SQL Lab

245362, Isak Skeie

Table of Contents

1 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

2 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.

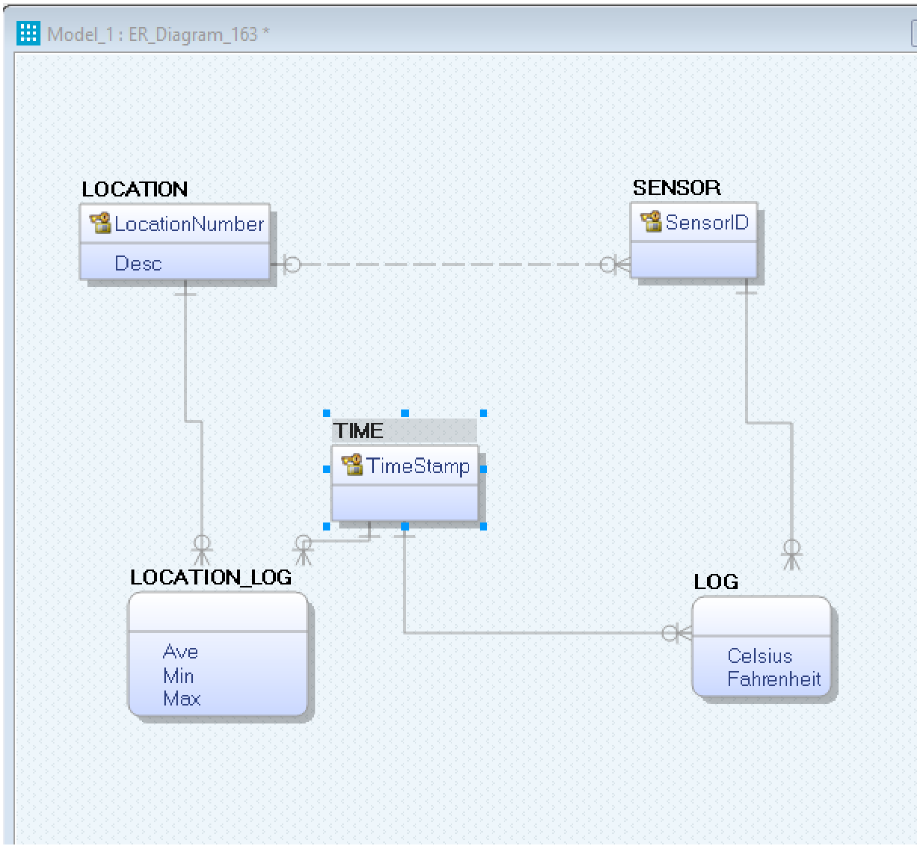

Figure 1: Logical Structure of the database

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

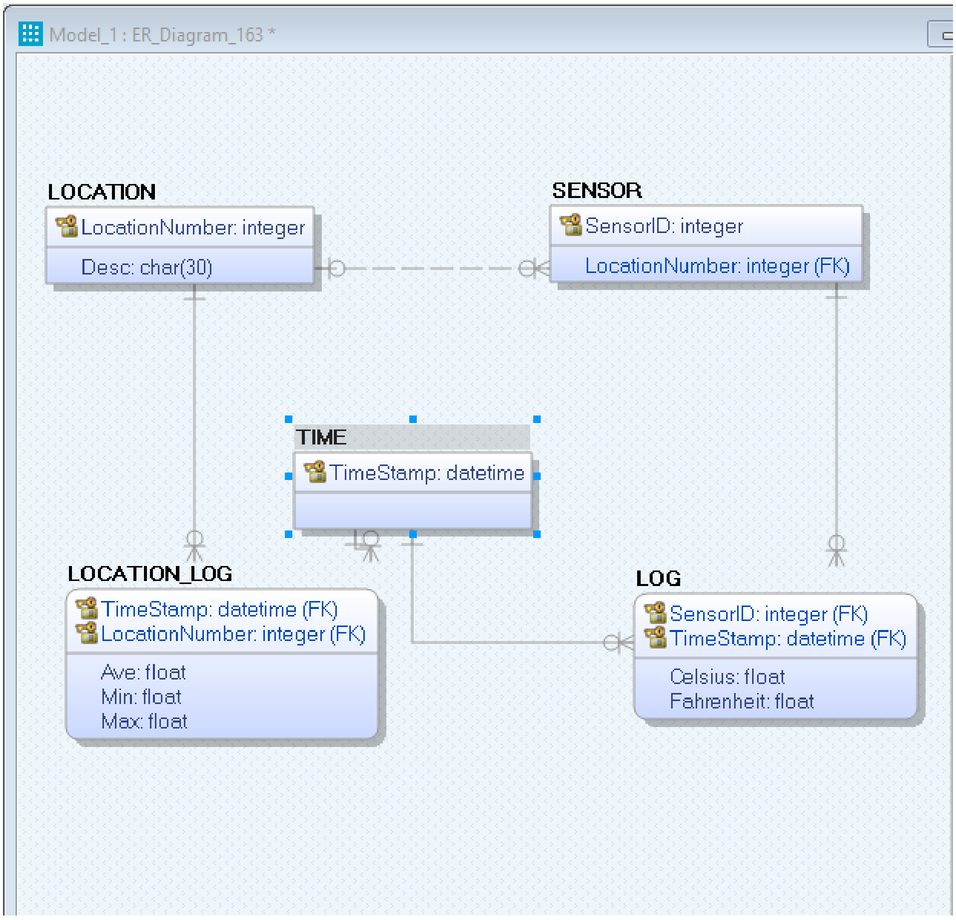

Figure 2: Physical Structure of the database

3 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.

