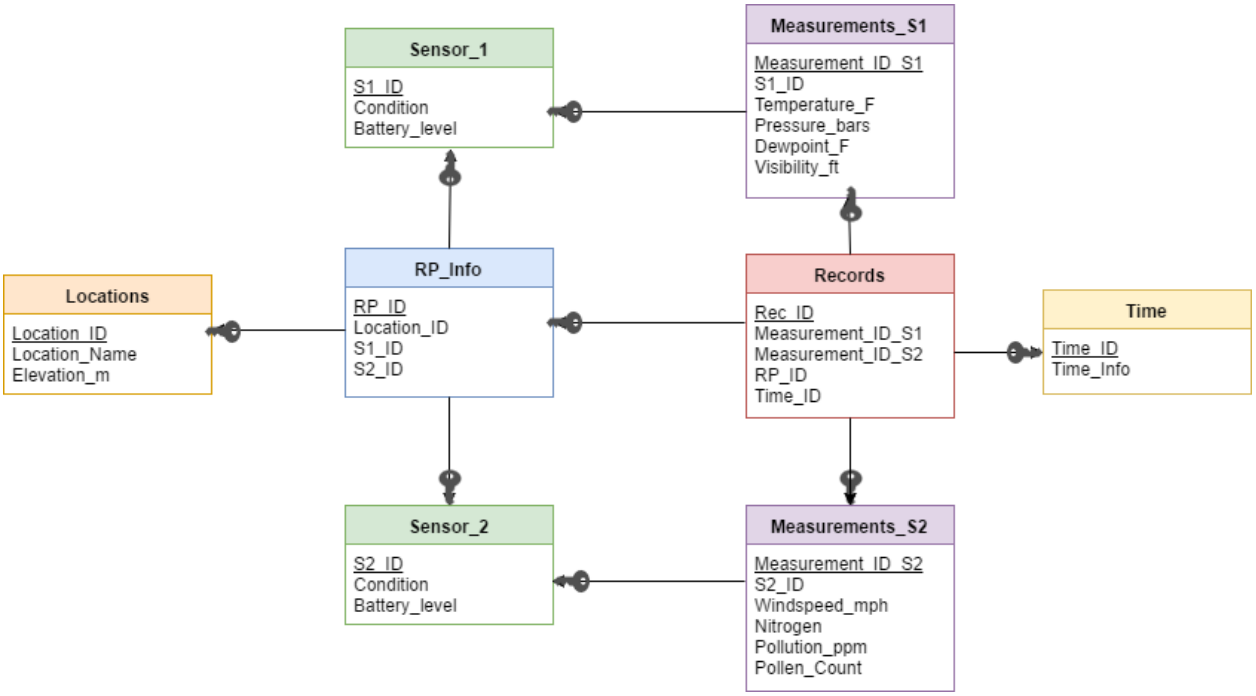
2017

SMURF Database

Contents

PART I: Design a Schema

Schema



Rationale

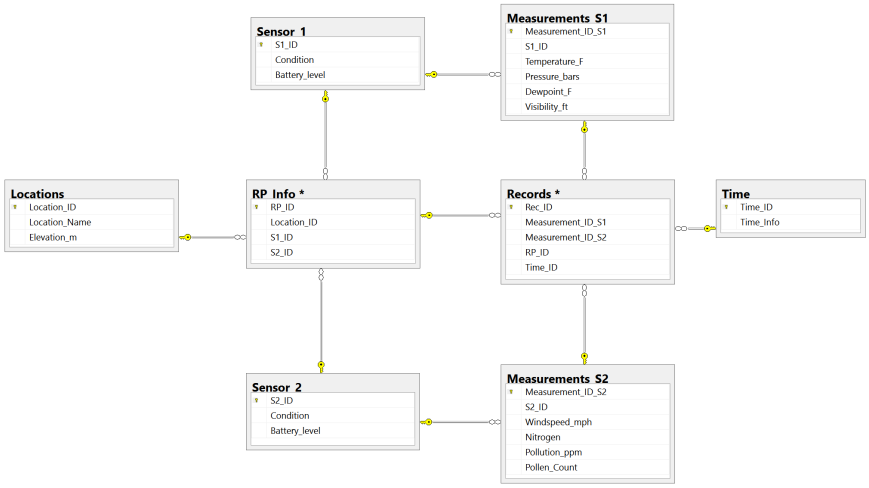
- **RP_Info:** "Raspberry Pi Information" will consist of the Raspberry Pi ID (PK), Location ID (FK), S1_ID (FK), and S2_ID (FK). The purpose of this table is to relate all the Raspberry Pi information together. Each RP will have two sensors on it, that each record different measurements, and each RP will be placed in a different location across GCU's campus. As we might increase the amount and type of sensors on the Raspberry Pi, we

will just add extra columns to the table that are foreign keys to the future new sensors table. When we want to change the location of where each Raspberry Pi is place, we can simply update the location ID. Same logic applies when we want to switch the sensors for the Raspberry Pi. Limitation: the table assumes that for each Raspberry Pi, it will not have duplicated types of sensor. For example, the assumption is that for one Raspberry Pi, it will not have two Type_2 sensors.

- **Records:** This table will contain a Record ID (PK), Sensor1 Measurement ID (FK), Sensor2 Measurement ID (FK), Raspberry Pi ID (FK), and Time ID (FK). This table relates the time to the measurements recorded and assigns them to a RP. In a sense, the Record contains that "what", "where", and "when" information about each environment measurement. "What" information is all the measurements conducted by different types of sensors. "Where" information say that the measurement is recorded using this Raspberry Pi. "When" information is recorded in the Time table, linked by Time ID. Limitation: We are assuming that the location of each Raspberry Pi does not change. Since the Records table does not contain direct location information of each record, but relies on the location information of the Raspberry Pi, if the location of the Raspberry Pi changes, the original measurement location of related record will be lost.
- **Time:** This table will contain Time_ID (PK) and Time_Info. It stores the time of the measurements taken. Since all records are measured at different times, the Time table and Records table have a one to one relationship.
- **Locations:** This table will contain the Location ID (PK), Location Name and Elevation of the Raspberry Pis. Having the elevation of the location is important because it is another factor that would influence the environment measurement.
- **Sensor_1:** This table will contain the ID for the first type of sensor, S1_ID (PK). It will also contain the condition of the sensor as well as the battery level. We plan to change the name of the table to the actual name of the sensor type when we figure it out. The reason we made each type of sensor an individual table is that different types of sensor measures different things (same goes to S2 table).
- **Measurements_S1:** This table will contain the Measurement_ID_S1 (PK), the S1_ID (FK), and the three factors it is responsible of measuring. These factors are temperature, pressure, and dewpoint. The corresponding units are included in the names of each of these fields. The three factors that the type_1 sensor measure are hypothetical and exemplary. We plan to change the factors after we find out what exactly does type_1 sensor measures (same goes to Measurments_S2 table).
- **Sensor_2:** This table will contain the ID for the second sensor, S2_ID (PK). It will also contain the condition of the sensor as well as the battery level.
- **Measurements_S2:** This table will contain the Measurement_ID_S2 (PK), the S2_ID (FK), and the four factors it is responsible of measuring. These factors are windspeed, pollution, nitrogen, and pollen. The corresponding units are included in the names of each of these fields.

Database Diagram and Tables Screenshots

Database Diagram



Locations Table

JOSH\SQLEXPRESS.S...F - dbo.Locations | JOSH\SQLEXPRESS.SMURF - Diagram_0

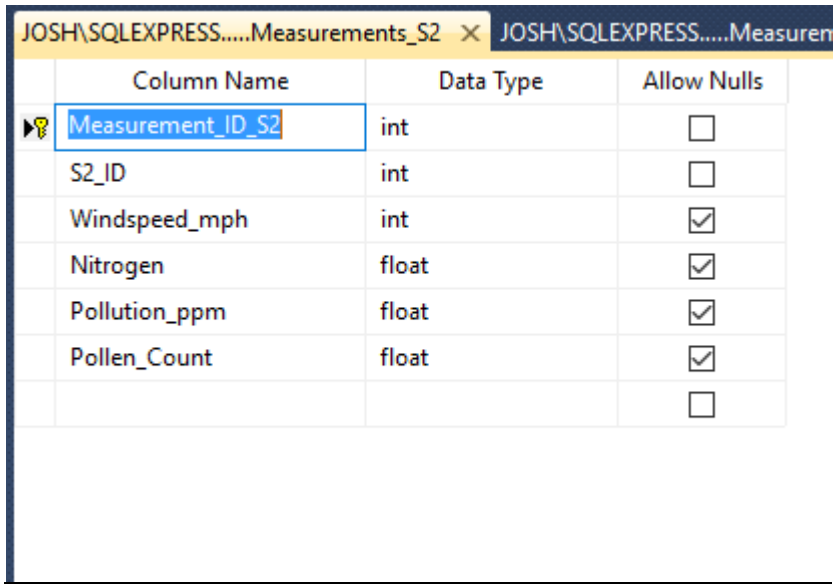
Column Name	Data Type	Allow Nulls
Location_ID	int	☐
Location_Name	varchar(20)	☐
Elevation_m	float	☐
		☐

Measurements_S1 Table

TOMMY\SQLEXPRES...Measurements_S1 | SQLQuery26.sql | SQLQuery11.sql - not connected*

Column Name	Data Type
Measurement_ID_S1	int
S1_ID	int
Temperature_F	int
Pressure_bars	int
Dewpoint_F	int
Visibility_ft	int

Measurements_S2 Table



Sensor_1 Table

JOSH\SQLEXPRESS.S...RF - dbo.Sensor_1 | JOSH\SQLEXPRESS.SMURF -

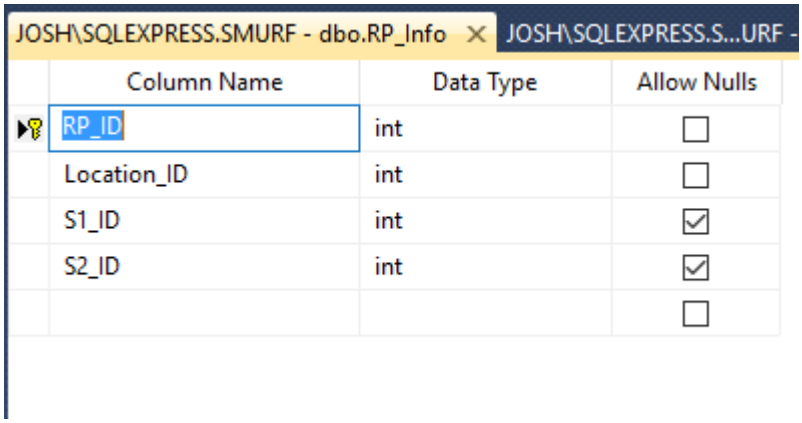
Column Name	Data Type	Allow Nulls
S1_ID	int	☐
Condition	varchar(10)	☑
Battery_level	varchar(10)	☑
		☐

Sensor_2 Table

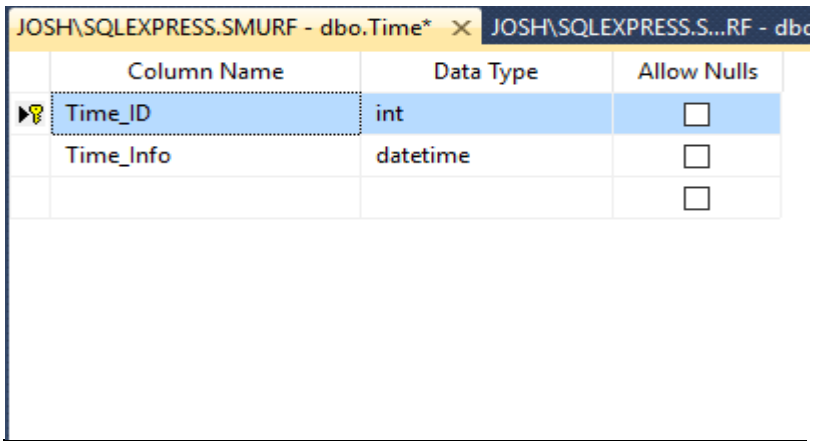
JOSH\SQLEXPRESS.S...RF - dbo.Sensor_2 | JOSH\SQLEXPRESS.S...RF - dbo.S

Column Name	Data Type	Allow Nulls
S2_ID	int	☐
Condition	varchar(10)	☑
Battery_level	varchar(10)	☑
		☐

