

2017

SMURF Database

Contents

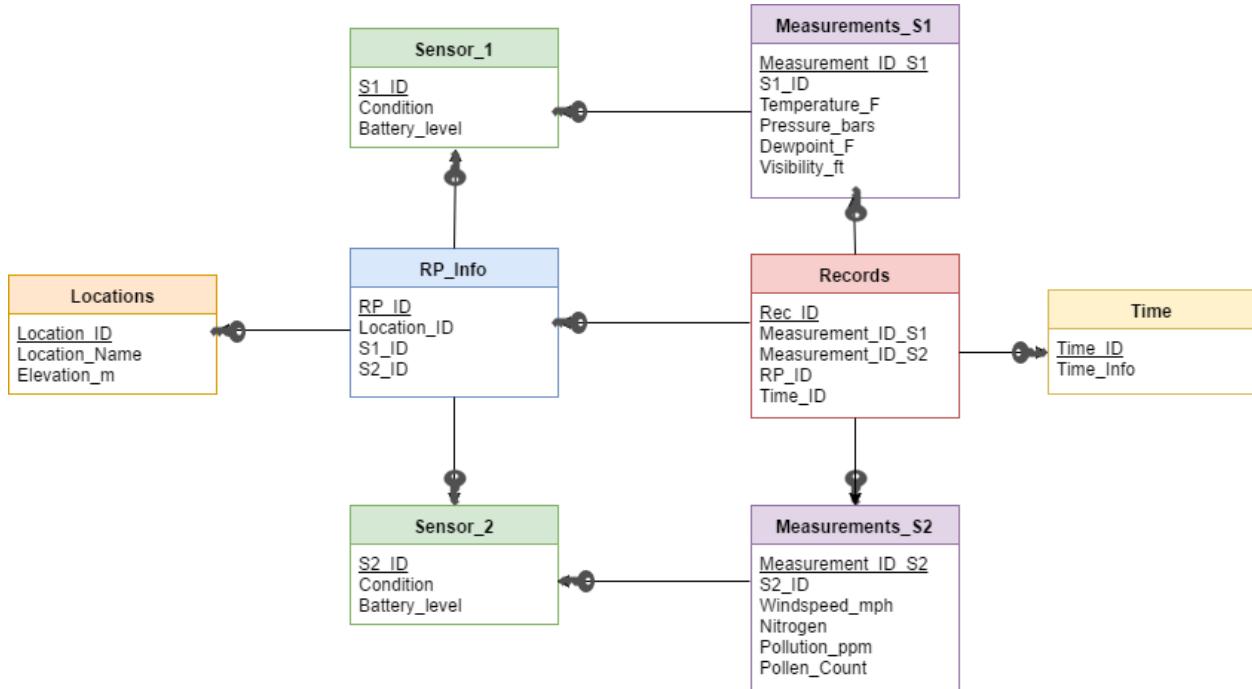
PART I: Design a Schema.....	4
Schema.....	4
Rationale.....	4
Database Diagram and Tables Screenshots.....	6
Database Diagram	6
Locations Table.....	7
Measurements_S1 Table.....	7
Measurements_S2 Table.....	8
Sensor_1 Table.....	9
Sensor_2 Table.....	9
RP_Info Table.....	10
Time Table.....	10
Records Table	10
Map Mockup	11
PART II: Integrity Constraint and Load	12
Insert Screenshots	12
Location Insert Query.....	12
Sensor_1 Insert Query	13
Measurements_S1 Insert Query	14

Sensor_2 Insert Query	15
Measurements_S2 Insert Query	16
RP_Info Insert Query.....	17
Time Insert Query	18
Records Insert Query	19
Constraints Screenshots and Rationale	20
Locations Constraints	20
Measurements_S1 Constraints.....	21
Sensor_1 Constraints	22
Measurements_S2 Constraints	23
Sensor_2 Constraints	24
PART III: Index and View Creation	25
Index and View Screenshots.....	25
Create View	25
Select and Join.....	26
Index on Measurements Primary Key	27
Index on Measurements Foreign Key.....	28
Index on Sensor Primary Key	29
PART IV: Transaction and Lock Management.....	30
Screenshots	30

