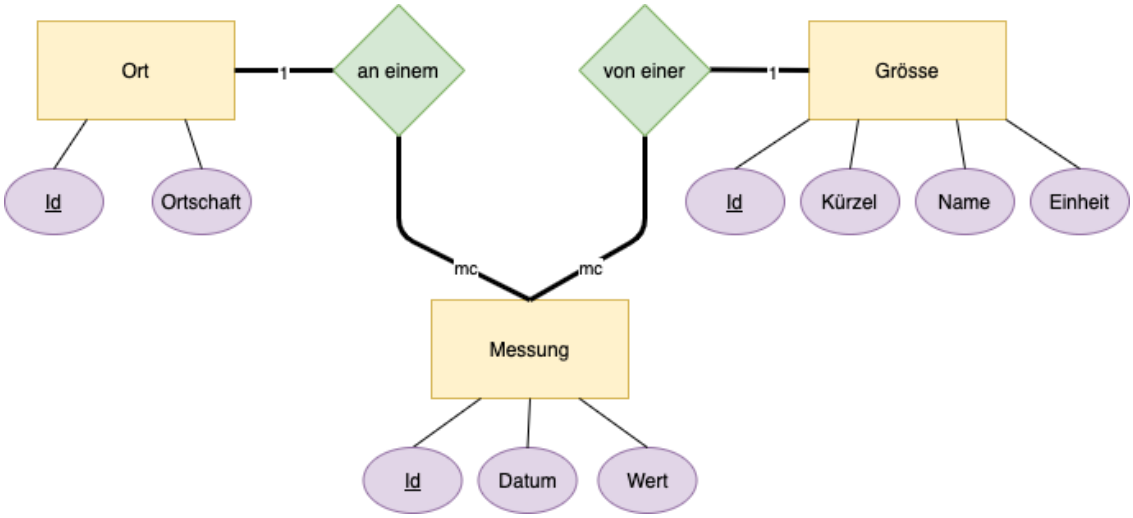

Abbildung 2 Entity-Relationship-Modell

3.2 Umsetzung Datenmodell

3.2.1 Datenbank einrichten

Das Erzeugen der Datenbank ist mit einem einfachem Query möglich und auch Rollen und Benutzer sind schnell erstellt. Da wir uns für einen Webapp für die Visualisierung der Daten entschieden haben, setzen wir für die Abfragen einen Web-User ein, welcher nur minimale Zugriffrechte besitzt. Nachfolgend die nötigen Queries, welche die Datenbank grundlegend bereitstellt.

```
CREATE DATABASE dbs;
USE dbs;
CREATE ROLE 'webuser';
GRANT SELECT ON dbs.* TO 'webuser';
CREATE USER 'php_user'@'localhost' IDENTIFIED BY '******';
GRANT 'webuser' TO 'php_user'@'localhost';
```

SQL-Statement 2 Erstellen der Datenbank, Rolle, User und Berechtigung

3.3 Schema

3.3.1 Tabelle Messung

Die Tabelle Messung besteht aus dem Primary Key, einem Datum und Uhrzeit der Messung und den Wert der entsprechenden Messgrösse. Die Messgrösse sowie der Ort der Messung werden über Referenzen in den Attributen *ort_id* und *groesse_id* festgehalten.

Für die Erzeugung einer ID setzen wir bei jeder Tabelle einen Trigger ein. Der Trigger generiert vor dem Einfügen einen universally unique identifier (UUID) und speichert ihn als Primary Key ab.

Um performantere Abfragen zu ermöglichen, haben wir zusätzlich einen Index auf dem Datumsattribut erstellt. Siehe Abschnitt 4.2.2 für weitere Informationen.

```
CREATE TABLE IF NOT EXISTS messung
(
    id BIGINT NOT NULL,
    datum DATETIME NOT NULL,
    wert DECIMAL(8,2) NULL,
    ort_id BIGINT NULL,
    groesse_id BIGINT NULL,
    CONSTRAINT messung_id_uindex
        UNIQUE (id),
    CONSTRAINT fk_messung_groesse
        FOREIGN KEY (groesse_id) REFERENCES groesse (id),
    CONSTRAINT fk_messung_ort
        FOREIGN KEY (ort_id) REFERENCES ort (id)
);

ALTER TABLE messung
    ADD PRIMARY KEY (id);

CREATE DEFINER = sa_dbs@`%` TRIGGER init_uuid_messung
    BEFORE INSERT
    ON messung
    FOR EACH ROW
    SET NEW.id = UUID_short();

CREATE INDEX idx_messung_datum ON messung (datum);
```