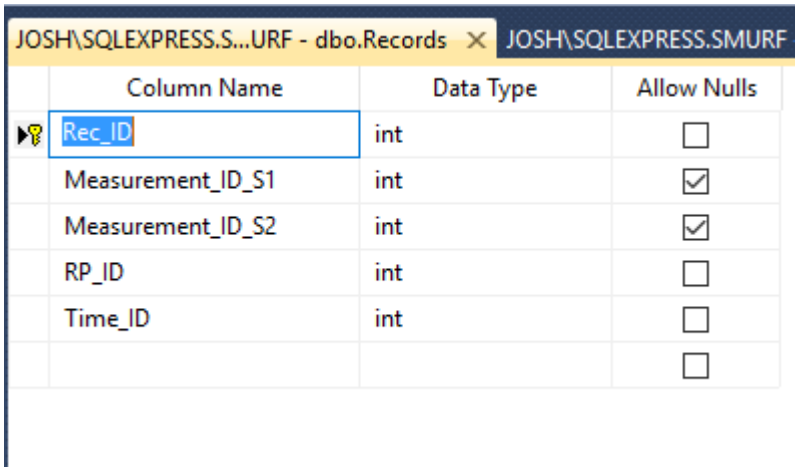
RP_Info Table



Time Table



Records Table



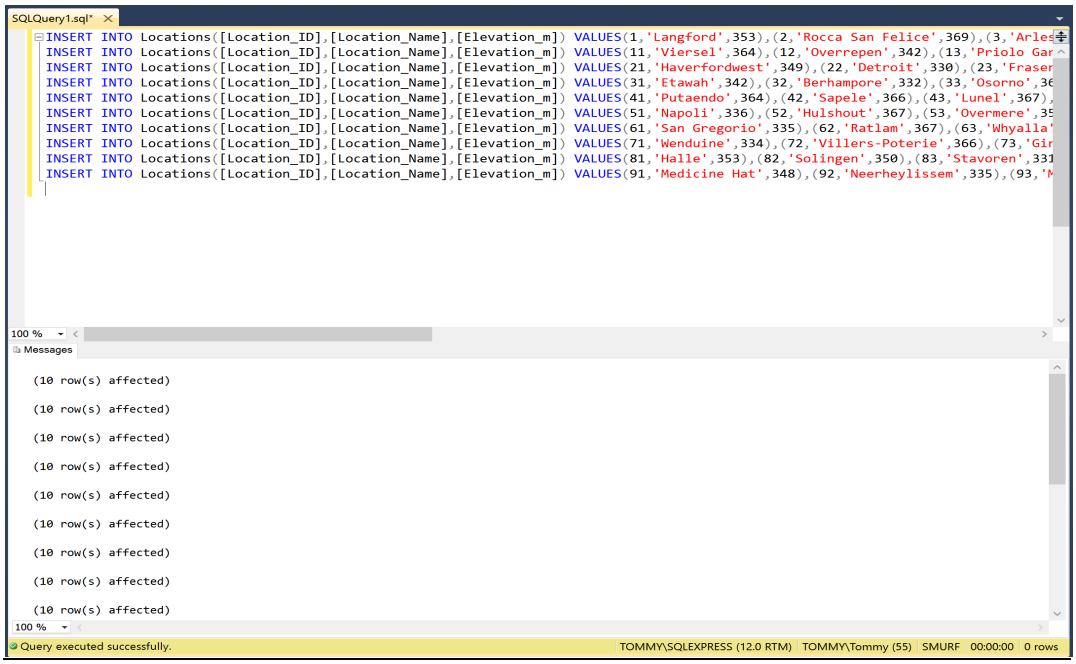
Map Mockup


82
20d
83
81
84
Location Name: Juniper Hall (Grove)
Time: February 3, 2017 12:00
Temperature (F) = 97
Pressure (Hg) = 30
Dew point(F) = 40
Wind speed (mph) = 3
Pollen Count = 4.70

