SQL Lab

245362, Isak Skeie

Table of Contents

[1 Databa logging specification 3](#_Toc101282280)

[2 Database Structure 3](#_Toc101282281)

[3 Database Server 4](#_Toc101282282)

[4 Data Generation 7](#_Toc101282283)

[5 Data monitoring 8](#_Toc101282284)

[6 Conclusion 10](#_Toc101282285)

[7 Appendix 10](#_Toc101282286)

# Databa logging specification

The assignment asks for a temperature logging system to be created. The temperature readings are originating from LabView, where it is sent to a SQL database where the data is processed and stored. The stored data is then going to be reported with a C# application, where an overview of the data is made. When creating the SQL database, scalability, autonomy and general overview has to be kept in mind when creating the structure of the database. With data creating uploaded to the database, as well as data fetching, central skills surrounding SQL is used. For this

# Database Structure

Before the structure for the database is created. Its use case needs to be investigated, to highlight the need for scalability and general overview. The dataset for the rooms are fixed, the amount of rooms wont change, this table is named LOCATION. The temperature sensors are placed in the rooms listed in LOCATION, with several sensors one or none being in a room, the table is names SENSORS. The logging of data is made in the tables LOG for the sensors, and LOCATION\_LOG for each of the rooms. In these tables, rows consists of data identified with a timestamp and a foreign key coming from either SENSOR or LOCATION.

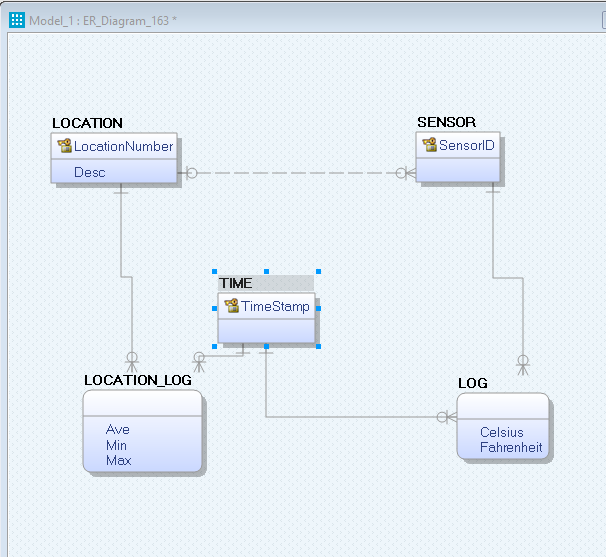


Figure 1: Logical Structure of the database

With the Physical layer created in Erwin Data Modeler. The Primary Keys combined with the foreign keys shows how all the tables are connected and which boundaries this gives. As with he LOG Table, with the SensorID from SENSOR being used as a primary key, its not possible to log a sensor reading with a sensor that’s not listed, the same applies to the LOCATION\_LOG and LOCATION. The same applies to the timestamps. Data can only be logged with the timestamps provided from TIME

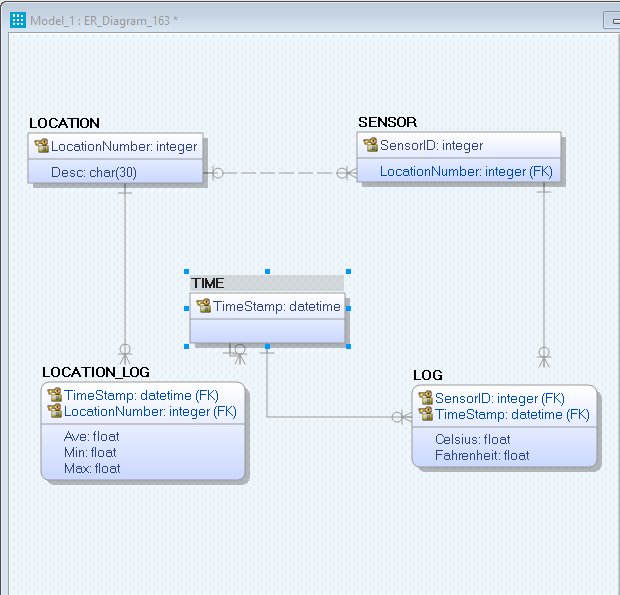


Figure 2: Physical Structure of the database

# Database Server

From Erwin Database Modeler. A SQL script is created based on the Logical and Physical structure created. As seen in the appendix. In SQL Management Studio a Table Diagram is created, as seen in Figure 3. This shows the same structure as seen in the Erwin Data Modeled structures. When sensor readings are being inserted into the database, two triggers are put in place, that handles the incoming data. The first one is connected to LOG and replaces the Insert statement. It makes the conversion from Celsius to Fahrenheit before the insert is performed. The last trigger executes on the LOG table after an INSERT. This generates data for the LOCATION\_LOG table, and inserts it into the LOCATION\_LOG table. Lastly a Storage Procedure is created, for sensor readings to be inserted. The Sensor ID and its value is inputted as parameters when executing the procedure.