Insert (before).....	30
Insert (After).....	30
Delete.....	31
Update.....	32
Select With Read-Only	33
Select Without Read-Only	33
Estimated Execution Plan	33
PART V: TSQL Development	36
Screenshots	36
Stored Procedure: Insert.....	36
Stored Procedure: Delete	37
Stored Procedure: Update	38
UDF Scalar Function.....	39
Cursor	40
Trigger: Insert to Logging Table.....	41
Rationale.....	42
IP Address / Local host	43

PART I: Design a Schema

Schema



Rationale

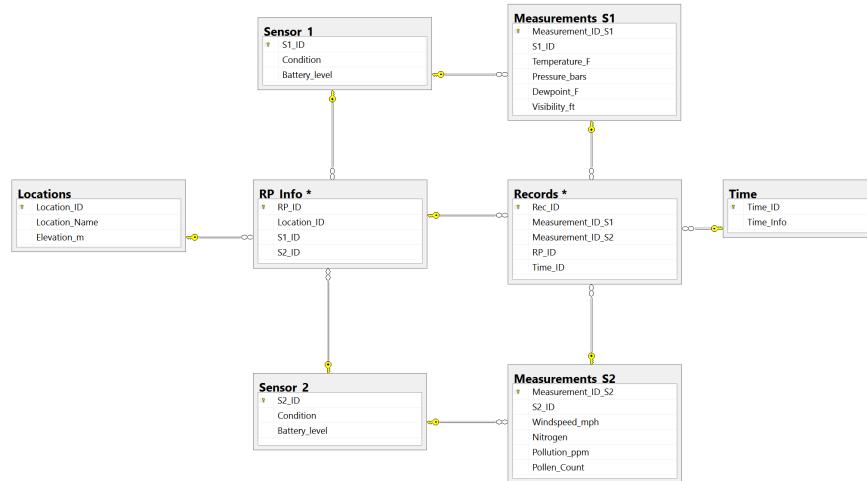
- **RP_Info:** “Raspberry Pi Information” will consist of the Raspberry Pi ID (PK), Location ID (FK), S1_ID (FK), and S2_ID (FK). The purpose of this table is to relate all the Raspberry Pi information together. Each RP will have two sensors on it, that each record different measurements, and each RP 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 Measurements_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

	Column Name	Data Type	Allow Nulls
PK	Location_ID	int	<input type="checkbox"/>
	Location_Name	varchar(20)	<input type="checkbox"/>
	Elevation_m	float	<input type="checkbox"/>
			<input type="checkbox"/>

Measurements_S1 Table

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

Measurements_S2 Table

Column Name	Data Type	Allow Nulls
Measurement_ID_S2	int	<input type="checkbox"/>
S2_ID	int	<input type="checkbox"/>
Windspeed_mph	int	<input checked="" type="checkbox"/>
Nitrogen	float	<input checked="" type="checkbox"/>
Pollution_ppm	float	<input checked="" type="checkbox"/>
Pollen_Count	float	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Sensor_1 Table

	Column Name	Data Type	Allow Nulls
PK	S1_ID	int	<input type="checkbox"/>
	Condition	varchar(10)	<input checked="" type="checkbox"/>
	Battery_level	varchar(10)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Sensor_2 Table

	Column Name	Data Type	Allow Nulls
PK	S2_ID	int	<input type="checkbox"/>
	Condition	varchar(10)	<input checked="" type="checkbox"/>
	Battery_level	varchar(10)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

RP_Info Table

	Column Name	Data Type	Allow Nulls
▶	RP_ID	int	<input type="checkbox"/>
	Location_ID	int	<input type="checkbox"/>
	S1_ID	int	<input checked="" type="checkbox"/>
	S2_ID	int	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