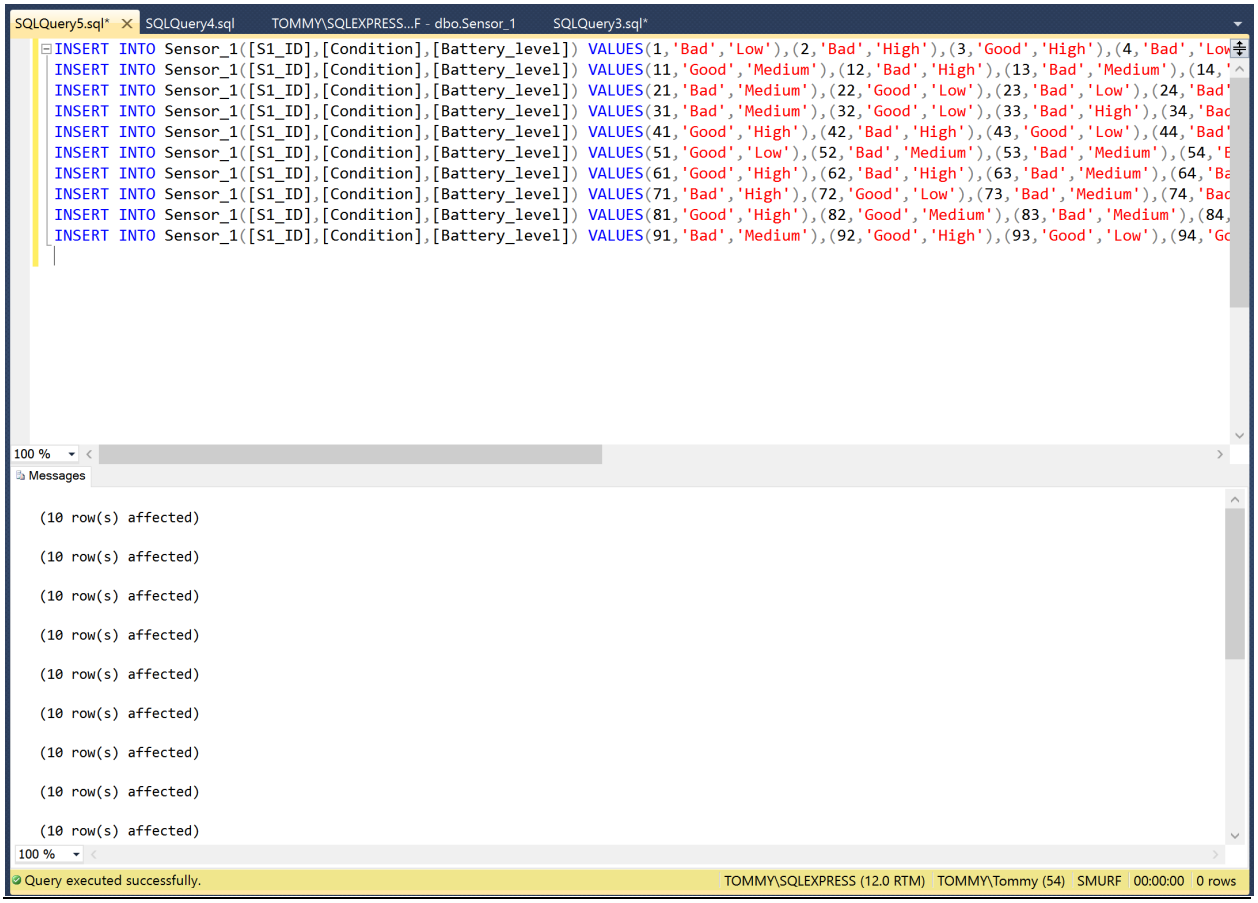
PART II: Integrity Constraint and Load

Insert Screenshots

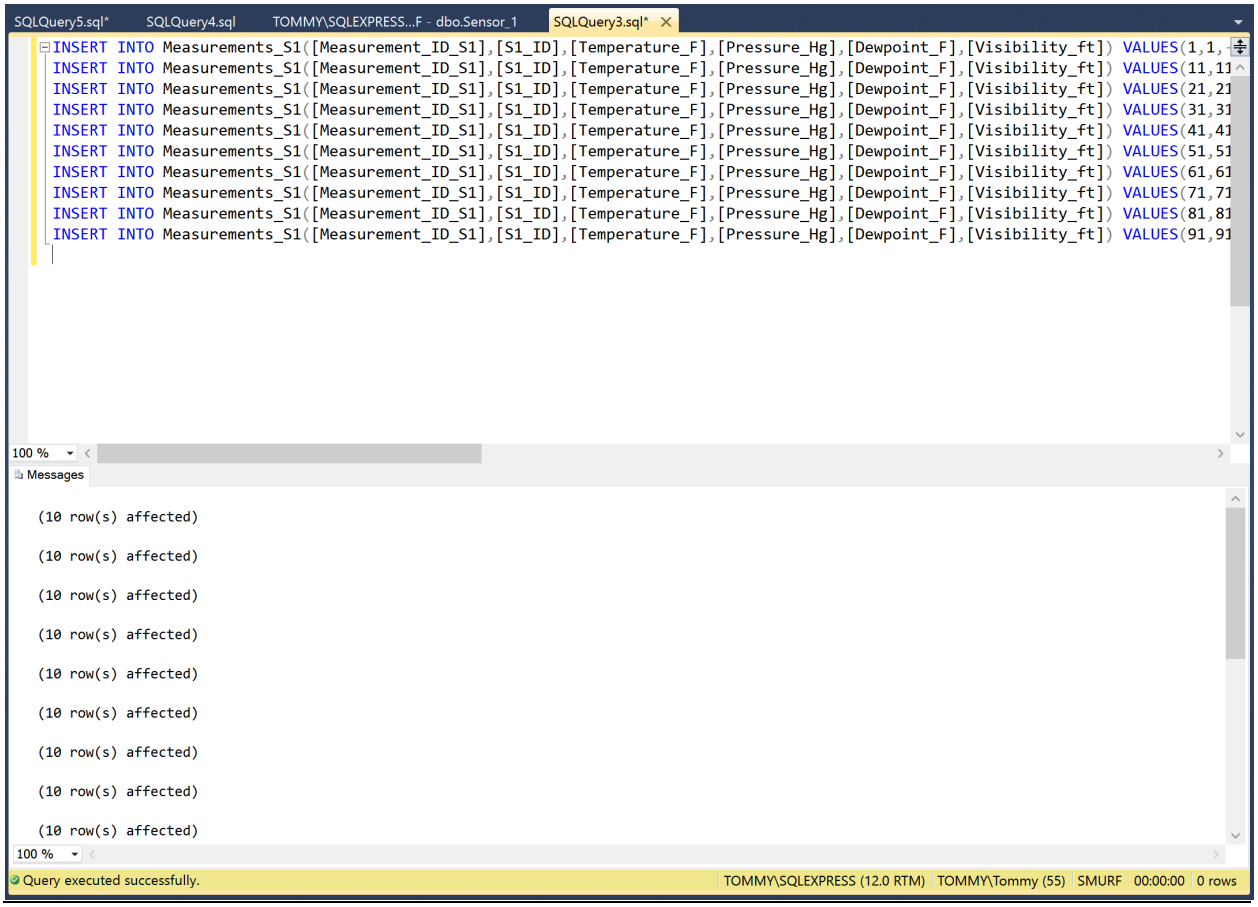
Location Insert Query



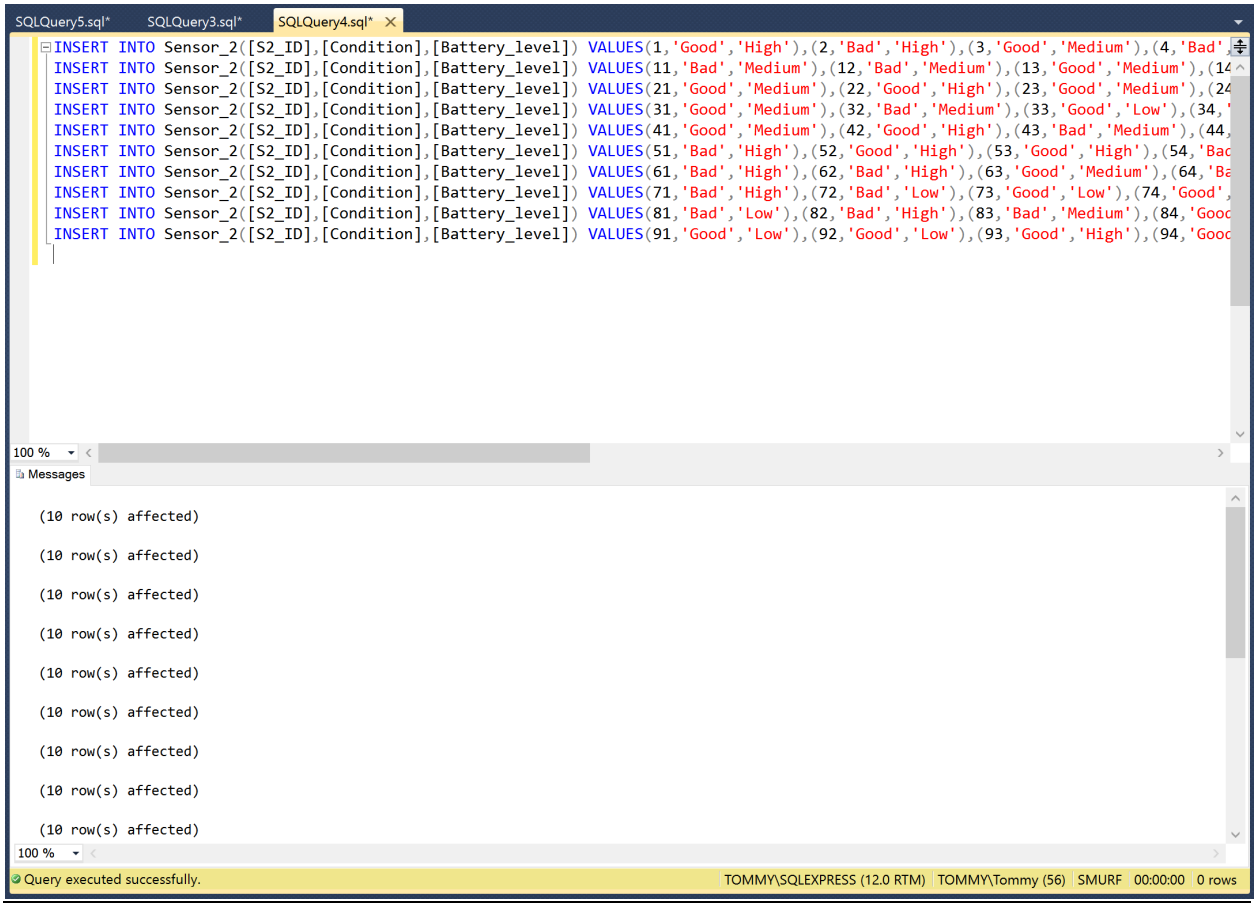
Sensor_1 Insert Query



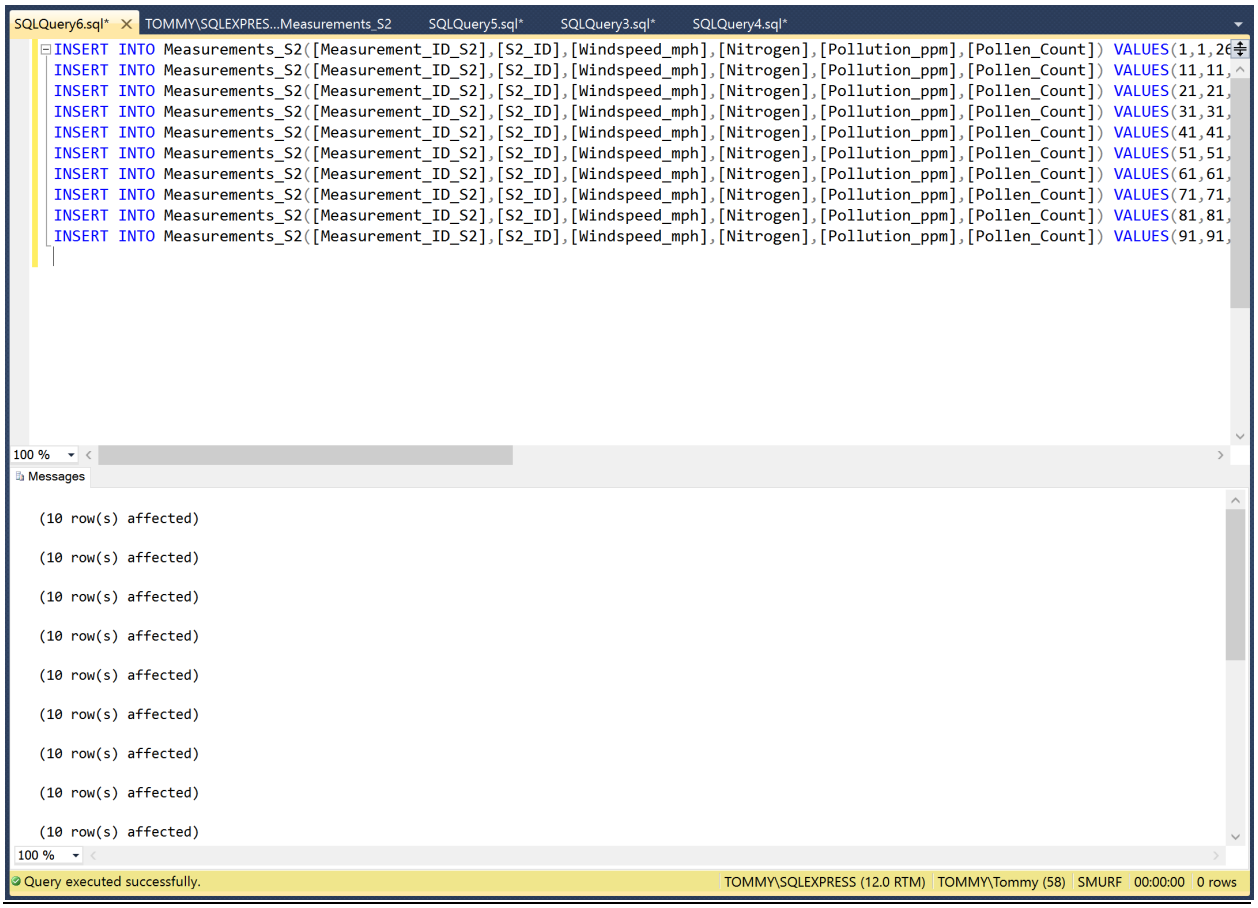
Measurements_S1 Insert Query



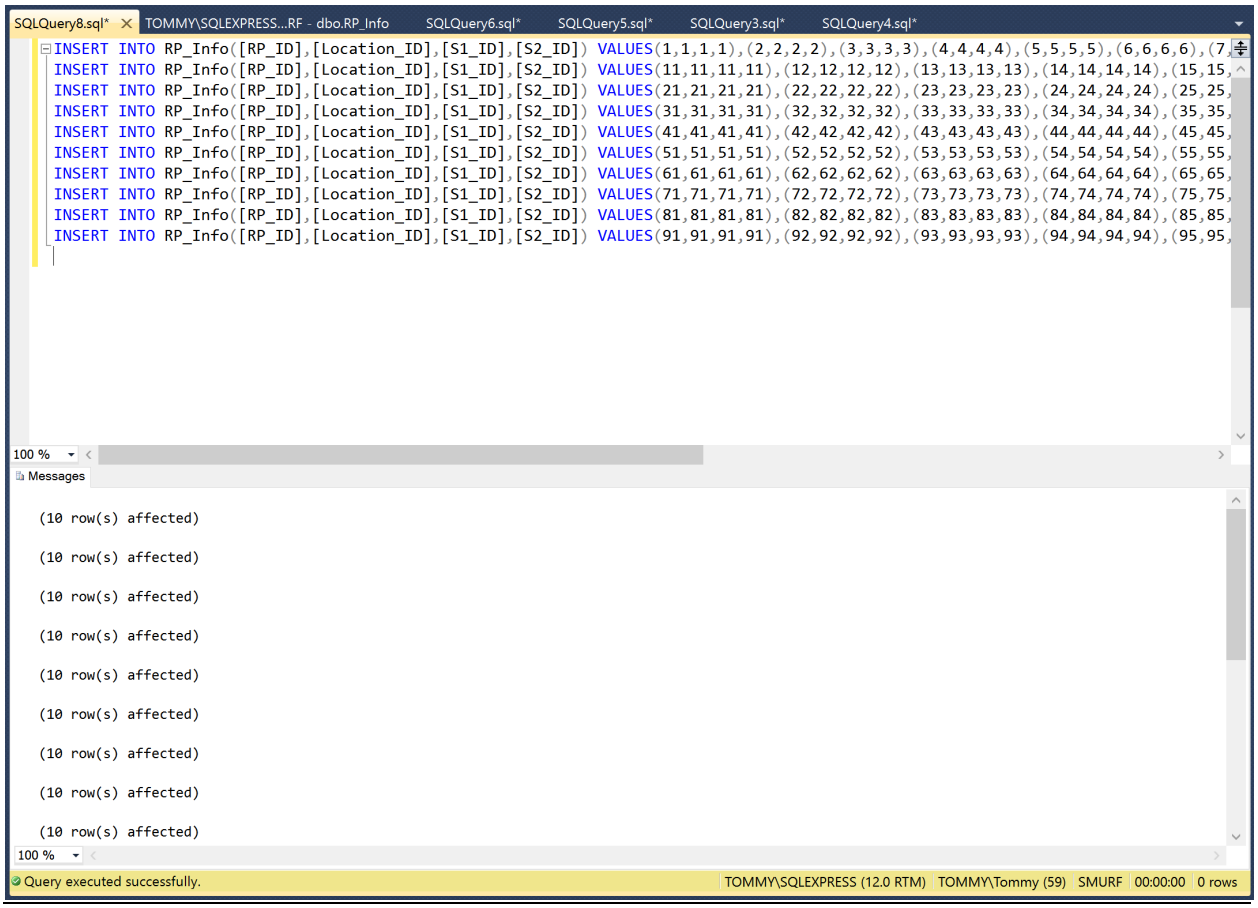
Sensor_2 Insert Query



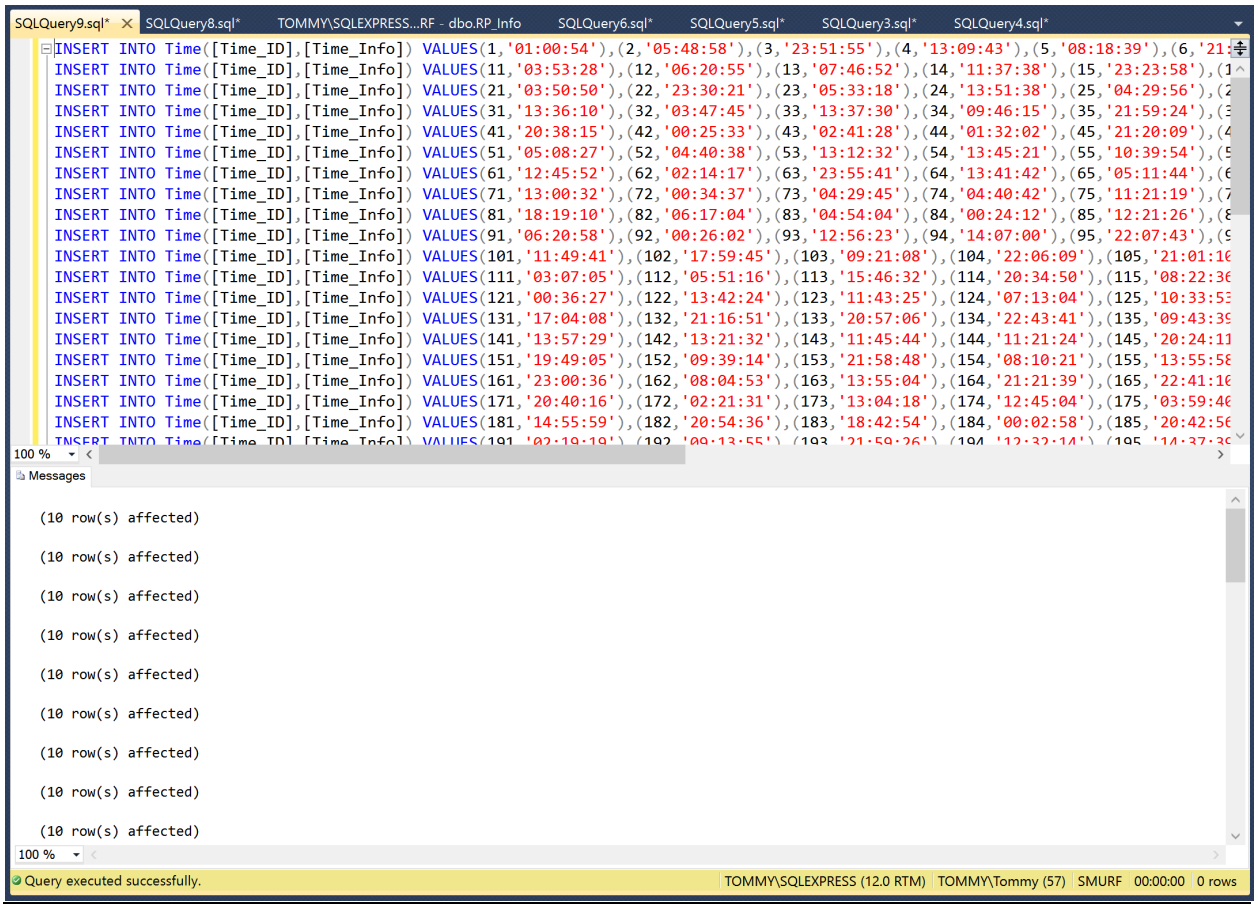
Measurements_S2 Insert Query



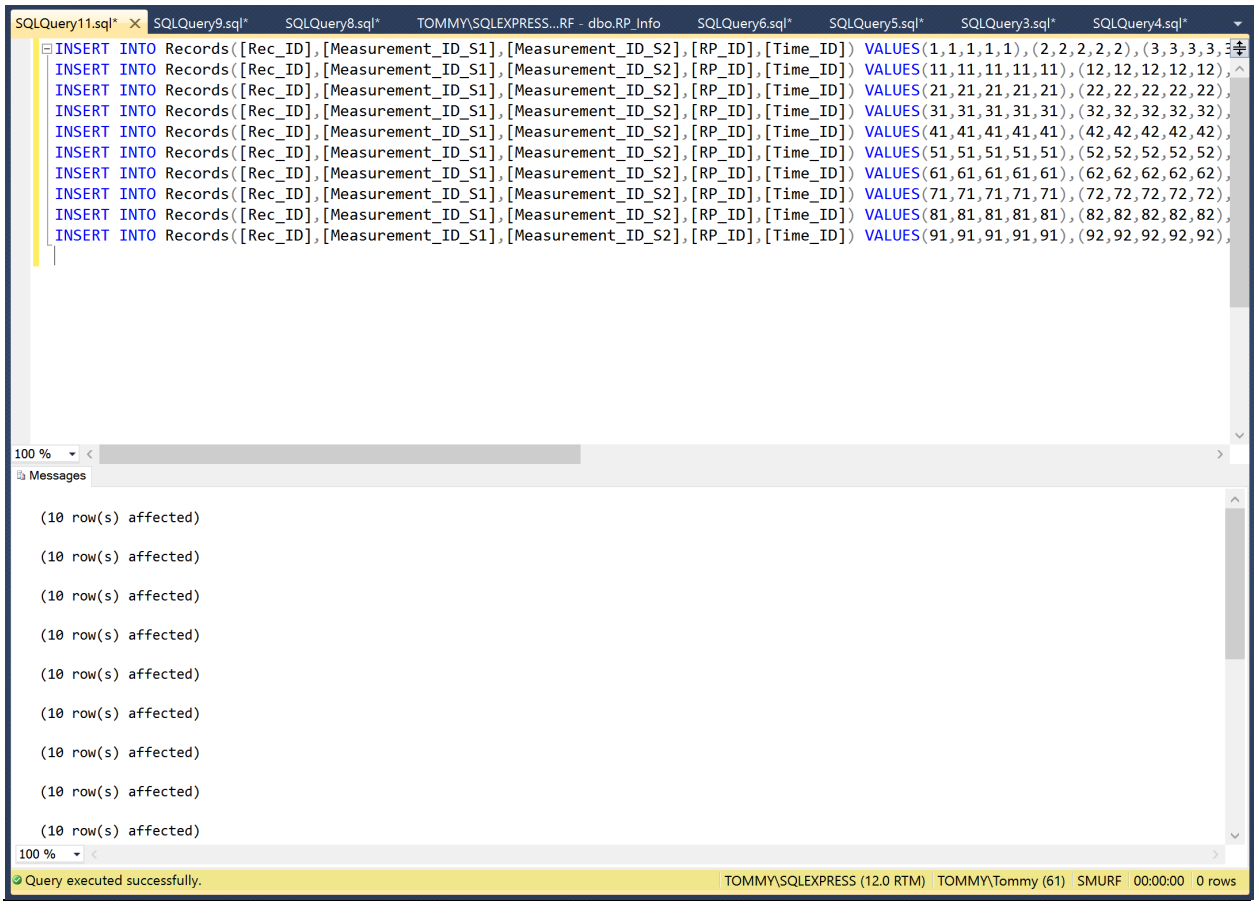
RP_Info Insert Query



Time Insert Query

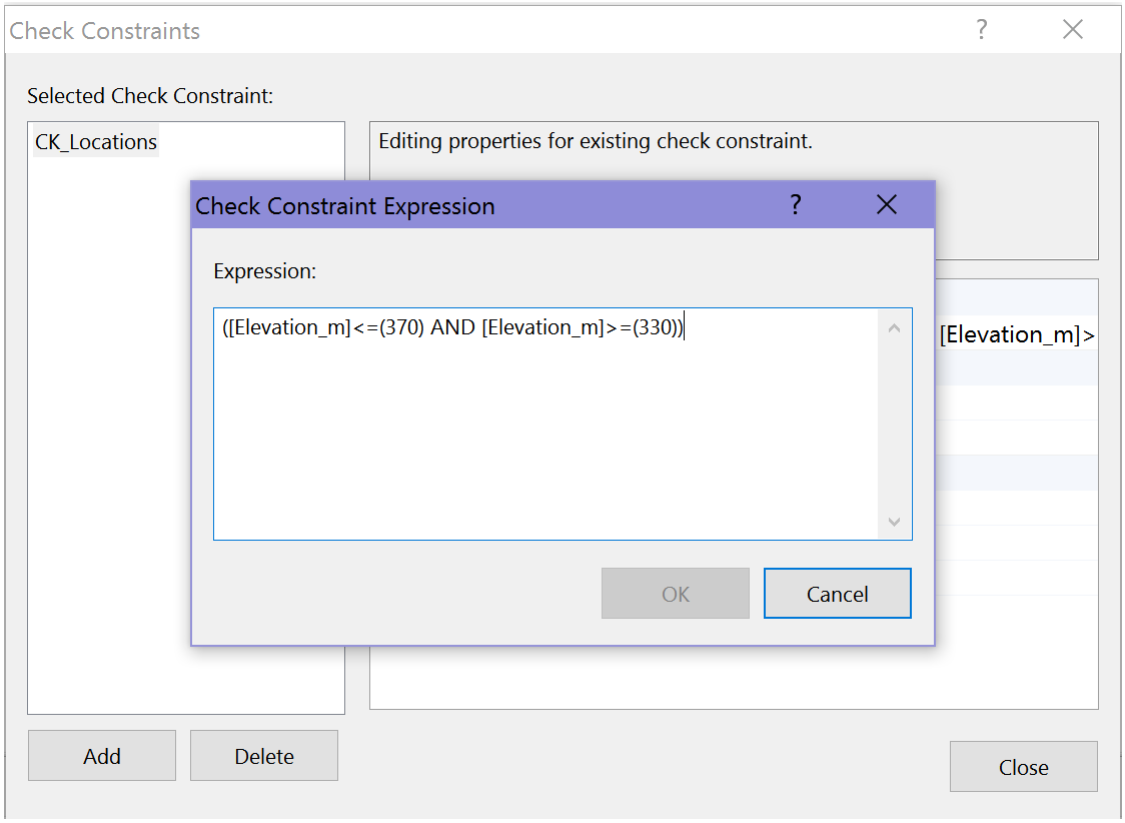


Records Insert Query



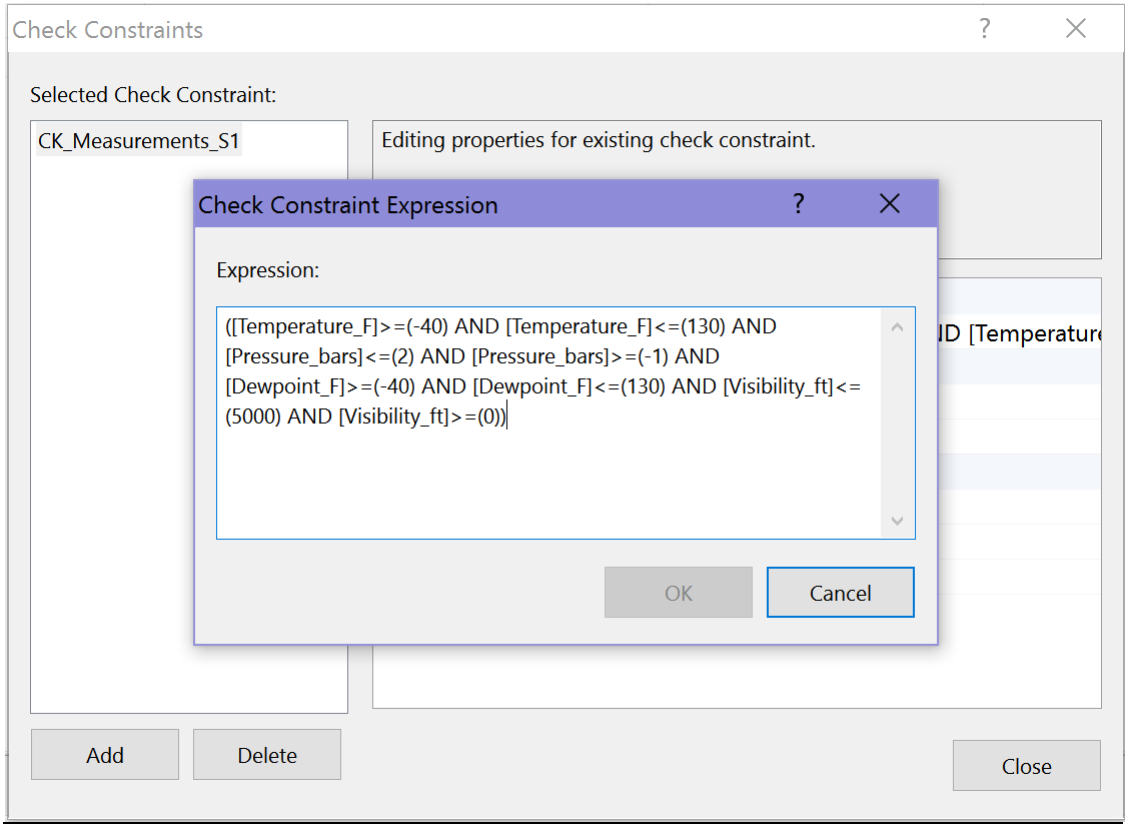
Constraints Screenshots and Rationale