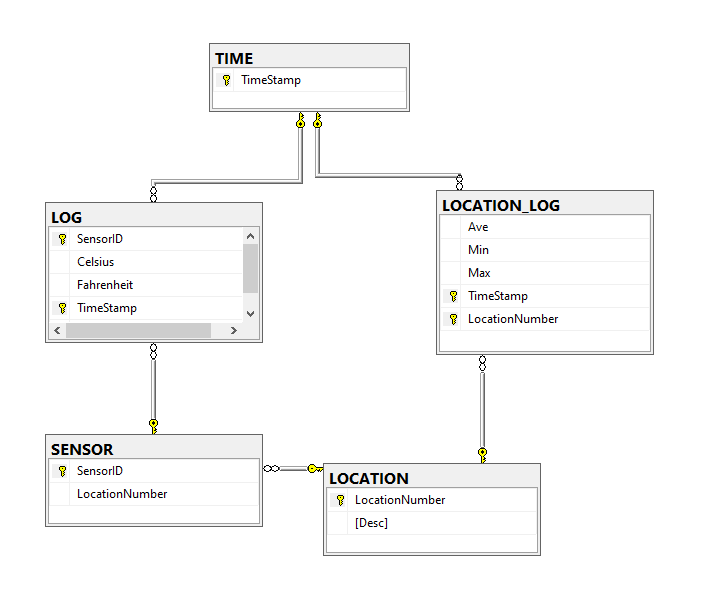


Figure 3: Table diagram after Database creation

Graphical user interface, text, application, email

Description automatically generated

Figure 4: Trigger that converts Celsius to Fahrenheit

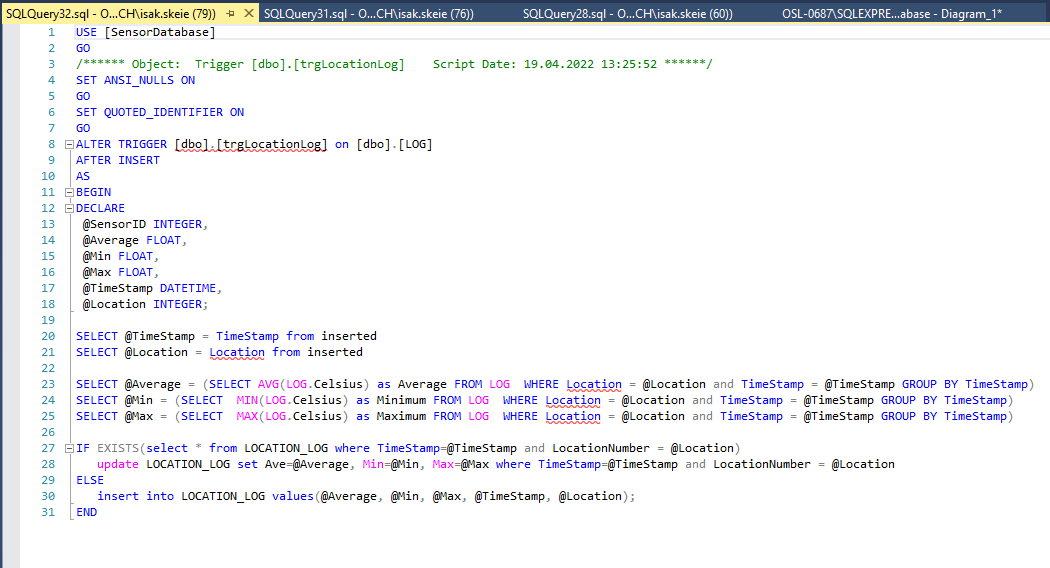


Figure 5: Trigger that creates and stores statistics for each room

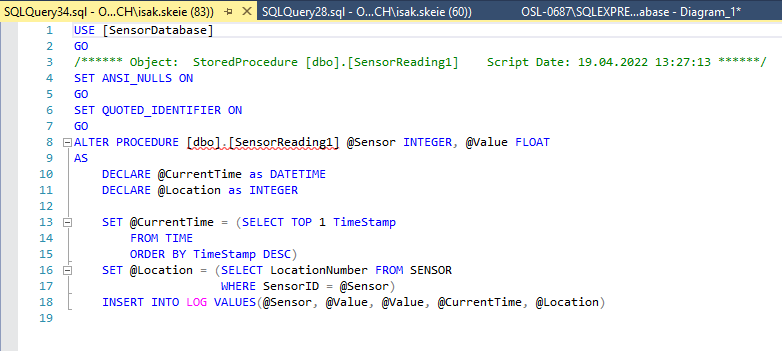


Figure 6: Storage procedure for sensor readings

# Data Generation

Without access to the physical lab room. Temperature readings are generated randomly and inserted to the LOG Table with a Storage Procedure. This is done in a FOR-Loop that iterates through the sensors, creating one random sample for each of them. The FOR-Loop is surrounded by a while loop, making sure the program samples and inserts continuously into the database. Inside the While-Loop, before the For-Loop an Insert query is executed, creating a timestamp for the samples to come. The Block Diagram for the LabView Program can be viewed in the Figure 7