SQL-Statement 3 CREATE Statement für die Tabelle Messung

3.3.2 Tabelle Ort

In der Tabelle Ort werden die Ortschaften gespeichert, wo die Messungen gemacht werden. Sie besteht aus einer ID, sowie der Ortschaftsbezeichnung.

```
CREATE TABLE ort
(
    id BIGINT NOT NULL,
    ortschaft VARCHAR(256) NULL,
    CONSTRAINT ort_id_uindex
        UNIQUE (id)
);

ALTER TABLE ort
    ADD PRIMARY KEY (id);

CREATE DEFINER = sa_dbs@`%` TRIGGER init_uuid_ort
    BEFORE INSERT
    ON ort
    FOR EACH ROW
    SET NEW.id = UUID_short();
```

SQL-Statement 4 CREATE Statement für die Tabelle Ort

3.3.3 Tabelle Grösse

Die Tabelle Grösse enthält alle Messgrössen, die in einer Messung vorkommen können. Die Tabelle besteht aus den Attributen ID, Kürzel, Name und der Einheit, in welcher die Messungen vorgenommen werden.

```
CREATE TABLE groesse
(
    id BIGINT NOT NULL,
    kuerzel VARCHAR(20) NULL,
    name VARCHAR(255) NULL,
    einheit VARCHAR(20) NULL,
    CONSTRAINT groesse_id_uindex
        UNIQUE (id)
);

ALTER TABLE groesse
    ADD PRIMARY KEY (id);

CREATE DEFINER = sa_dbs@`%` TRIGGER init_uuid_groesse
    BEFORE INSERT
    ON groesse
    FOR EACH ROW
    SET NEW.id = UUID_short();
```

SQL-Statement 5 CREATE Statement für die Tabelle Grösse

3.4 Physisches Datenbankschema

Das physische Datenbankschema zeigt die Referenzen unter den Tabellen leicht verständlich auf. Wir erkennen den Index auf dem Datumsattribut in der Tabelle Messung, sowie die Trigger in jeder Tabelle. Die Tabelle Messung enthält fast 28 Millionen Einträge, die anderen beiden hingegen enthalten nur eine Handvoll Elemente.

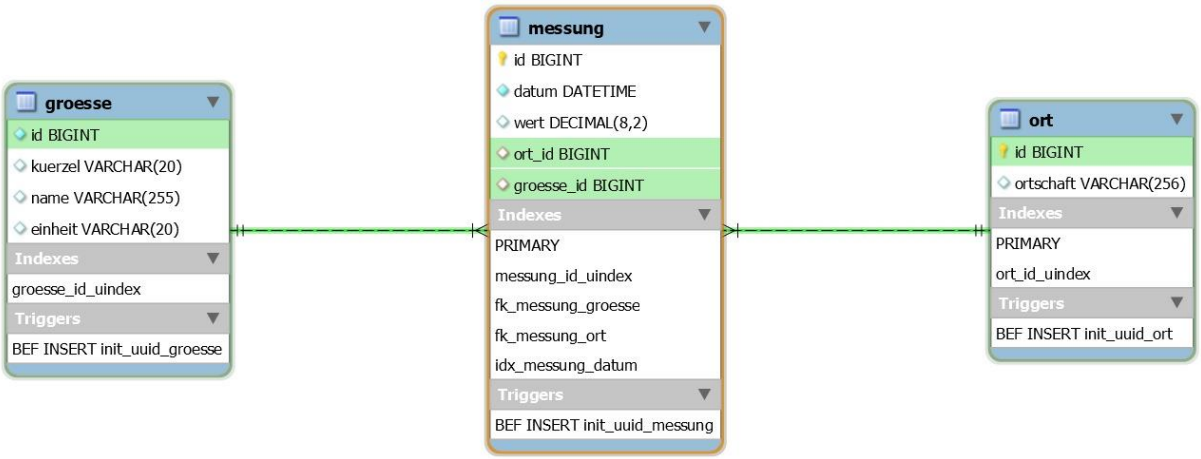

Abbildung 3 Physisches Datenbankschema

4 Datenbanksprachen

4.1 Datenbankabfragen

Datenbankabfragen erfolgen über SQL SELECT Statements. Als Beispiel folgendes Query, welches die ID, den Namen, das Kürzel und die Einheit einer Messgrösse mit der ID «98721425388470291» abgefragt.

```
SELECT id, name, kuerzel, einheit
    FROM groesse
    WHERE id = '98721425388470291';
```

SQL-Statement 6 Beispiel Query für eine Messgrösse

Die Rückgabe des Servers ist:

Id	Name	Kürzel	Einheit
98721425388470291	Feinstaub	Pm10	Ug/m3

Tabelle 1 Resultat des SELECT einer Messgrösse

Um den Standort abzufragen, wird ähnlich wie oben, von der Ort Tabelle mit einer ID selektiert. Hier ein Beispiel mit der ID «98721425388470272».

```
SELECT id, ortschaft
    FROM ort
    WHERE id = '98721425388470272';
```

SQL-Statement 7 Beispiel Query für eine Ortschaft

Der Server antwortet mit folgenden Daten:

Id	Ortschaft
98721425388470272	Bern

Tabelle 2 Resultat des SELECT einer Ortschaft

Komplexer wird das Query wenn Messungen an einem bestimmten Ort, während einer Zeitperiode abgefragt werden. Bei folgender Query wird aus der Tabelle Messung das Datum und der Wert projiziert. Die Ortschaft und die Messgrösse werden aus der Tabelle Ort bzw. Grösse über ihre jeweilige ID vereinigt.

```
SELECT datum, wert, ortschaft, name
    FROM messung m
    JOIN ort o ON (o.id = m.ort_id)
    JOIN groesse g ON (g.id = m.groesse_id)
    WHERE datum BETWEEN '2019-07-25 00:00:00' AND '2019-07-26 00:00:00'
    AND ort_id = '98721425388470272'
    AND groesse_id = '98721425388470291';
```

SQL-Statement 8 Query von Messungen mit INNER JOIN der Tabellen Grösse und Ortschaft

Resultat:

Datum	Ortschaft	Name	Wert
2019-07-25 00:00:00	Bern	Feinstaub	29.40
2019-07-25 01:00:00	Bern	Feinstaub	24.20
2019-07-25 02:00:00	Bern	Feinstaub	20.50
…	…	…	…

Table 1 Query einer Messung mit Joins der Grösse und Ortschaft

4.2 Verbesserungen der Performance

Die Abfrage der Daten war ein stetiger Prozess der Verbesserung. Am Anfang brauchte der Server etwa 7s für einen Monate mit etwa 700 Rows. Um diese Abfrage zu beschleunigen wurden mehrere Anpassungen gemacht.

4.2.1 Select auf IDs statt Namen

Bei der ersten Version wurde bei den Orten nicht auf die Ort ID selektiert, sondern auf den Ortsnamen (Ortschaft). Da auf dem Ortsnamen kein Index gesetzt ist, wurde ein String-Vergleich notwendig, der lange dauerte.

4.2.2 Attribut Datum der Tabelle Messung indexieren

Das Attribut Datum der Tabelle Messung war am Anfang nicht indexiert. Weil es das relevanteste Argument in der `WHERE` Klausel ist, haben wir zur Performancesteigerung nachträglich einen Index auf dem Datum erstellt. Die Indexierung des Datums brachte eine grosse Performanceverbesserung.

```
CREATE INDEX idx_messung_datum ON messung (datum);
```

SQL-Statement 9 Erstellung eines Indizes für das Attribut Datum

4.2.3 Optimierung der Query für eine einzelne Datenbankanfrage

Damit nur eine einzelne Datenbankconnection für eine Abfrage notwendig ist, werden mit folgender Query alle benötigten Daten vereint und als einzelnes Resultset zurückgegeben.

```
SELECT m.datum, kuerzel, einheit, ortschaft, name, wert,
    name2, wert2, einheit2, kuerzel2
    FROM (SELECT datum, wert, ort_id, groesse_id
        FROM messung
        WHERE datum BETWEEN '$DATUM_FROM' AND '$DATUM_TO'
        AND ort_id = '$ORT_ID'
        AND groesse_id = '$MESSWERT1_ID') m
        JOIN (SELECT datum, ort_id, wert AS wert2, g2.name AS name2,
            g2.einheit AS einheit2, g2.kuerzel AS kuerzel2
            FROM messung
            JOIN groesse g2 ON (g2.id = groesse_id)
            WHERE groesse_id = '$MESSWERT2_ID') m2
            ON (m.datum = m2.datum AND m.ort_id = m2.ort_id)
        JOIN ort o ON (o.id = m.ort_id)
        JOIN groesse g ON (g.id = m.groesse_id)
    ORDER BY 1
```

SQL-Statement 10 Query-Optimierung durch Vereinigung

4.2.4 Reduktion vom Resultset

Das Selektieren von Daten über einen längeren Zeitraum bedeutet automatisch auch eine grosse Anzahl Messungen. Einen Monat entspricht bereits über 700 Rows. Um dem entgegenzuwirken wurden die Messpunkte auf etwa 120 Rows reduziert. Damit die Daten aber trotzdem noch richtig sind, wurde ein Durchschnitt und ein Maximalwert genommen. Die Umsetzung mit Query sieht folgendermassen aus:

```
SELECT m.datum, kuerzel, einheit, ortschaft, name,
    ROUND(AVG(wert), 2) AS wert, ROUND(MAX(wert), 2) AS maxwert,
    ROUND(AVG(wert2), 2) AS wert2, ROUND(MAX(wert2), 2) AS maxwert2,
    name2, einheit2, kuerzel2
    FROM (SELECT datum, wert, ort_id, groesse_id
        FROM messung
        WHERE datum between '$DATUM_FROM' AND '$DATUM_TO'
```

```
        AND ort_id = '$ORT_ID'
        AND groesse_id = '$MESSWERT1_ID') m
        JOIN (SELECT datum,ort_id, wert AS wert2, g2.name AS name2,
            g2.einheit AS einheit2, g2.kuerzel AS kuerzel2
            FROM messung
            JOIN groesse g2 ON (g2.id = groesse_id)
            WHERE groesse_id = '$MESSWERT2_ID') m2
            ON (m.datum = m2.datum AND m.ort_id = m2.ort_id)
        JOIN ort o ON (o.id = m.ort_id)
        JOIN groesse g ON (g.id = m.groesse_id)
        GROUP BY UNIX_TIMESTAMP(m.datum) DIV ($numDays*720)
    ORDER BY 1
```