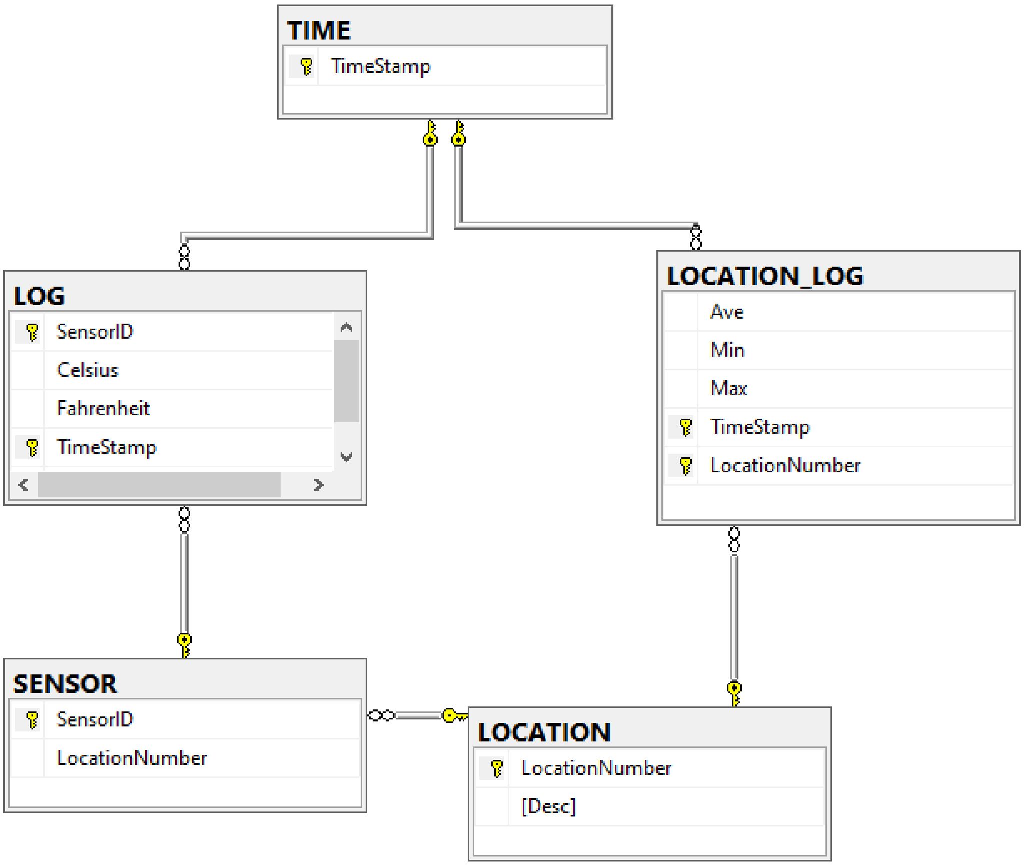

Figure 3: Table diagram after Database creation

SQLQuery33.sql - O...CH\isak.skeie (82)) SQLQuery32.sql - O...CH\isak.skeie (79)) SQLQuery31.sql - O...CH\isak.skeie (76))

```sql
USE [SensorDatabase]
GO
/****** Object:  Trigger [dbo].[trgCelsiusToFahrenheit]    Script Date: 19.04.2022 13:25:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER   [dbo].[trgCelsiusToFahrenheit] on [dbo].[LOG]
INSTEAD OF INSERT
AS DECLARE
    @SensorID INTEGER,
    @Celsius FLOAT,
    @Fahrenheit FLOAT,
    @TimeStamp DATETIME,
    @Location as INTEGER;

SELECT @SensorID = SensorId from inserted
SELECT @Celsius = Celsius from inserted
SELECT @Fahrenheit = 32 + @Celsius*1.8
SELECT @TimeStamp = TimeStamp from inserted
SELECT @Location = Location from inserted

INSERT INTO LOG
    VALUES (@SensorID, @Celsius, @Fahrenheit, @TimeStamp, @Location)
```

Figure 4: Trigger that converts Celsius to Fahrenheit

SQLQuery32.sql - O...CH\isak.skeie (79)) SQLQuery31.sql - O...CH\isak.skeie (76)) SQLQuery28.sql - O...CH\isak.skeie (60)) OSL-0687\SQLEXPRE...abase - Diagram_1*

```sql
USE [SensorDatabase]
GO
/****** Object:  Trigger [dbo].[trgLocationLog]    Script Date: 19.04.2022 13:25:52 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER TRIGGER [dbo].[trgLocationLog] on [dbo].[LOG]
AFTER INSERT
AS
BEGIN
DECLARE
 @SensorID INTEGER,
 @Average FLOAT,
 @Min FLOAT,
 @Max FLOAT,
 @TimeStamp DATETIME,
 @Location INTEGER;

SELECT @TimeStamp = TimeStamp from inserted