Diagram

Description automatically generated with medium confidence

Figure 7: LabView Block Diagram for simulated temperature readings, uploaded to SQL

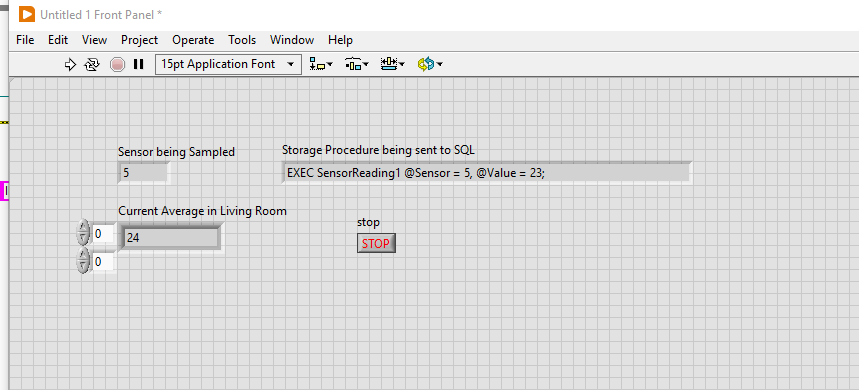


Figure 8: User Inteface for LabView Data and queries.

# Data monitoring

The stored data is displayed in a web application based on the Blazor Server Side Framework. With one page for the Sensor readings as seen in Figure 9 and another page for the Room statistics, as seen in Figure 10. Sorting capabilities are demonstrated with date and number of rows shown on the page. Additionally, export functionality could be added for further analysis of the data.