SQL-Statement 11 Query über längeren Zeitraum mit Average- und Max-Value

5 Systemarchitektur

5.1 Produktive Umgebung

Für die produktive Umgebung haben wir im Enterprise Lab einem Server-Host beantragt. Weil wir zu Beginn des Semesters mit einem Microsoft SQL-Server planten, liessen wir uns ein Windows Server 2019 mit 4GB RAM, sowie 50 GB Speicherbedarf bereitstellen. Die Ressource mit OS wurde schnell ausgeliefert und wir installierten einen MS SQL-Server 2019. Gegen Ende des Semesters änderten sich die Modulendprüfungsbestimmungen. Daraufhin haben wir uns entschieden, die Visualisierung mittels WebApp auf Basis von PHP umzusetzen. MS SQL als Datenbank-Managment-System gepaart mit PHP erschien uns unpassend. Deshalb deinstallierten wir den MS SQL-Server komplett und installierten einen MySQL Community Server. Für PHP musste eine zusätzliche Apache Distribution installiert werden. Dazu nutzen wir das Programmpaket von XAMPP.

Nachdem wir erstmals den kompletten Datensatz importiert hatten, merkten wir bald, dass die veranschlagten 4GB RAM viel zu knapp kalkuliert waren. Der Host war überlastet. Wir haben erneut beim Enterprise Lab angeklopft und sie gebeten der VM weitere 8 GB RAM zuzuweisen. Mit den 12 GB RAM ist der Host mit dem Grundrauschen des MySQL-Servers gut ausgelastet (~60%), hat aber dennoch die Kapazität grössere Abfragen effizient zu verarbeiten.

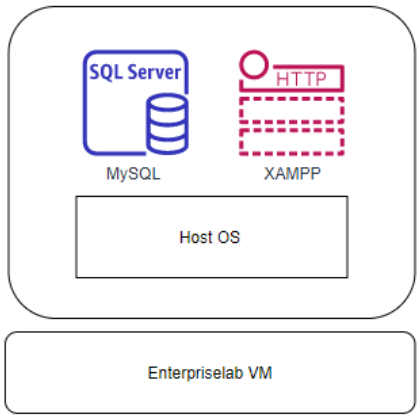

Abbildung 4 Übersichtsschema produktive Umgebung

5.2 Test Umgebung

Während des Projektes hatten wir zwei verschiedene Testumgebungen aufgebaut. Dank der lokalen Testumgebungen konnte jeder von uns unabhängig voneinander am Projekt arbeiten. Die Umgebung kann bei Bedarf auf dem eigenen Rechner aufgebaut und wieder abgebaut werden.

5.2.1 MySQL

Der MySQL Server wird als Docker Container gestartet. Das gesamte Datenbankschema wird beim Erstellen des Containers automatisch geladen. Dem Entwickler stehen verschiedene Datenimports zur Verfügung, welche manuell mit einem Bulk Import geladen werden können. Das Base Image wird direkt vom Dockerhub Repostiory heruntergeladen, die Config (inkl. Dockerfile) ist im Gitlab Repository gespeichert.

Der XAMPP Webserver ist als Applikation pro Benutzer auf dem Host OS installiert, da dies nicht als Docker Container existiert. Die Konfiguration und PHP-Skripte stehen den Entwicklern im Gitlab Repository zur Verfügung.

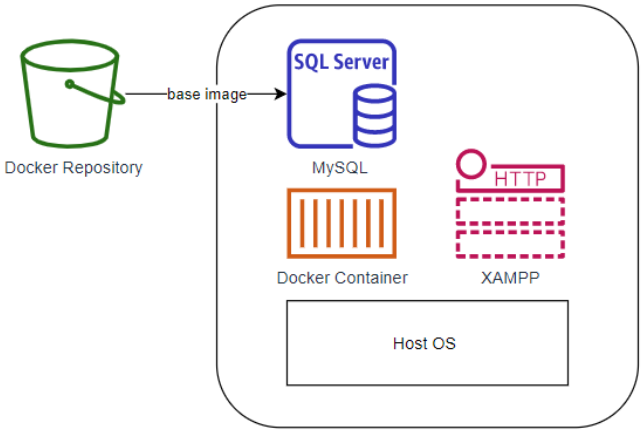

Abbildung 5 Übersichtsschema Testumgebung MySQL

5.2.2 MySQL Cluster

Für die Performance Optimierung haben wir die MySQL Clusterlösung mit NDB als DB Engine getestet. Anstelle mit einzelner Dockerfiles wurde der gesamte Service mit Docker compose orchestriert und wird bei Bedarf hoch- oder wieder heruntergefahren. Diese Testumgebung haben wir als folgenden Gründen schlussendlich nicht in der produktiven Umgebung aufgebaut:

- MySQL Cluster ist ein Hochverfügbarkeitssystem, welches für Grossunternehmen ausgelegt ist.
- Der Grundressourcenbedarf ist sehr hoch und konnte mit unserer produktiven Umgebung nicht gedeckt werden.
- Der initiale Datenimport ist ineffizient, da Datenimports direkt in die NDB Engine sehr lange dauern. Als Alternative haben wir die Tabellen der InnoDB Engine zugeteilt gelassen, den Import durchgeführt und dann mit einem Update auf die NDB Engine gewechselt. Der Import bleibt aber dennoch ineffizient.

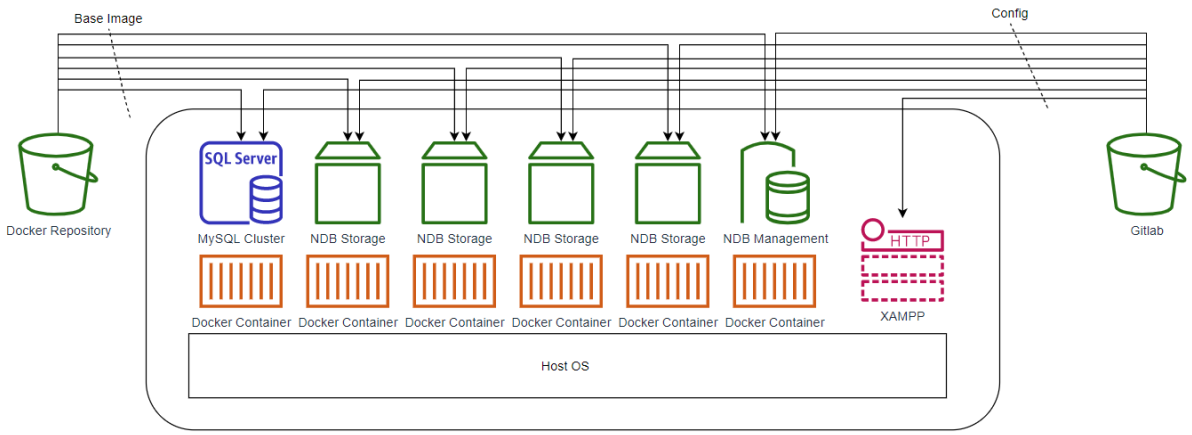

Abbildung 6 Übersichtsschema Testumgebung MySQL Cluster

6 Resultate

Die im Projekt gewonnenen Erkenntnisse und Resultate bestätigen die dem Projekt zu Grunde liegende Frage «Verschmutzen Feuerwerke unsere Luft messbar?».

Aus den Daten haben sich zusätzlich noch weitere Vermutungen bestätigt. Zum Beispiel, dass ein Regenschauer die in der Luft schwebenden Teilchen bindet und somit hilft die Feinstaubbelastung akut zu senken.

Wenn man grössere Zeiträume untersucht, zeigt sich jedoch auch, dass der 1. August, wie auch die Feier an Neujahr mit den Feuerwerken nur wenig zur Feinstaubbelastung über die Zeit beitragen. Es gibt sogar einzelne Tage, an denen die Feinstaubbelastung um den Faktor 8 grösser ist als am 1. August. – Eine Abfrage, welche genau dies zeigt, ist wie folgt möglich:

Messwert 1:	Feinstaub pm10
Messwert 2:	Niederschlag prec
Ort:	Bern
Datum von:	01.01.2017
Darum bis:	10.08.2017

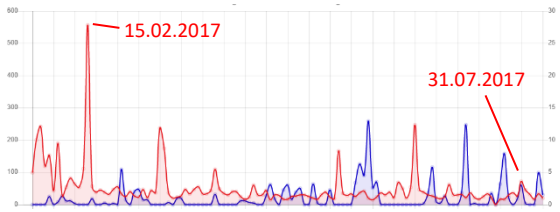

Abbildung 7 Beispiel Auswertung über 8 Monate

6.1 Visualisierung

Durch die Visualisierung haben wir eine Möglichkeit gefunden die Daten so darzustellen, dass sie einfach verständlich in der Form eines Graphen lesbar und interpretierbar sind. Das als WebApp implementierte GUI lässt es zu, eine Zeitspanne von einem Tag bis zu einem Jahr zu selektieren und dabei die entsprechenden Werte zu vergleichen.