Locations Constraints



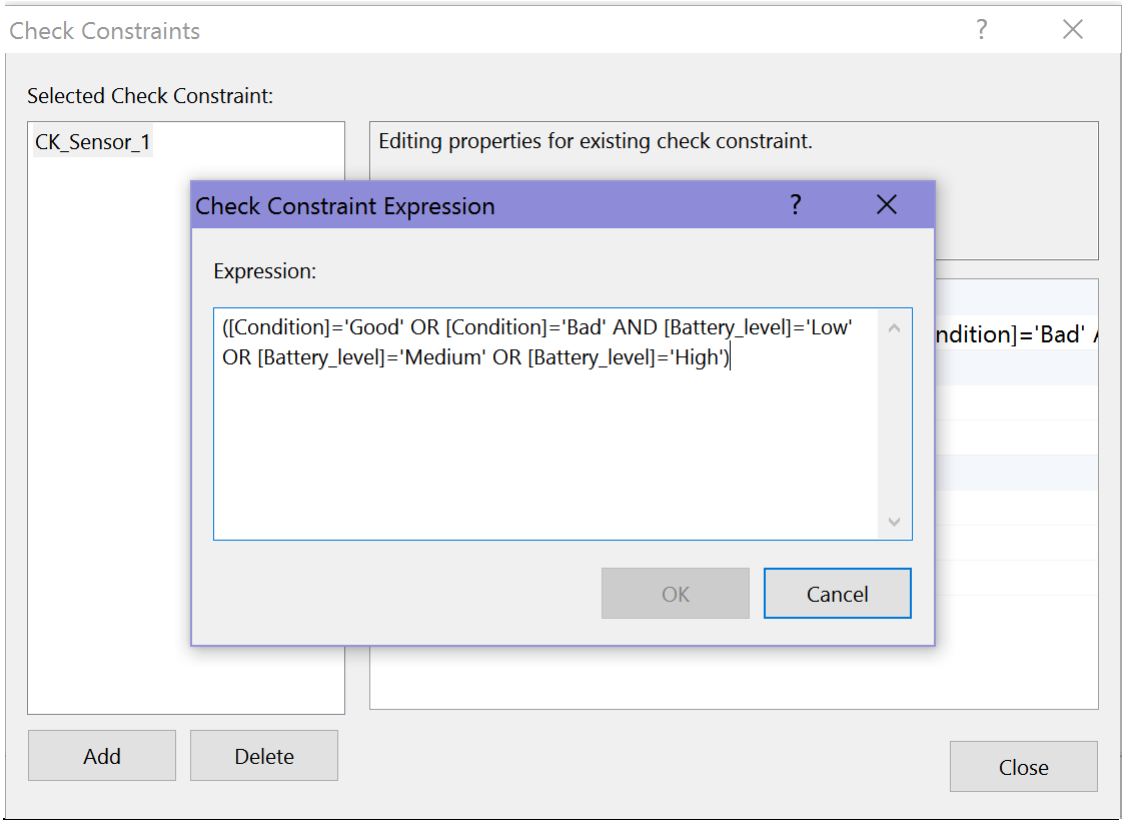
The reason for adding the constraints for elevation is to make sure that elevation is in reasonable range of value in unit of meters. When the value is not within the range, we can be certain that the data is wrong.

Measurements_S1 Constraints



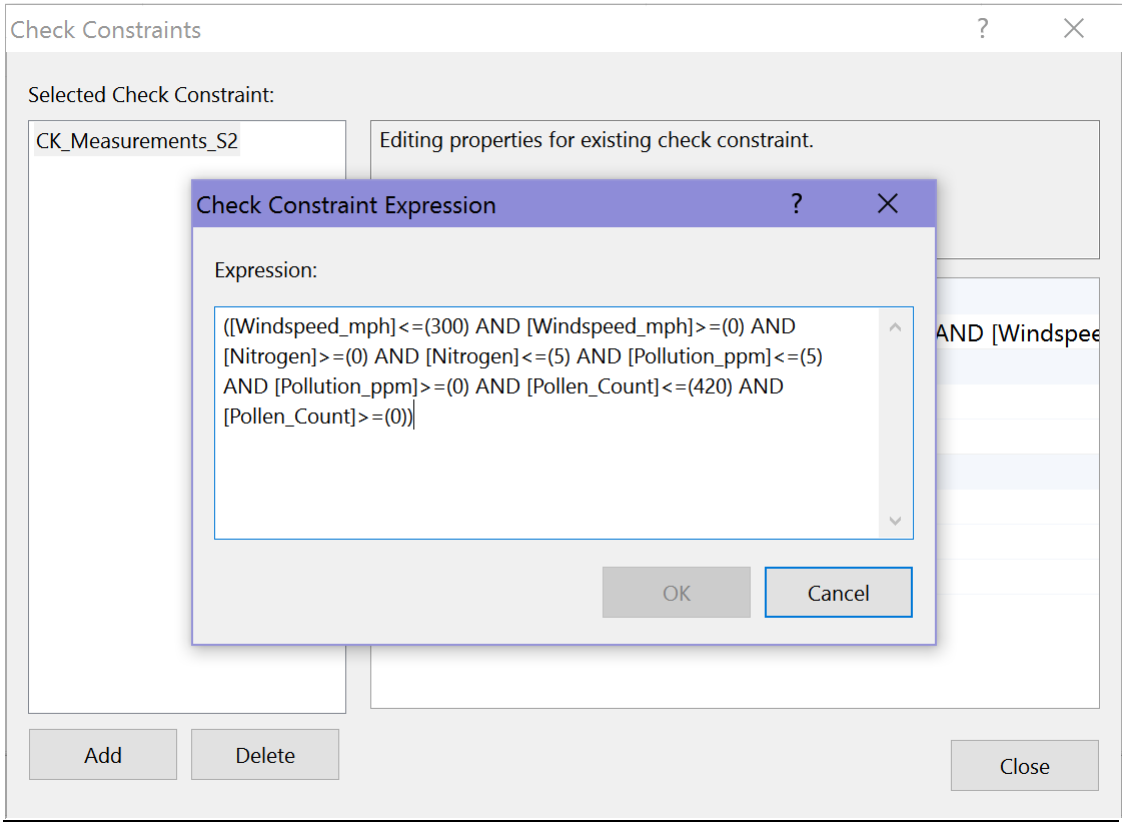
The reason for adding the constraints for temperature, pressure, dewpoint, and visibility is to make sure that these values is reasonable. When the data is off the range, we can be sure that the data is wrong. For example, the temperature in Phoenix would not exceed 130 or below -40 Fahrenheit, and visibility would not below 0.