SELECT @Location = Location from inserted

SELECT @Average = (SELECT AVG(LOG.Celsius) as Average FROM LOG  WHERE Location = @Location and TimeStamp = @TimeStamp GROUP BY TimeStamp)
SELECT @Min = (SELECT  MIN(LOG.Celsius) as Minimum FROM LOG  WHERE Location = @Location and TimeStamp = @TimeStamp GROUP BY TimeStamp)
SELECT @Max = (SELECT  MAX(LOG.Celsius) as Maximum FROM LOG  WHERE Location = @Location and TimeStamp = @TimeStamp GROUP BY TimeStamp)

IF EXISTS(select * from LOCATION_LOG where TimeStamp=@TimeStamp and LocationNumber = @Location)
   update LOCATION_LOG set Ave=@Average, Min=@Min, Max=@Max where TimeStamp=@TimeStamp and LocationNumber = @Location
ELSE
   insert into LOCATION_LOG values(@Average, @Min, @Max, @TimeStamp, @Location);
END
```

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

SQLQuery34.sql - O...CH\isak.skeie (83)) SQLQuery28.sql - O...CH\isak.skeie (60)) OSL-0687\SQLEXPRE...abase - Diagram_1*

```sql
USE [SensorDatabase]
GO
/****** Object:  StoredProcedure [dbo].[SensorReading1]    Script Date: 19.04.2022 13:27:13 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[SensorReading1] @Sensor INTEGER, @Value FLOAT
AS
    DECLARE @CurrentTime as DATETIME
    DECLARE @Location as INTEGER

    SET @CurrentTime = (SELECT TOP 1 TimeStamp
        FROM TIME
        ORDER BY TimeStamp DESC)
    SET @Location = (SELECT LocationNumber FROM SENSOR
                WHERE SensorID = @Sensor)
    INSERT INTO LOG VALUES(@Sensor, @Value, @Value, @CurrentTime, @Location)