Als Beispiel werden nachfolgend Bern und Basel verglichen. Im Zeitraum vom 31.07.2017 bis zum 04.08.2017 wird in Rot die Feinstaubbelastung PM10 und in Blau der gemessene Niederschlag ausgegeben.

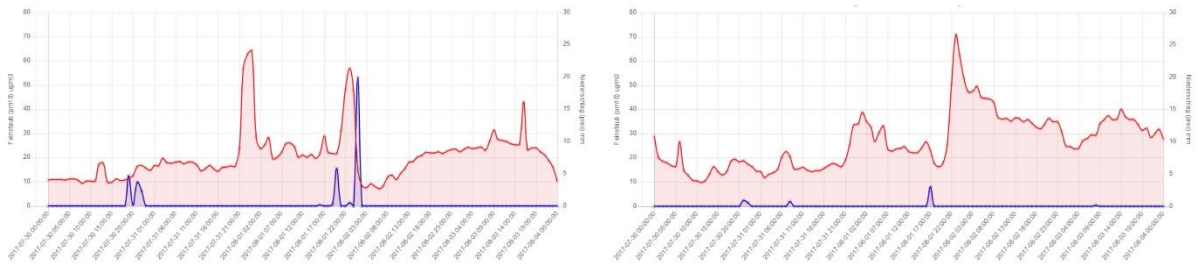

Abbildung 8 Beispiel Auswertung Vergleich Basel und Bern um den 1. August

Im Beispiel ist gut zu erkennen, dass in Basel (links) zweimal gefeiert wird, wohingegen in Bern (rechts) lediglich eine grosse Feier stattfindet. In Basel auch sehr schön zu sehen ist, der bereits erwähnte positive Einfluss von Regen auf die Feinstaubbelastung.

6.1.1 Anzahl Messpunkte

In der aktuellen Version der Visualisierung lassen wir eine Zeitspanne von 380 Tagen zu, um die Möglichkeit zu bieten ein ganzes Jahr auf einmal zu analysieren. Dies bedeutet aber, dass etwas über 9000 Messungen im Zugriff stehen und als Punkte dargestellt würden. Nebst dem, dass der Graph bei so vielen Punkten schon optisch keine sinnvolle Interpretation mehr zulässt, ist mit einer solchen Datenmenge auch der Browser mit Rendering überfordert.

Wir haben mittels manuellen Tests festgestellt, dass Abfragen über 5 Tage rund 120 Messpunkte ergeben und dies einen homogenen, gut lesbaren Graphen ergibt. Davon ausgehend haben wir einen Weg gesucht, die Ergebnisse immer auf eine konstante Anzahl Messpunkte zu reduzieren, falls mehr Messpunkte vorhanden sind.

Die SQL-Funktion DIV hat uns mit wenig Aufwand ermöglicht, dies zu erreichen.

```
GROUP BY UNIX_TIMESTAMP(m.datum) DIV ($numDays*720)
```

SQL-Statement 12 Beschränkung der Messpunkte über längeren Zeitraum mit DIV

Solange weniger als sechs Tage selektiert werden, bleibt der Divider unter dem Wert 3600. Diese 3600 entsprechen Dank UNIX_TIMESTAMP den Sekunden und somit genau einer Stunde.

Etwas vereinfacht ausgedrückt, behält der Divider den Wert «1» solange weniger als 6 Tage im Zugriff sind. Da er linear mit der Anzahl Rows wächst, ergibt dies bei einer Abfrage von 10 Tagen einen Divider von 2. Statt also 240 Rows (10 Tage * 24 Stunden) im Zugriff zu haben, bleiben es weiterhin handliche und übersichtliche 120 Messpunkte.

6.1.2 Durchschnitt / Maximalwerte

Bei einer Selektion von Daten über ein ganzes Jahr, würden mit dem Divider lediglich jede 73. Row berücksichtigt (365*6*120/3600). Eine Interpretation über einen Verlauf wird somit ungenau. Um dennoch eine plausible Aussage zu machen, wird in der Abfrage zusätzlich der Maximal- und der Durchschnittswert ermittelt.

Solange weniger als 6 Tage im Zugriff sind, sind beide Werte identisch und es wird lediglich ein Graph pro Messgrösse dargestellt. Ab 6 Tagen werden zusätzlich pro Messgrösse ein Graph der Durchschnittswerte eingebettet und man hat die Möglichkeit diese durch klicken auf die Legende zu aktivieren. Die Durchschnittswerte werden initial ausgeblendet, damit die Leserlichkeit gewahrt bleibt.

7 Diskussion

Aus den Resultaten geht hervor, dass die Feuerwerke einen klaren Effekt auf die Feinstaubbelastung haben, dieser jedoch nur einen kleinen Teil zur Gesamtbelastung beitragen. Trotzdem wäre es falsch diese Erkenntnis als Freipass anzusehen, immer mehr und grössere Feuerwerke zu veranstalten.