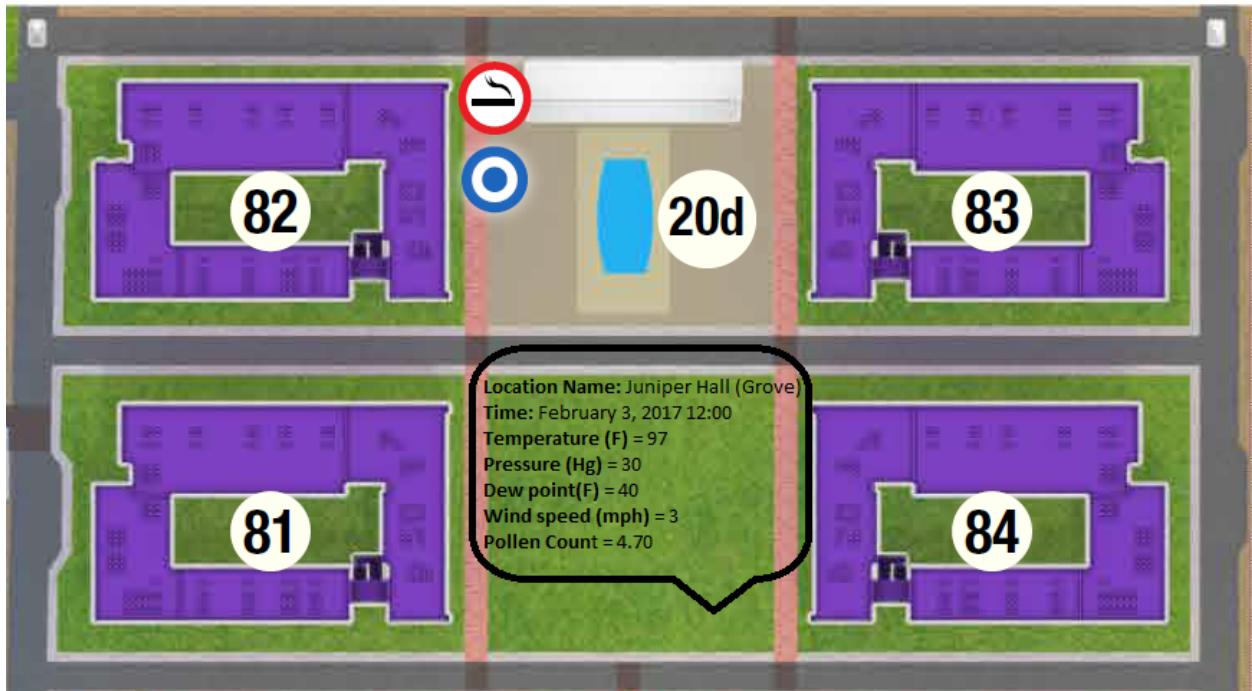
Time Table

	Column Name	Data Type	Allow Nulls
▶	Time_ID	int	<input type="checkbox"/>
	Time_Info	datetime	<input type="checkbox"/>
			<input type="checkbox"/>

Records Table

	Column Name	Data Type	Allow Nulls
▶	Rec_ID	int	<input type="checkbox"/>
	Measurement_ID_S1	int	<input checked="" type="checkbox"/>
	Measurement_ID_S2	int	<input checked="" type="checkbox"/>
	RP_ID	int	<input type="checkbox"/>
	Time_ID	int	<input type="checkbox"/>
			<input type="checkbox"/>