Table

Description automatically generated

Figure 9: Table that displays temperature readings from the sensors

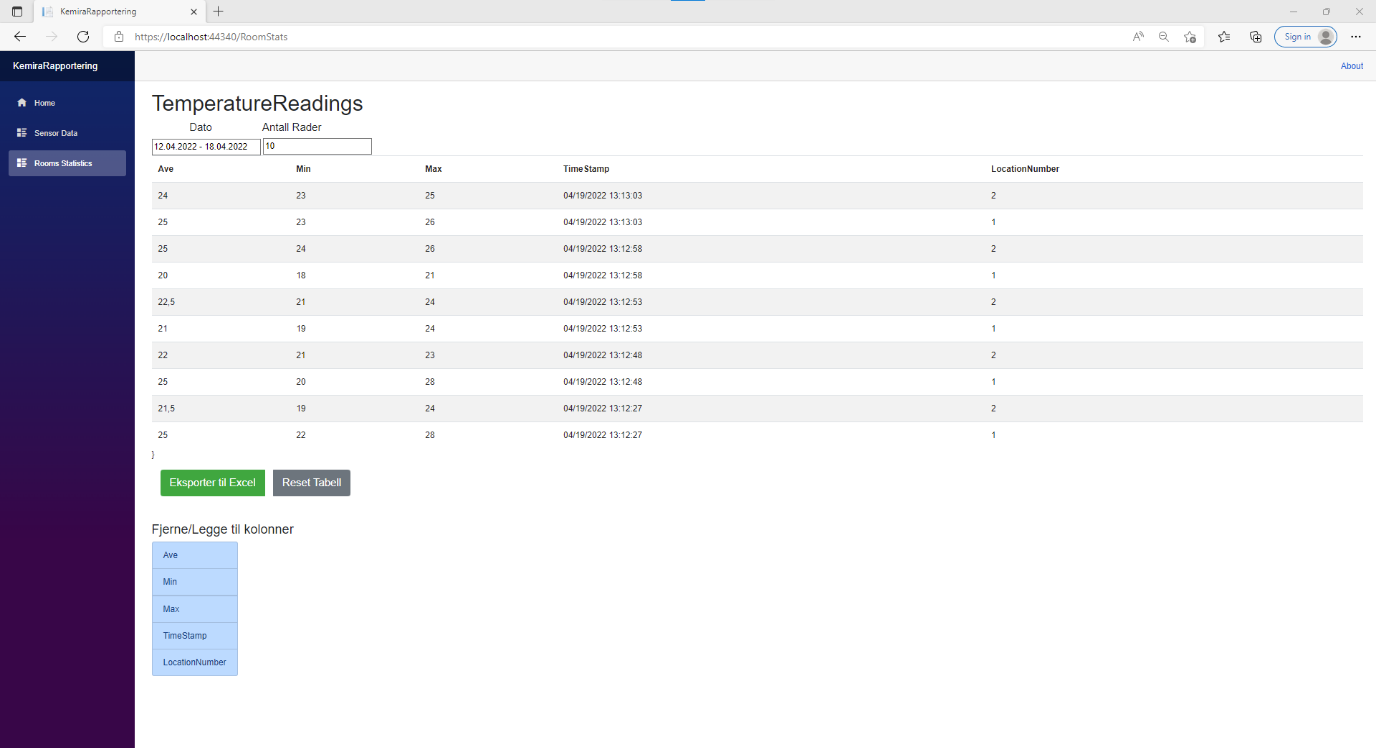


Figure 10: Table that displays Room Statistics for each room and Time Stamp.

# Conclusion

The database and its surrounding parts work as intended. The database is highly scalable as intended. The database is solved in such a way that makes it easy to deploy to a real world scenario. With the code for the SQL database created by Erwin Data Modeler, its easy to reuse for a real use case. The same applies to the data monitoring application. The application can be published for deployment cross platform. With the appsettings, containing SQL server details easily edited before server launch.

The LabView program was made purely to generate data for the DataBase, it could be made more user friendly and with extended functionality. For a deployed use case of this database, LabView would likely be omitted, and it had therefore no need of being elaborated . The reporting application however, could be extended. With charts showing the trends of the measurements and statistics.

By Successfully going through this lab , creating a well structured SQL database . Key aspects of SQL is used and learned. Important skills to have as an Industrial IT engineer.

# Appendix

[1]. Code for SQL database creation:

|  |
| --- |
| CREATE TABLE [LOCATION]  (  [LocationNumber] integer NOT NULL ,  [Desc] char(30) NULL ,  CONSTRAINT [XPKLOCATION] PRIMARY KEY CLUSTERED ([LocationNumber] ASC)  )  go  CREATE TABLE [SENSOR]  (  [SensorID] integer NOT NULL ,  [LocationNumber] integer NULL ,  CONSTRAINT [XPKSENSOR] PRIMARY KEY CLUSTERED ([SensorID] ASC),  CONSTRAINT [R\_1] FOREIGN KEY ([LocationNumber]) REFERENCES [LOCATION]([LocationNumber])  ON DELETE NO ACTION  ON UPDATE NO ACTION  )  go  CREATE TABLE [TIME]  (  [TimeStamp] datetime NOT NULL ,  CONSTRAINT [XPKTIME] PRIMARY KEY CLUSTERED ([TimeStamp] ASC)  )  go  CREATE TABLE [LOG]  (  [SensorID] integer NOT NULL ,  [Celsius] float NULL ,  [Fahrenheit] float NULL ,  [TimeStamp] datetime NOT NULL ,  CONSTRAINT [XPKLOG] PRIMARY KEY CLUSTERED ([SensorID] ASC,[TimeStamp] ASC),  CONSTRAINT [R\_2] FOREIGN KEY ([SensorID]) REFERENCES [SENSOR]([SensorID])  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT [R\_3] FOREIGN KEY ([TimeStamp]) REFERENCES [TIME]([TimeStamp])  ON DELETE NO ACTION  ON UPDATE NO ACTION  )  go  CREATE TABLE [LOCATION\_LOG]  (  [Ave] float NULL ,  [Min] float NULL ,  [Max] float NULL ,  [TimeStamp] datetime NOT NULL ,  [LocationNumber] integer NOT NULL ,  CONSTRAINT [XPKLOCATION\_LOG] PRIMARY KEY CLUSTERED ([TimeStamp] ASC,[LocationNumber] ASC),  CONSTRAINT [R\_4] FOREIGN KEY ([TimeStamp]) REFERENCES [TIME]([TimeStamp])  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT [R\_5] FOREIGN KEY ([LocationNumber]) REFERENCES [LOCATION]([LocationNumber])  ON DELETE NO ACTION  ON UPDATE NO ACTION  )  go |

[2]. BackBone for WebApplication: [Intro to Blazor Server Side - Includes SQL Data Access and Best Practices - YouTube](https://www.youtube.com/watch?v=8DNgdphLvag&t=3544s)

[3]. Code for Web Application: [IsakSkeie/KemiraRapport (github.com)](https://github.com/IsakSkeie/KemiraRapport)