Sensor_1 Constraints



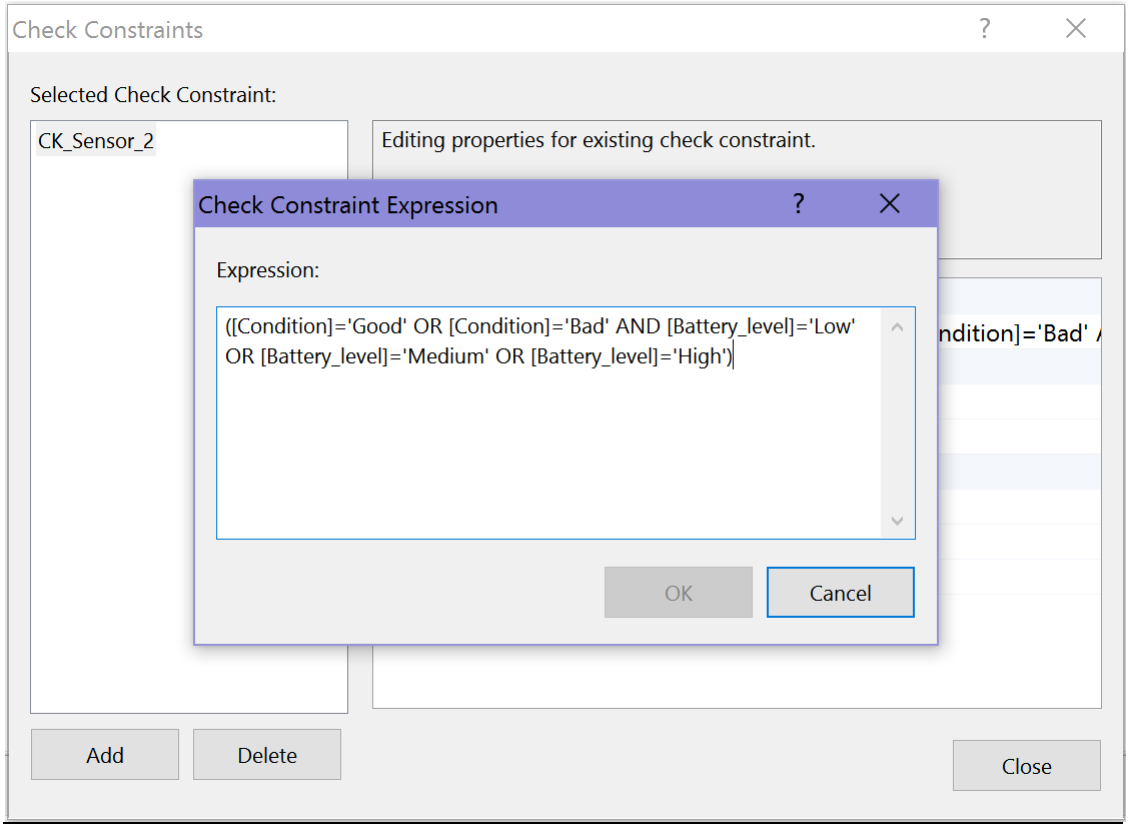
The reason to add the constraints for sensors is to make sure there are only two state for Condition: Good or Bad. Same logic goes to Battery level. There are only three fixed states for battery level: low, medium and high.

Measurements_S2 Constraints



The reason for adding the constraints for wind speed, nitrogen, pollution, and pollen count is to make sure that these values is reasonable. When the data is off the range, we can be sure that the data is wrong. For example, the wind speed in Phoenix would not exceed 300 or below -0 mph, and pollen count would not below 0.

Sensor_2 Constraints



The reason to add the constraints for sensors is to make sure there are only two state for Condition: Good or Bad. Same logic goes to Battery level. There are only three fixed states for battery level: low, medium and high.

PART III: Index and View Creation

Index and View Screenshots

Create View

SQLQuery11.sql*

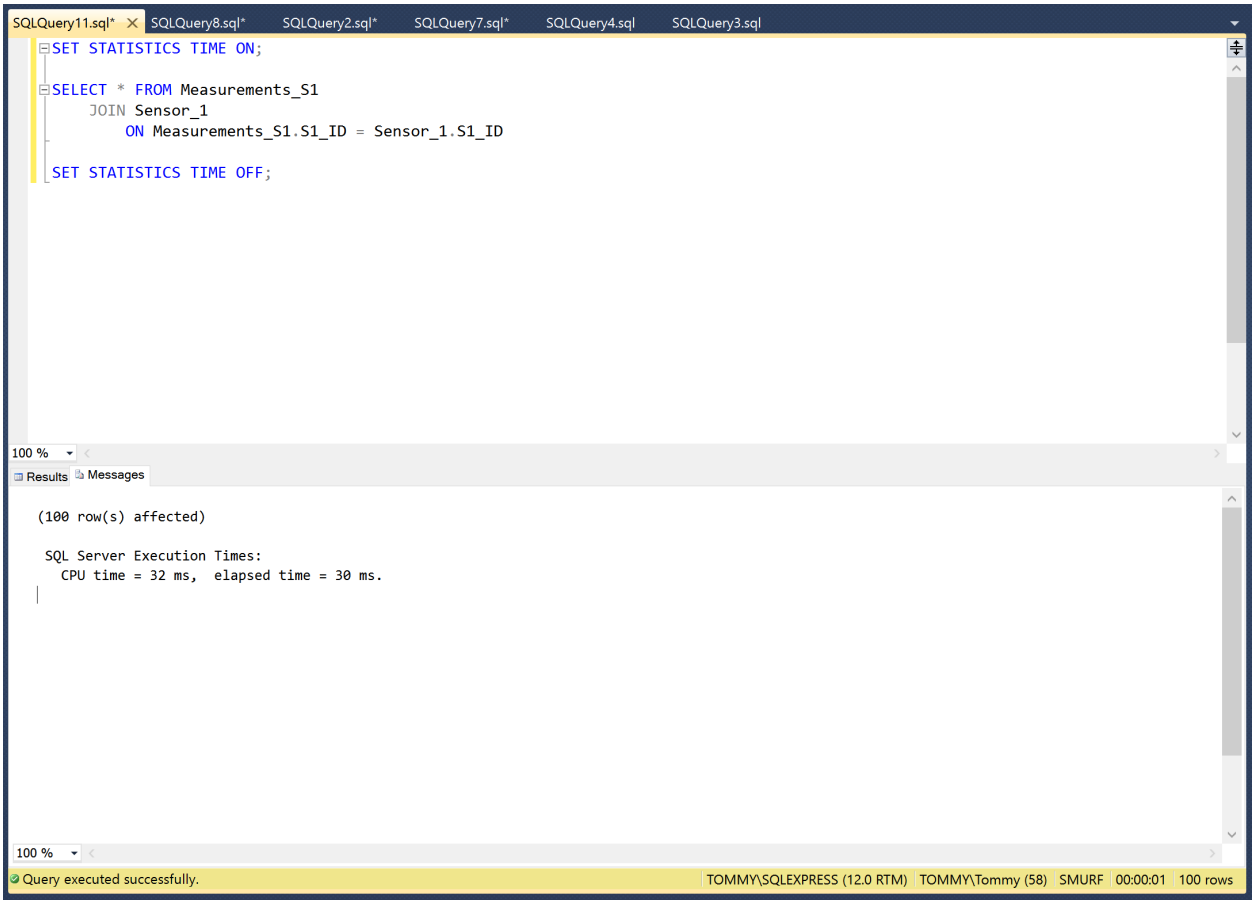
```
CREATE VIEW sensor_index AS
SELECT Measurement_ID_S1, Measurements_S1.S1_ID,Temperature_F,
Dewpoint_F, Visibility_ft, Condition, Battery_level
FROM Measurements_S1
    JOIN Sensor_1
        ON Measurements_S1.S1_ID = Sensor_1.S1_ID
```

100 %

Messages

```
Command(s) completed successfully.
```

Select and Join



Index on Measurements Primary Key

SQLQuery11.sql*

```
SET STATISTICS TIME ON;

SELECT * FROM Measurements_S1
    JOIN Sensor_1
        ON Measurements_S1.S1_ID = Sensor_1.S1_ID

SET STATISTICS TIME OFF;
```

100 %

Results | Messages

```
(100 row(s) affected)

 SQL Server Execution Times:
   CPU time = 31 ms,  elapsed time = 33 ms.
```

Index on Measurements Foreign Key

SQLQuery11.sql*

```
SET STATISTICS TIME ON;

SELECT * FROM Measurements_S1
    JOIN Sensor_1
        ON Measurements_S1.S1_ID = Sensor_1.S1_ID

SET STATISTICS TIME OFF;
```

100 %

Results | Messages

```
(100 row(s) affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 2 ms.
```

Index on Sensor Primary Key

SQLQuery11.sql*

```
SET STATISTICS TIME ON;

SELECT * FROM Measurements_S1
    JOIN Sensor_1
        ON Measurements_S1.S1_ID = Sensor_1.S1_ID

SET STATISTICS TIME OFF;
```

100 %

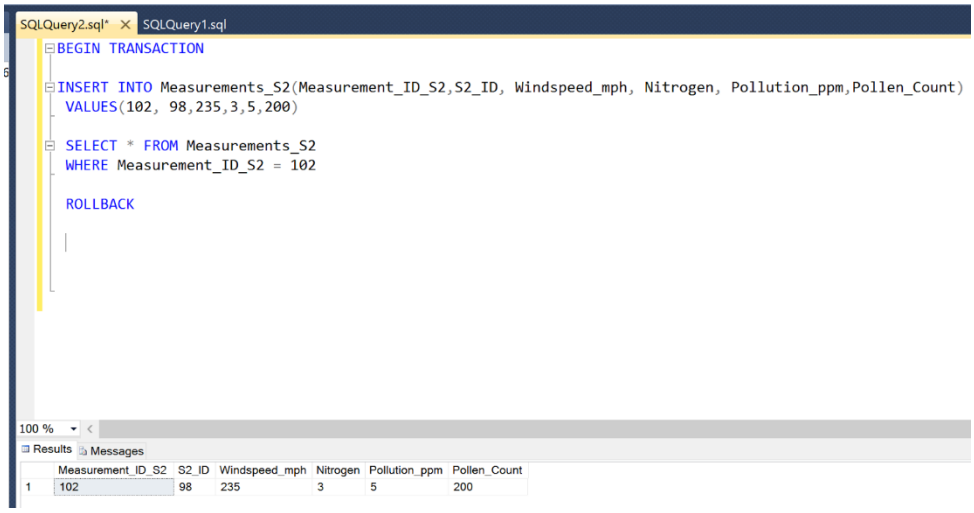
Results | Messages

```
(100 row(s) affected)

 SQL Server Execution Times:
   CPU time = 31 ms,  elapsed time = 35 ms.
```

PART IV: Transaction and Lock Management

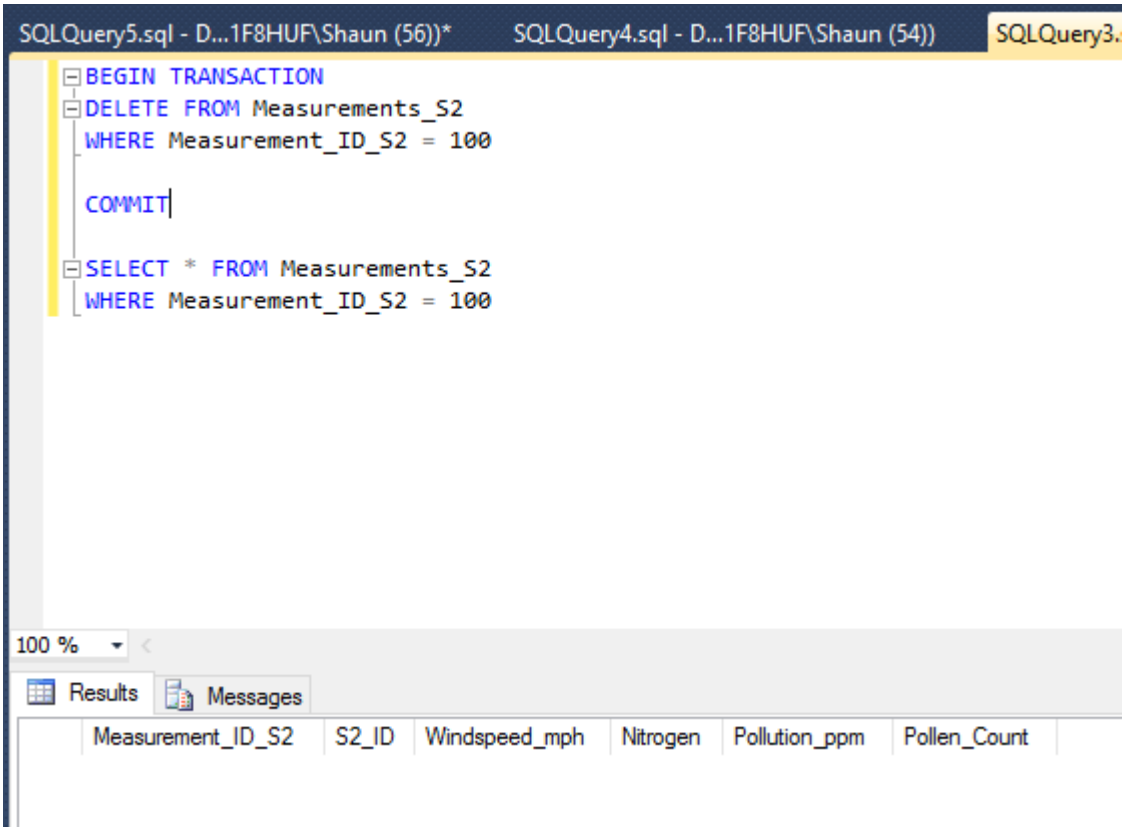
Screenshots

Insert (before)



Insert (After)

Delete


SQLQuery5.sql - D...1F8HUF\Shaun (56))*
SQLQuery4.sql - D...1F8HUF\Shaun (54))
SQLQuery3.
BEGIN TRANSACTION
DELETE FROM Measurements_S2
WHERE Measurement_ID_S2 = 100
COMMIT
SELECT * FROM Measurements_S2
WHERE Measurement_ID_S2 = 100
100 %
Results
Messages
Measurement_ID_S2
S2_ID
Windspeed_mph
Nitrogen
Pollution_ppm
Pollen_Count

SQLQuery5.sql - D...1F8HUF\Shaun (56))* SQLQuery4.sql - D...1F8HUF\Shaun (54)) SQLQu

```
BEGIN TRANSACTION
DELETE FROM Measurements_S2
WHERE Measurement_ID_S2 = 100

ROLLBACK

SELECT * FROM Measurements_S2
WHERE Measurement_ID_S2 = 100
```

100 %

Results Messages

	Measurement_ID_S2	S2_ID	Windspeed_mph	Nitrogen	Pollution_ppm	Pollen_Count
1	100	50	0	3	3	378

Update

SQLQuery3.sql - D...1F8HUF\Shaun (51))* SQLQuery2.sql - D...1F8HUF\Shaun (53))