```

Figure 6: Storage procedure for sensor readings

4 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

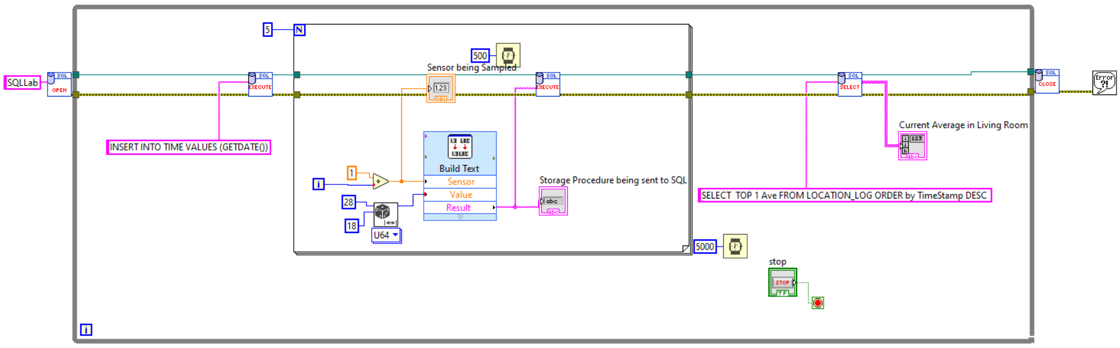

Figure 7: LabView Block Diagram for simulated temperature readings, uploaded to SQL

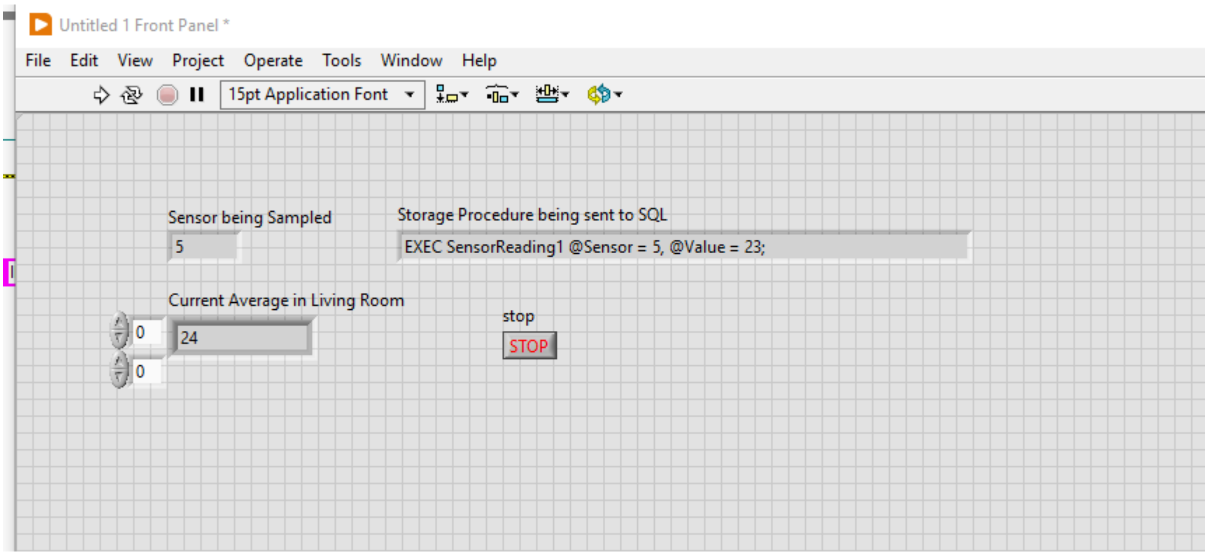

Figure 8: User Inteface for LabView Data and queries.

5 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.