Vielmehr sollte man das Gesamtbild aller Belastungen über eine lange Zeit betrachten oder sogar aktiv überwachen, um den Belastungstrend fortlaufend aufzuzeigen. Auf diese Weise könnte man auch Massnahmen definieren und ergreifen, um einem steigenden Trend entgegenzuwirken.

Eine Analyse der Daten vom Januar 2020 bis Mitte Jahr wird sicherlich auch sehr interessante Werte liefern. Wir erwarten, dass durch das stark gehemmte Reiseaufkommen, die Auswertung diverser Luftschadstoffe einen sehr positiven Trend aufzeigen werden.

8 Lessons Learned

Skripte sind ein Business Case, der Default ist kein Business-Case

Eines der Highlights ist das skriptbasierte herunterladen und aufbereiten der Daten. Damit konnten wir die Rohdaten ideal auf unser Datenbankmodell vorbereiten und aufwändige manuelle Tätigkeiten konnten vermieden werden. Weiter wurden durch die vielen Datensätze Optimierungen am SQL Server notwendig. Die Vanilla-Konfiguration des MySQL-Servers reicht dafür bei weitem nicht aus. Dazu mussten wir uns detailliert in die Manuals einlesen. Mit diesem zusätzlichen Wissen verstehen wir die Funktionsweise eines Datenbank-Managements-Systems erheblich besser.

Mit den umfangreichen Daten erfuhren wir, wie Query-Optimierungen die Performanz effektiv beeinflussen können. Auch diese Erkenntnis trägt viel für unser Verständnis in der Formulierung der SQL Abfragen bei.

Menge der Daten

Die Datenmenge hat uns überrascht. Anfangs haben wir mit 2 Millionen Datenreihen gerechnet. Dabei haben wir die Normalisierung nicht miteinberechnet. Nach der Normalisierung ist die Anzahl Datenreihen auf 28 Millionen gestiegen. Dabei haben wir gelernt, dass erst die Datengrundlage genau bekannt sein muss, bevor die Dimensionierung oder die Wahl der Datenbanktechnologie vorgenommen werden kann.

Mikrooptimierungen in der Performance

Wir haben mit der Indexierung und Query Optimierung eine gute Performance erreicht. Trotzdem haben wir aus Ehrgeiz die Datenbank weiter zu optimieren versucht und haben Parameter an der InnoDB verändert. Auch haben wir einen MySQL Cluster mit der NDB Engine aufgebaut und getestet. Damit haben wir Mikromanagement betrieben, welches nicht nötig gewesen wäre und viel Zeit in Anspruch nahm. Gleichwohl hat sich der Aufwand gelohnt, weil wir sehr viel über die DB Engines gelernt haben.

UUID in MySQL

Eine Universally Unique Identifier kurz «UUID» ist eine zuverlässige und sichere Art eindeutige Schlüssel zu verwenden. Aus diesem Grund haben wir uns zu dieser Schlüsselmethode entschieden. Beim Erstellen der Tabellen mussten wir aber feststellen, dass MySQL zwar eine Methode UUID() zum Generieren einer UUID bereitstellt (via `SELECT`), diese aber nicht als Defaultwert in einer Kolonne verwendet werden kann.

Um trotzdem eine UUID verwenden zu können entschlossen wir, dafür das im Unterricht vorgestellte Konzept eines Triggers, zum Generieren einer UUID zu verwenden. Vor jedem `INSERT` in eine Tabelle wird der Trigger «init_uuid_*» aufgerufen und erzeugt eine UUID und setzt diese in die Spalte id.

Datenqualität der Rohdaten

Daten sind überall im Netz verfügbar und können genutzt werden. Für einen Import in eine Datenbank sind jedoch oft Anpassungen an den Rohdaten nötig. Durch manuelle Änderungen können die Daten fehlerhaft und somit unbrauchbar werden. In unserem Beispiel musste das Datumsformat von "DD.MM.YYYY hh:mm" auf "YYYY-MM-DD hh:mm:ss" angepasst werden und auch zusätzliche Spalten angefügt werden. Gute Qualität der Rohdaten sollte nicht unterschätzt werden und somit auch der Aufwand, die Qualität zu verbessern.

Wir als Gruppe empfehlen mit einem kleineren Datenset auf einer lokalen DB zu starten. Die Validierung der Daten geht schnell und Fehler können einfacher und effizienter korrigiert werden.

Das richtige Projekt

Wir als Team haben uns ziemlich schwer getan uns auf ein Thema zu einigen. Es brauchte mehrere Iterationen und doch einiges an Zeit und Recherche, bis wir überhaupt mit dem Technischen Teil fürs Projekt starten konnten. Unser Learning dabei ist zum einen, dass es sehr viel einfacher ist mit einem fertigen Use Case zu entwickeln und zum anderen, dass es durchaus aufwändig sein kann, eben genau diesen Use Case soweit zu bekommen, bis man tatsächlich etwas entwickeln kann.