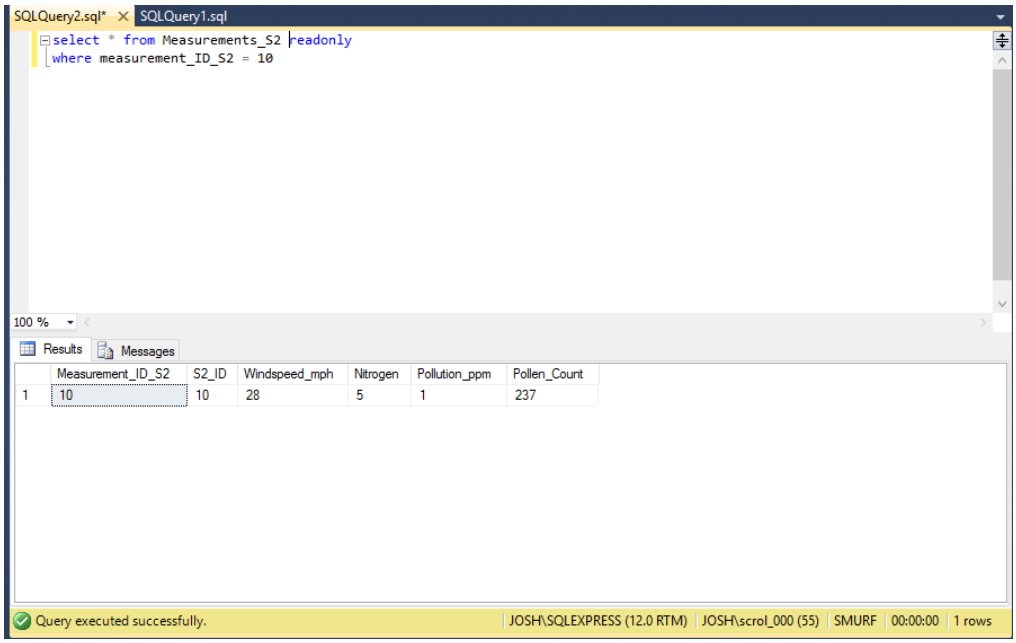
```
BEGIN TRANSACTION
UPDATE Measurements_S2
SET S2_ID = 50
WHERE Measurement_ID_S2 = 100

COMMIT

SELECT * FROM Measurements_S2
WHERE Measurement_ID_S2 = 100
```

100 %

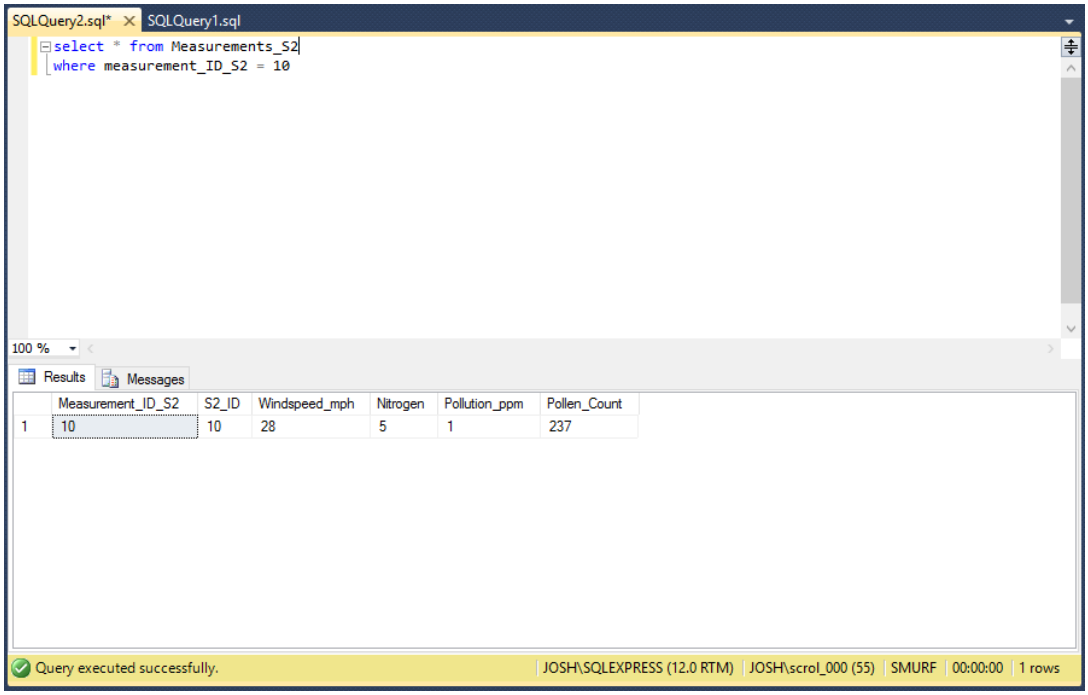
Results Messages

	Measurement_ID_S2	S2_ID	Windspeed_mph	Nitrogen	Pollution_ppm	Pollen_Count
1	100	50	0	3	3	378

Select With Read-Only

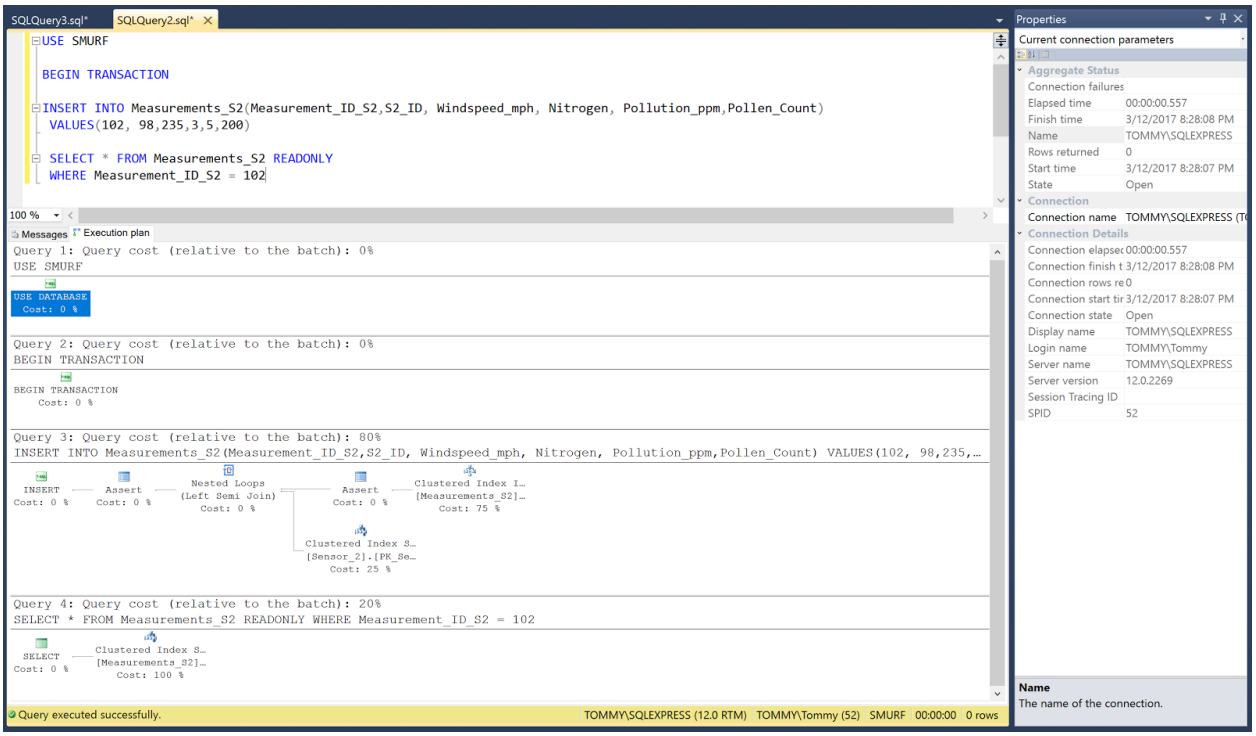


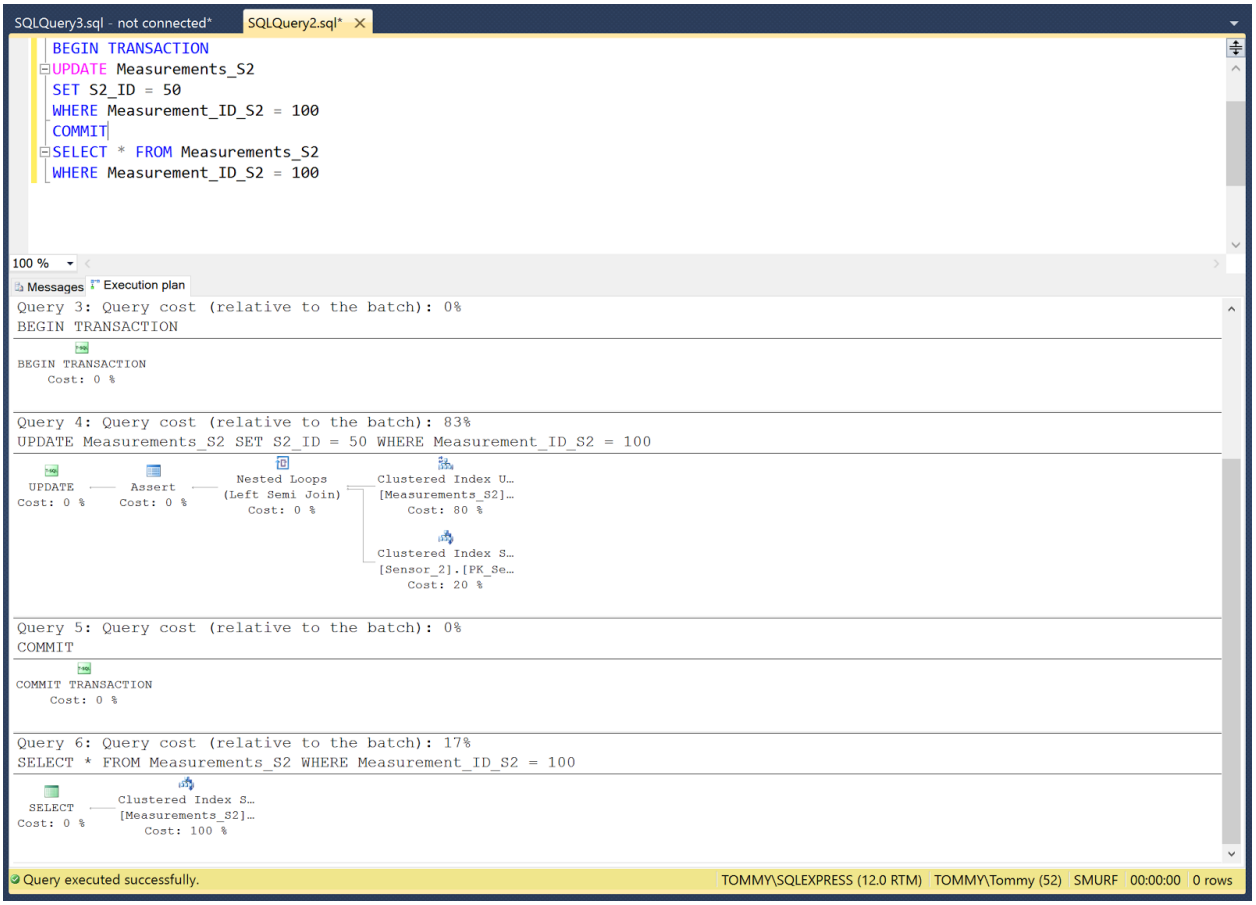
Select Without Read-Only



Estimated Execution Plan

An estimated execution plan represents the output of the query optimizer. The query optimizer models the way a relational database engine works. It takes the requested query and figures out the best way to implement it. The following screenshots demonstrate the estimated execution plan in action:



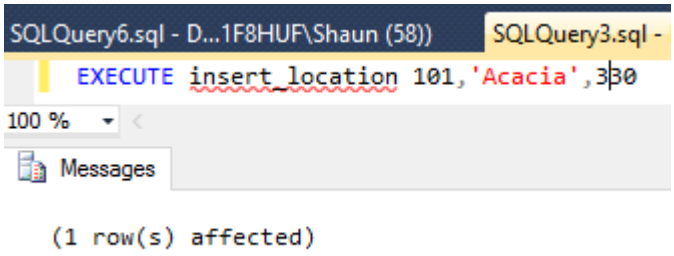

SQLQuery3.sql - not connected*
SQLQuery2.sql*
BEGIN TRANSACTION
UPDATE Measurements_S2
SET S2_ID = 50
WHERE Measurement_ID_S2 = 100
COMMIT
SELECT * FROM Measurements_S2
WHERE Measurement_ID_S2 = 100
100 %
Messages
Execution plan
Query 3: Query cost (relative to the batch): 0%
BEGIN TRANSACTION
BEGIN TRANSACTION
Cost: 0 %
Query 4: Query cost (relative to the batch): 83%
UPDATE Measurements_S2 SET S2_ID = 50 WHERE Measurement_ID_S2 = 100
UPDATE
Cost: 0 %
Assert
Cost: 0 %
Nested Loops
(Left Semi Join)
Cost: 0 %
Clustered Index U…
[Measurements_S2]…
Cost: 80 %
Clustered Index S…
[Sensor_2].[PK_Se…
Cost: 20 %
Query 5: Query cost (relative to the batch): 0%
COMMIT
COMMIT TRANSACTION
Cost: 0 %
Query 6: Query cost (relative to the batch): 17%
SELECT * FROM Measurements_S2 WHERE Measurement_ID_S2 = 100
SELECT
Cost: 0 %
Clustered Index S…
[Measurements_S2]…
Cost: 100 %
Query executed successfully.
TOMMY\SQLEXPRESS (12.0 RTM)
TOMMY\Tommy (52)
SMURF
00:00:00
0 rows