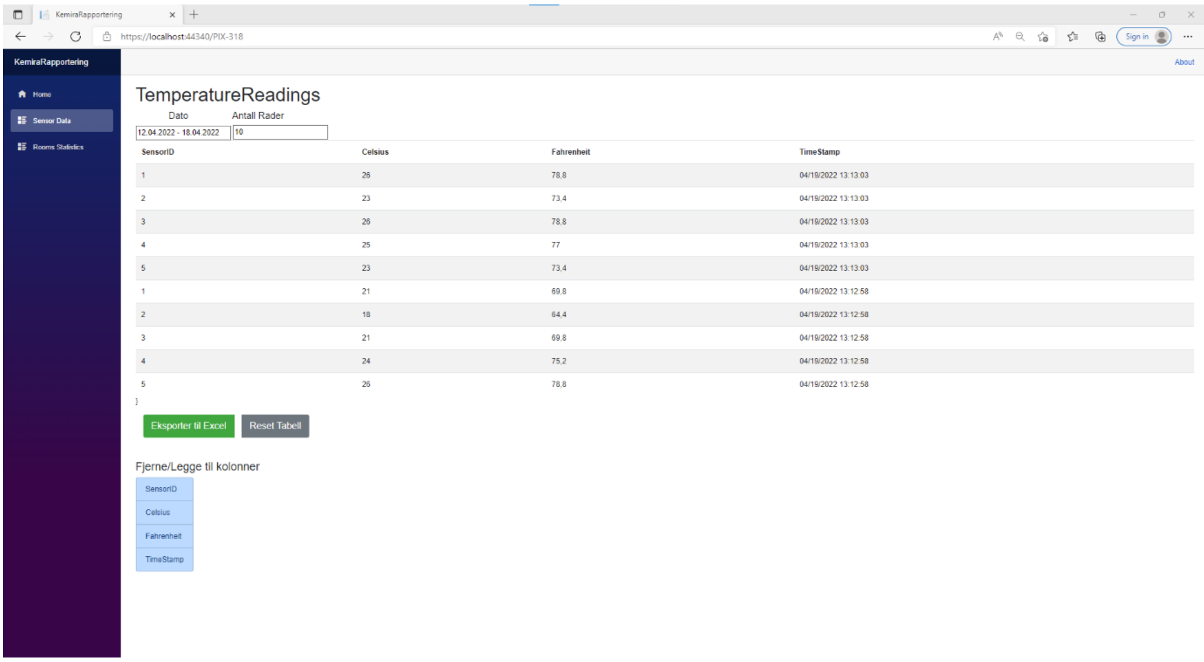

Figure 9: Table that displays temperature readings from the sensors

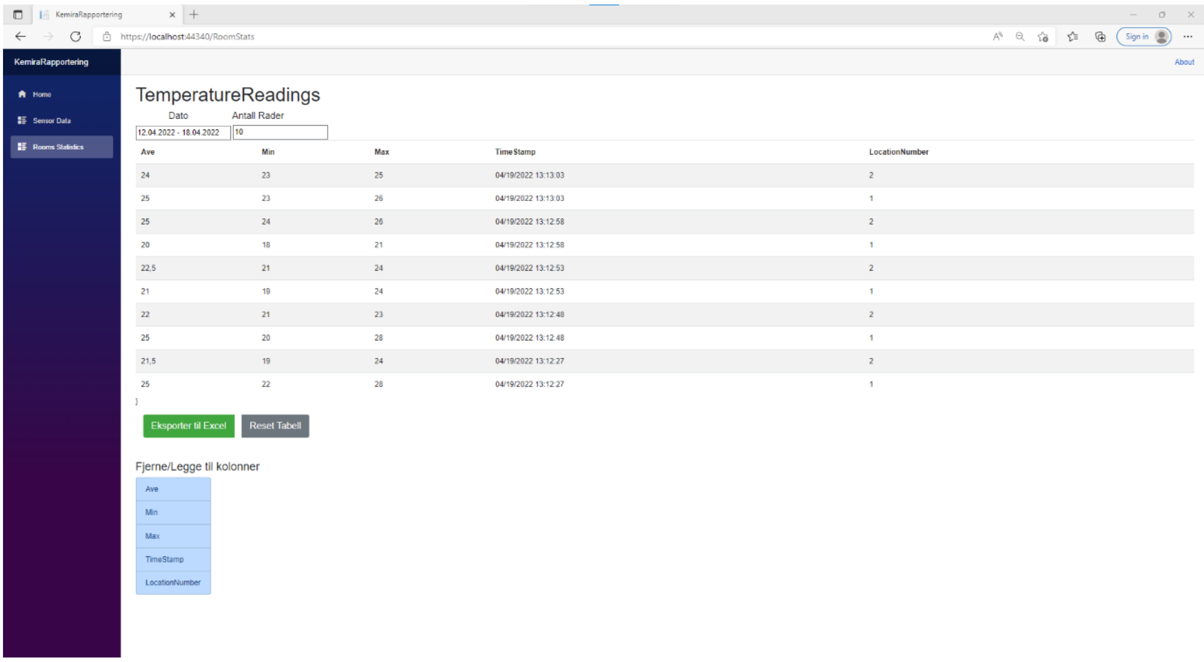

Figure 10: Table that displays Room Statistics for each room and Time Stamp.

6 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.

7 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

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