Für künftige Absolventen bieten sich vor allem die von uns beschriebenen Problemstellungen an, um nicht dieselben Fehler zu machen und vielleicht die eine oder Andere Hürde überspringen zu können.

Social Distance

Das Arbeiten in der Gruppe und in Projekten wie diesem ist eine der dankbarsten Lernmethoden. Durch das unterschiedliche Wissen der Teilnehmer, können sich diese gegenseitig sehr viel erklären und so kann man im Rahmen vom Projekt, technisch wie auch zwischenmenschlich viel profitieren.

Auf einer Meta-Ebene waren in diesem Semester Studenten wie Dozenten zusätzlich gefordert. Die wegen der Corona Pandemie eingeführten Massnahmen haben einen weiteren unbekannten Faktor in die Gleichung gebracht und es hat seine Zeit gebraucht, bis wir alle uns an die neue Situation gewöhnt haben. Der Austausch hat sich stark erschwert und die nonverbale Kommunikation ist fast gänzlich weggefallen. Wir haben dabei gelernt, dass ein höheres Mass an Disziplin und Organisation unumgänglich sind, um trotzdem sauber zusammen arbeiten zu können.

Niemals Notepad für my.ini Manipulation verwenden

In der my.ini Datei haben wir die RAM Kapazität für die MySQL Datenbank angepasst. Um die Anpassung zu aktivieren muss der MySQL Service neugestartet werden. Dieser konnte sich jedoch nicht mehr vollständig starten. Grund dafür war der verwendete Editor Notepad. Notepad arbeitet im Encoding mit UTF-8 BOM und fügt der Datei die drei nicht sichtbaren HEX Charakter "EF BB BF" hinzu. Diese entsprechen einer der Byte Order Mark (BOM) und damit wurde das Starten des MySQL Service verhindert. Nachdem wir die Datei mit Notepad++ als neue Datei gespeichert haben und die alte Datei gelöscht wurde, konnte der MySQL Service wieder gestartet werden.

Nach Manipulation mit Notepad	Nach Manipulation mit Notepad++
Powershell Kommando für Datei im Hex Format Format-Hex -Path my.ini | select -first 10	Powershell Kommando für Datei im Hex Format Format-Hex -Path my.ini | select -first 10
EF BB BF 23 20 4F 74 68 65 72 20 64 65 66 61 75 ï»¿# Other defau 6C 74 20 74 75 6E 69 6E 67 20 76 61 6C 75 65 73 lt tuning values 0D 0A 23 20 4D 79 53 51 4C 20 53 65 72 76 65 72 ..# MySQL Server 20 49 6E 73 74 61 6E 63 65 20 43 6F 6E 66 69 67 Instance Config 75 72 61 74 69 6F 6E 20 46 69 6C 65 0D 0A 23 20 uration File..# 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 2D 2D 2D 0D 0A 23 20 47 65 6E 65 72 61 ------..# Genera	23 20 4F 74 68 65 72 20 64 65 66 61 75 6C 74 20 # Other default 74 75 6E 69 6E 67 20 76 61 6C 75 65 73 0D 0A 23 tuning values..# 20 4D 79 53 51 4C 20 53 65 72 76 65 72 20 49 6E MySQL Server In 73 74 61 6E 63 65 20 43 6F 6E 66 69 67 75 72 61 stance Configura 74 69 6F 6E 20 46 69 6C 65 0D 0A 23 20 2D 2D 2D tion File..# --- 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D ---------- 2D 2D 2D 0D 0A 23 20 47 65 6E 65 72 61 74 65 64 ---..# Generated

Tabelle 3 my.ini mit (links) und ohne BOM (rechts)

9 Anhang

9.1 Interessante Datenreihen

Nachfolgend sind einige interessante Datenreihen aufgeführt, die Lesende dazu motivieren soll auch selber nach interessanten Messwerten zu suchen.

Das GUI (Webinterface) ist im HSLU-Netz oder via VPN unter https://dbs-f20-tbstofer.el.eee.intern/ erreichbar.

Messung 1	Messung 2	Ort	Von	Bis	Beschreibung
Pm10	-	Bern	15.01.2017	20.02.2017	Sehr hohe Belastungswerte verglich für Feinstaub verglichen mit dem 1.8.
Pm10	Temp	Alle	30.07.2018	04.08.2018	Feuerwerkverbot durch Jahrhunderthitze
Pm10	Prec	Dübendorf	30.07.11-15	04.08.11-15	Abnehmender Belastungstrend

9.2 Datenimport Skripts

Die Importskripte für den initialen Datenimport, sowie für den täglichen Update haben wir in einem Github-Repository zur Verfügung gestellt.

https://github.com/JulienGrueter/fs20-dbs/blob/master/project/bulk_load.sh

https://github.com/JulienGrueter/fs20-dbs/blob/master/project/daily_import.sh