PART V: TSQL Development

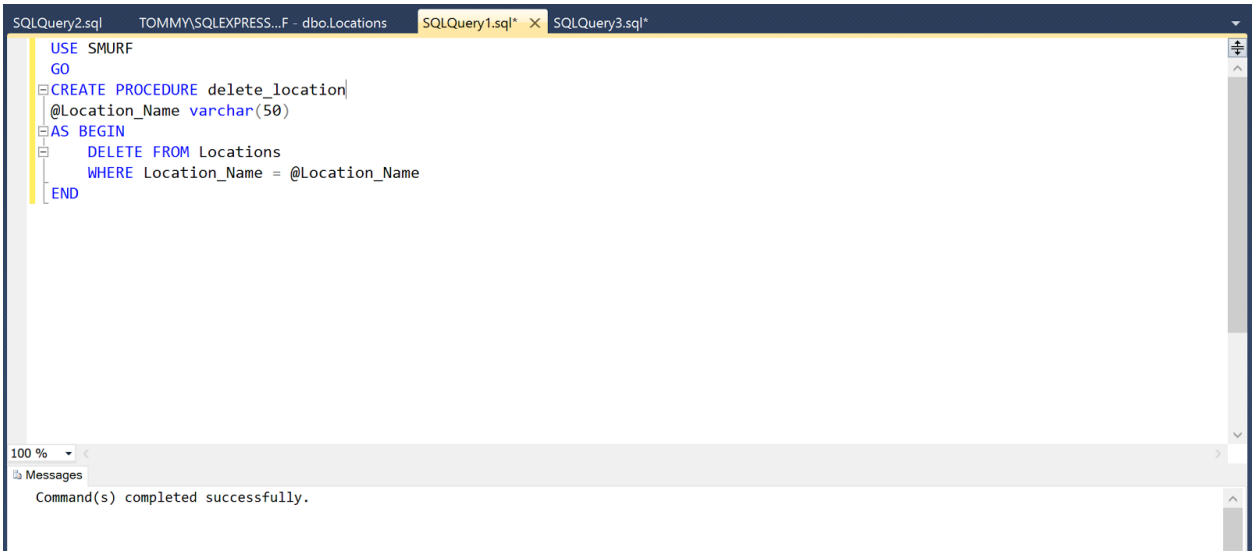
Screenshots

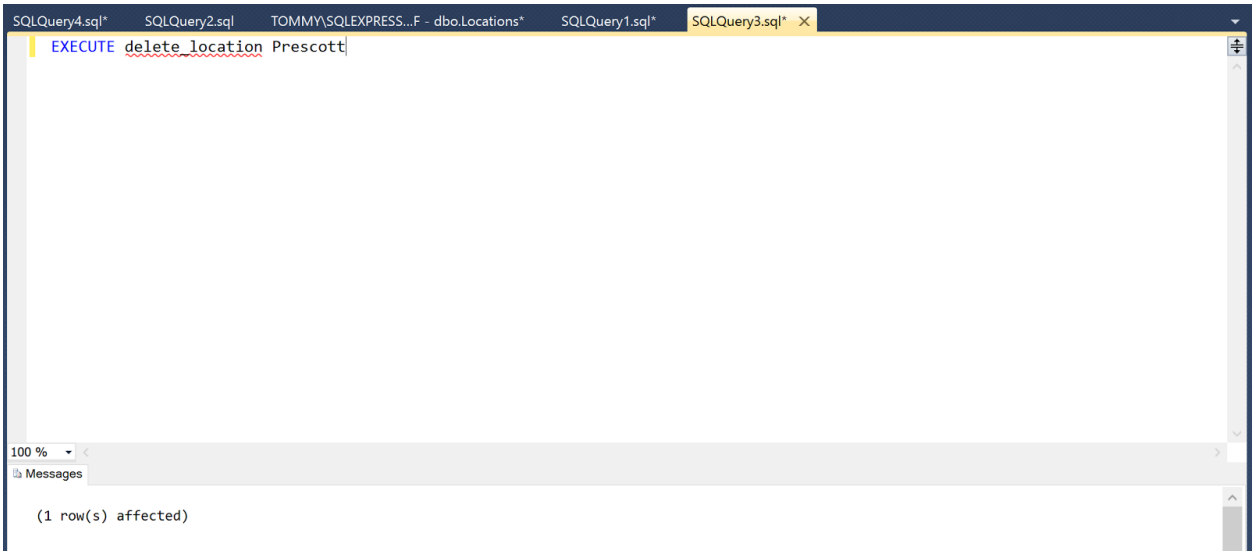
Stored Procedure: Insert

```
USE [SMURF(III)]
GO
/****** Object:  StoredProcedure [dbo].[insert_location]    Script Date: 3/28/2017 6:35:30 PM
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[insert_location]
@Location_ID int,
@Location_Name varchar(50),
@Elevation_m float
AS
BEGIN
INSERT INTO Locations(Location_ID, Location_Name, Elevation_m)
VALUES(@Location_ID, @Location_Name, @Elevation_m)
END
```

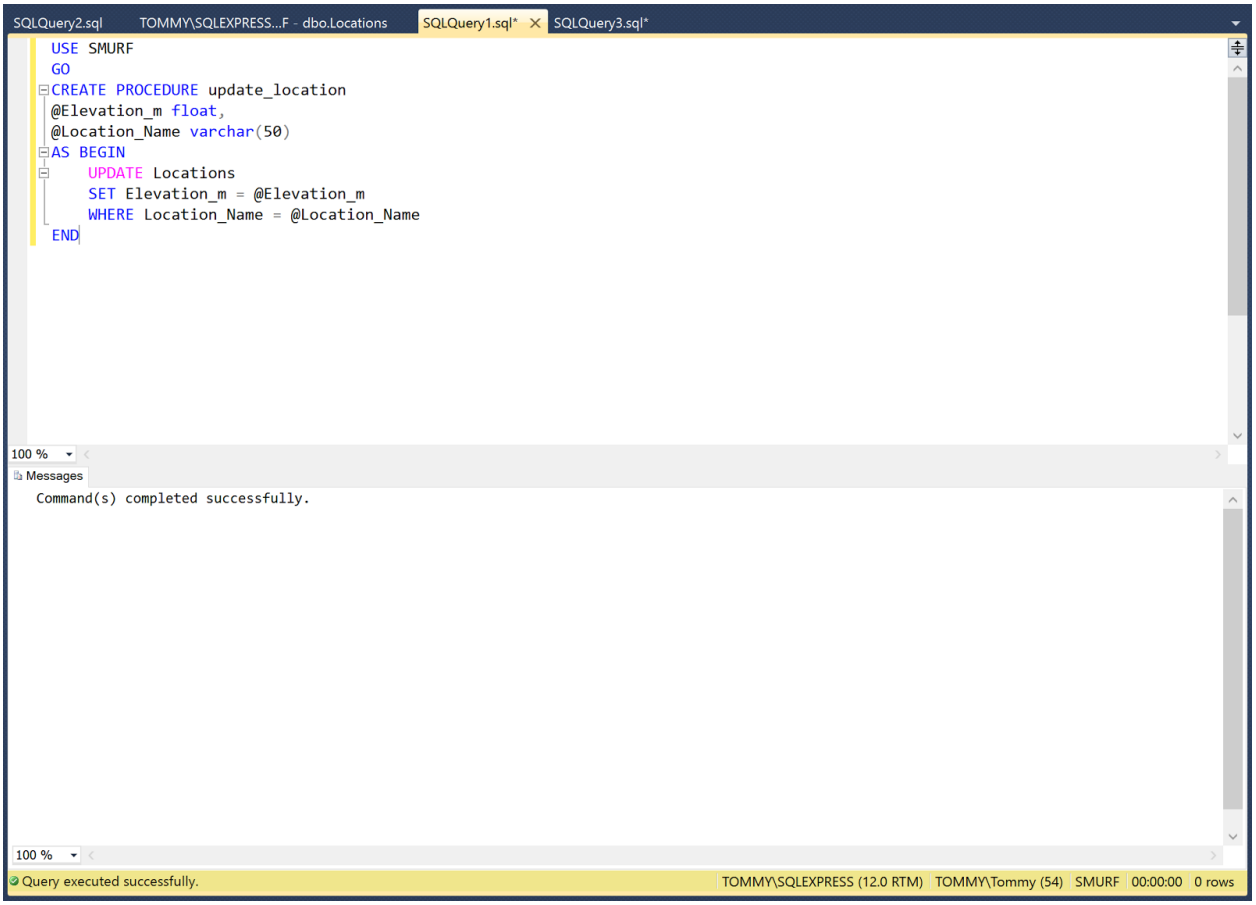


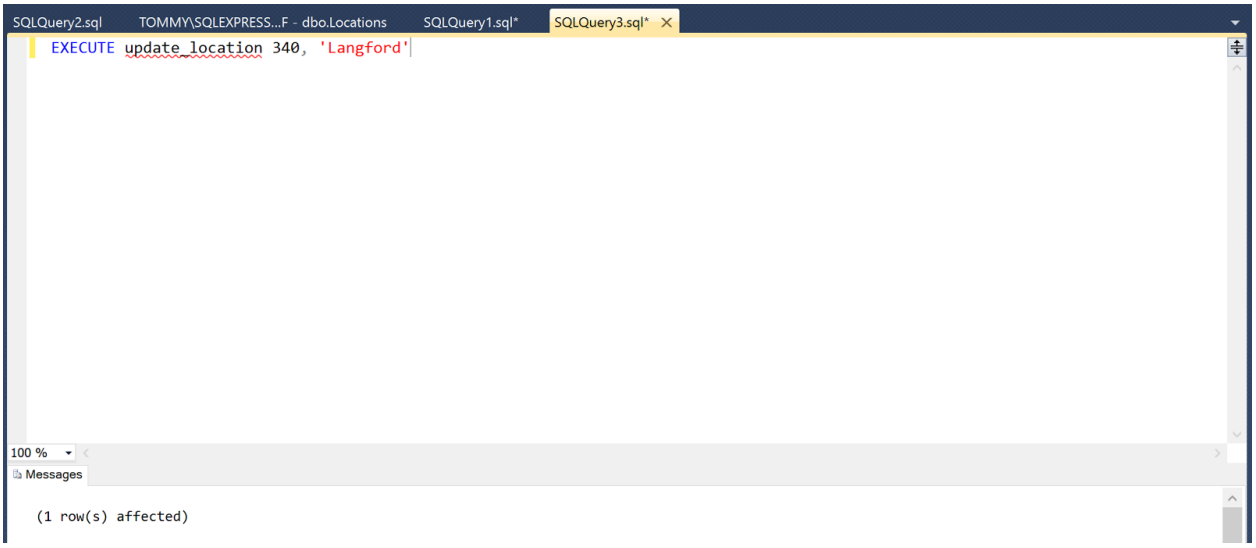
Stored Procedure: Delete



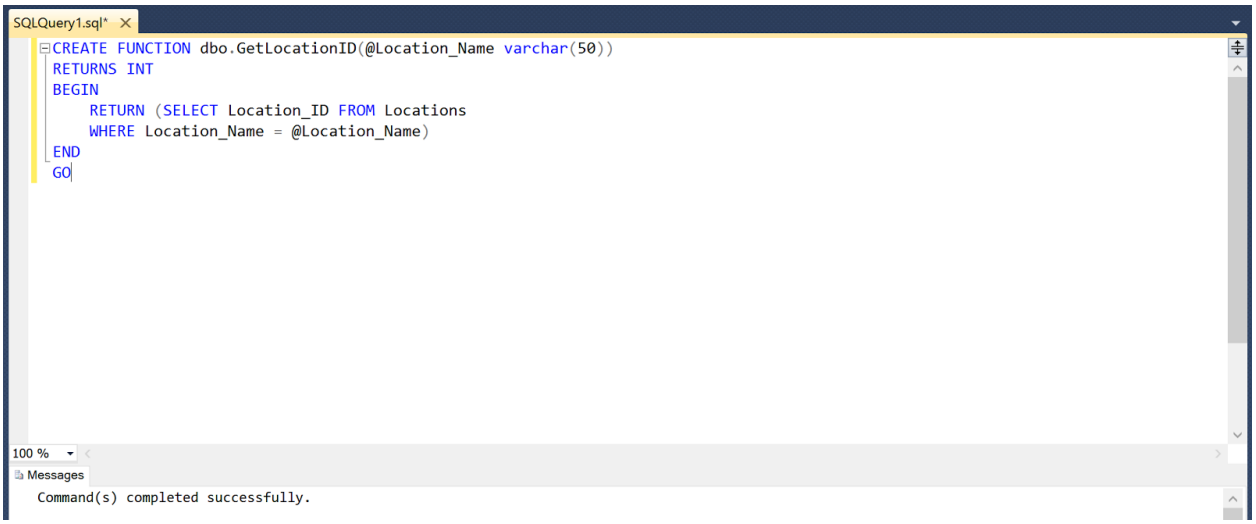


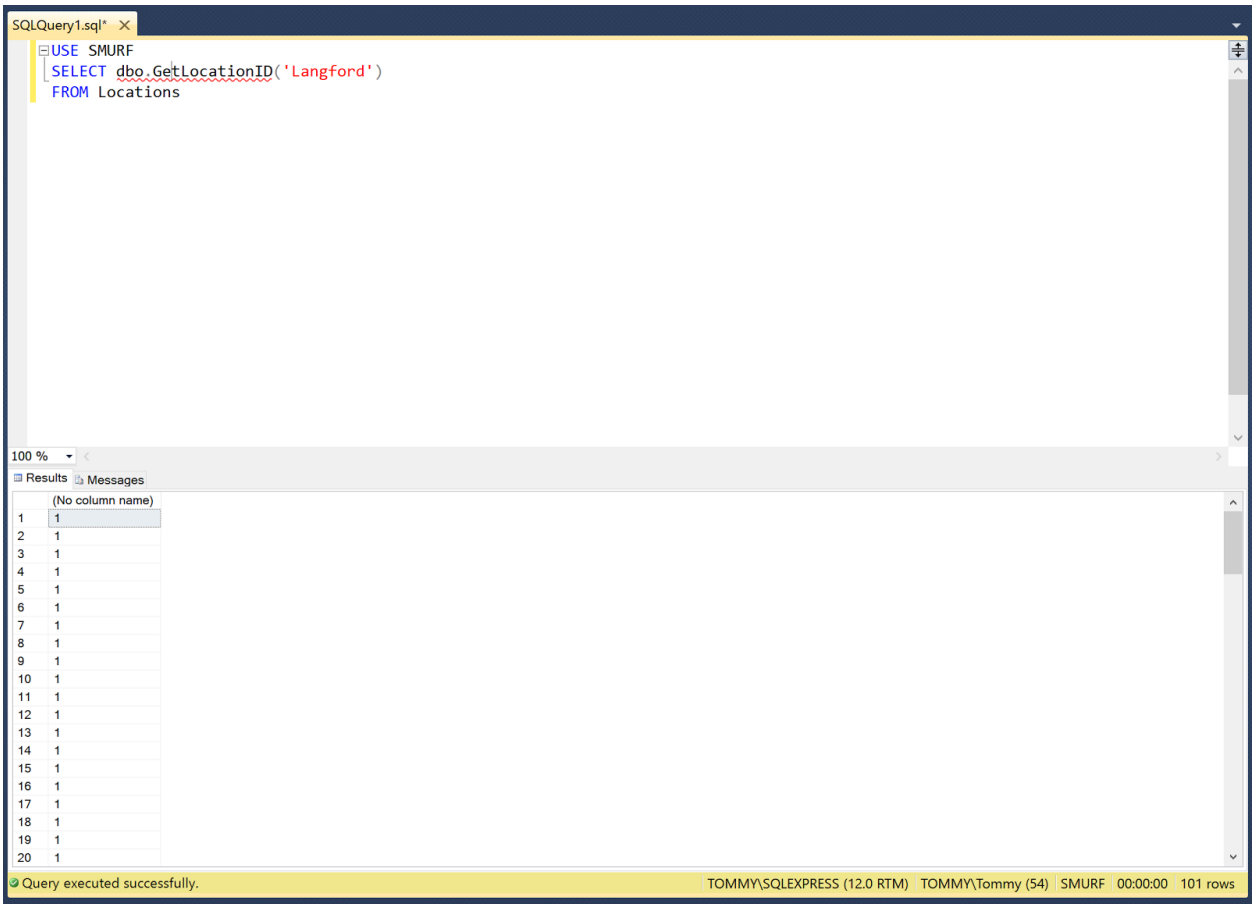
Stored Procedure: Update



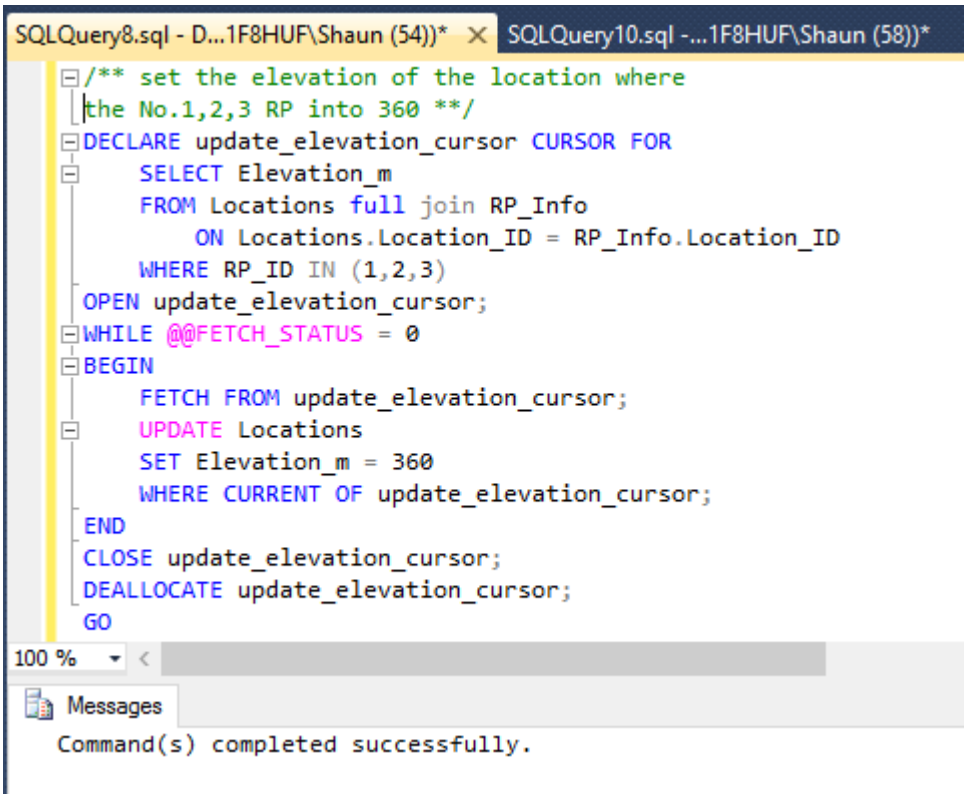


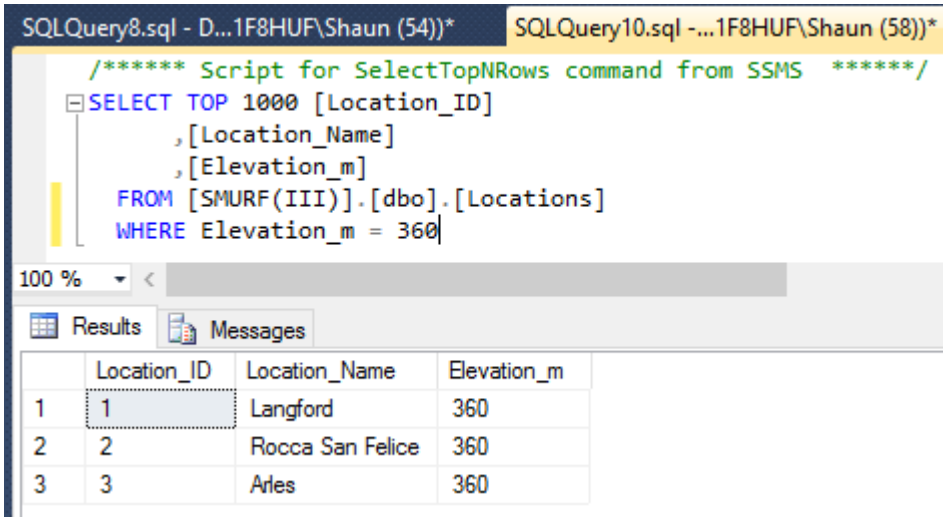
UDF Scalar Function



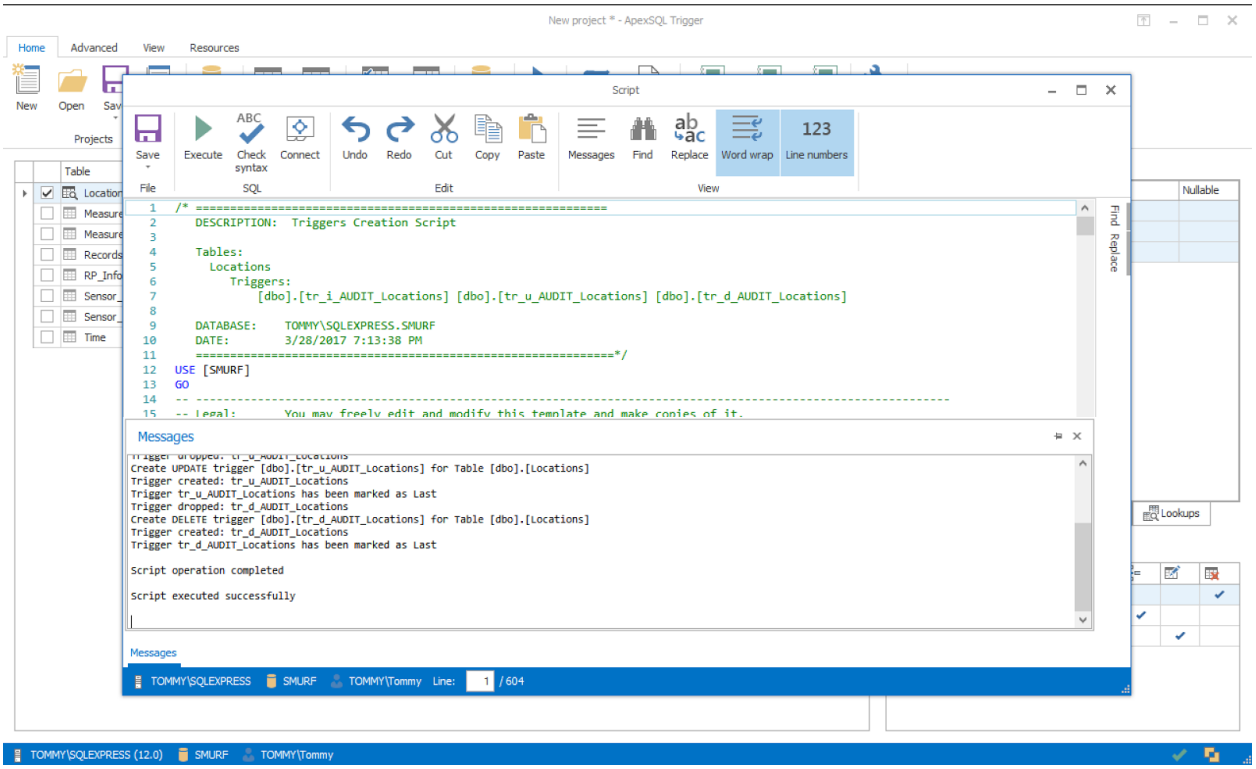


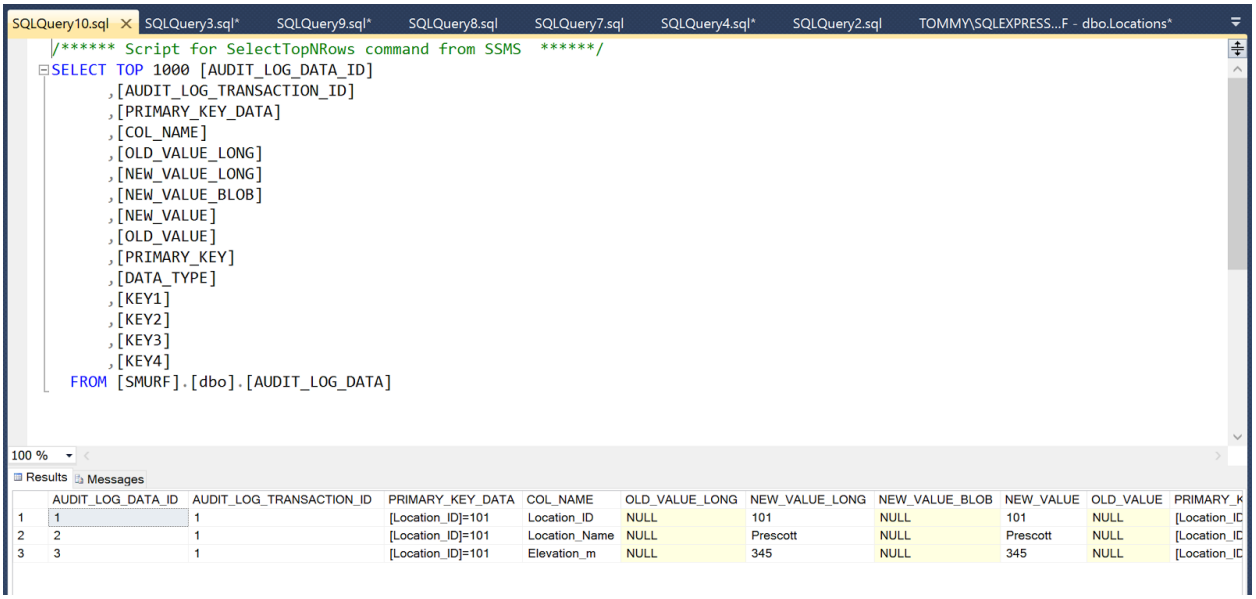
Cursor


SQLQuery8.sql - D...1F8HUF\Shaun (54))*
SQLQuery10.sql -...1F8HUF\Shaun (58))*
/** set the elevation of the location where
the No.1,2,3 RP into 360 **/
DECLARE update_elevation_cursor CURSOR FOR
SELECT Elevation_m
FROM Locations full join RP_Info
ON Locations.Location_ID = RP_Info.Location_ID
WHERE RP_ID IN (1,2,3)
OPEN update_elevation_cursor;
WHILE @@FETCH_STATUS = 0
BEGIN
FETCH FROM update_elevation_cursor;
UPDATE Locations
SET Elevation_m = 360
WHERE CURRENT OF update_elevation_cursor;
END
CLOSE update_elevation_cursor;
DEALLOCATE update_elevation_cursor;
GO
100 %
Messages
Command(s) completed successfully.


SQLQuery8.sql - D...1F8HUF\Shaun (54))*
SQLQuery10.sql -...1F8HUF\Shaun (58))*
/****** Script for SelectTopNRows command from SSMS ******/
SELECT TOP 1000 [Location_ID]
,[Location_Name]
,[Elevation_m]
FROM [SMURF(III)].[dbo].[Locations]
WHERE Elevation_m = 360
100 %
Results
Messages
Location_ID Location_Name Elevation_m
1 1 Langford 360
2 2 Rocca San Felice 360
3 3 Arles 360

Trigger: Insert to Logging Table





Rationale

The goal for this project is to find out how creating index on tables increases the time and speed of query operations. To be specific, we recorded the time it takes to run certain query without any index, then we add the index and run the query again. Ideally, the time it took should be shorter. For our view, we joined the Measurements_S1 table and Sensor_1 table using the ID for sensor 1 (S1_ID). We want to know indexing which key would help the speed of querying the most.

We first indexed the primary key of Measurements_S1, which is Measurement_ID_S1. After running the query again, the runtime did not improve (Shown in screenshot). We then tried to index the primary key of the Sensor_S1 table, which is S1_ID. It did not help either. At last, we decided to index the foreign key in Measurements_S1 table that is the primary key of the Sensor_S1 table, that is, S1_ID. The column is also the column used to join the two tables together. After running the query, the run time improved from 30ms to 2ms, which is a significant improvement.

IP Address / Local host