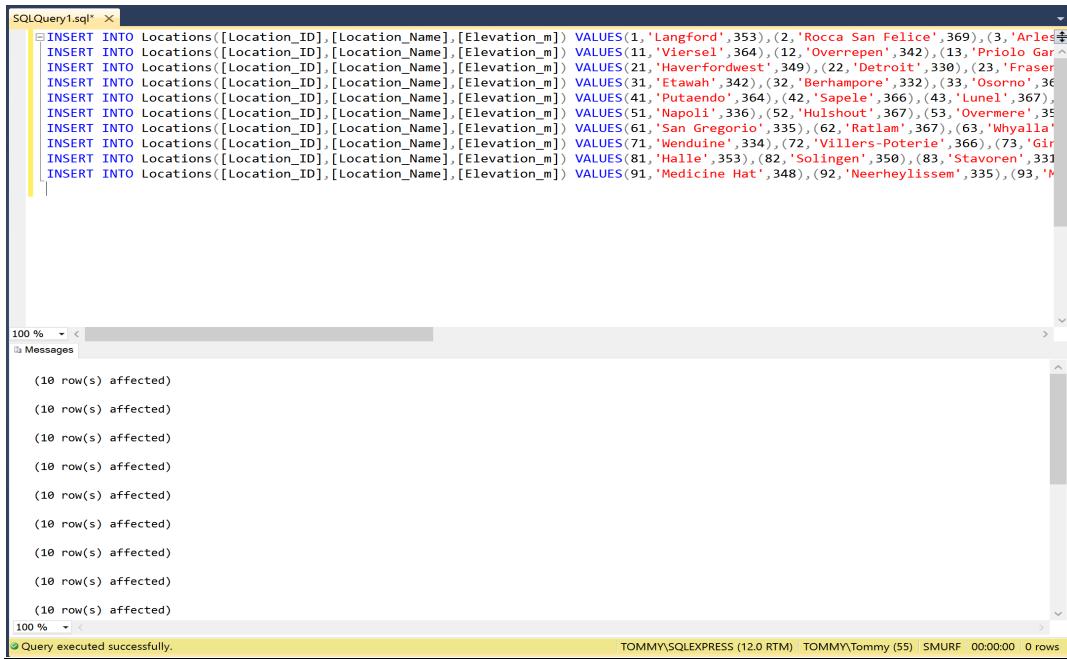
Map Mockup



PART II: Integrity Constraint and Load

Insert Screenshots

Location Insert Query



The screenshot shows an SQL query window titled "SQLQuery1.sql*". The query is an INSERT INTO statement for the "Locations" table, specifying columns [Location_ID], [Location_Name], and [Elevation_m]. The values are listed in groups of three rows each, spanning multiple lines. The query is executed successfully, with the status bar at the bottom indicating "Query executed successfully".

```

SQLQuery1.sql* ×
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(1, 'Langford', 353), (2, 'Roca San Felice', 369), (3, 'Arles', 369)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(11, 'Viersel', 364), (12, 'Overrepen', 342), (13, 'Priolo Gar', 364)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(21, 'Haverfordwest', 349), (22, 'Detroit', 330), (23, 'Fraser', 364)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(31, 'Etawah', 342), (32, 'Berhampore', 332), (33, 'Osorno', 364)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(41, 'Putaendo', 364), (42, 'Sapele', 366), (43, 'Lunel', 367)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(51, 'Napoli', 336), (52, 'Hulshout', 367), (53, 'Overmere', 351)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(61, 'San Gregorio', 335), (62, 'Ratlam', 367), (63, 'Whalla', 367)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(71, 'Wenduine', 334), (72, 'Villiers-Poterie', 366), (73, 'Gir', 367)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(81, 'Halle', 353), (82, 'Solingen', 350), (83, 'Stavoren', 331)
INSERT INTO Locations([Location_ID], [Location_Name], [Elevation_m]) VALUES(91, 'Medicine Hat', 348), (92, 'Neerheyilissem', 335), (93, 'M', 367)

100 % < >
Messages
(10 row(s) affected)
100 % < >
TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (55) | SMURF | 00:00:00 | 0 rows
Query executed successfully.

```

Sensor_1 Insert Query

The screenshot shows the SQL Server Management Studio interface. In the top bar, there are tabs for 'SQLQuery5.sql*', 'SQLQuery4.sql', 'TOMMY\SQLEXPRESS...F - dbo.Sensor_1', and 'SQLQuery3.sql*'. The main window displays an SQL script for inserting data into the 'Sensor_1' table. The script consists of multiple 'INSERT INTO' statements, each with a different row ID (e.g., 1, 11, 21, 31, 41, 51, 61, 71) and specific values for Condition and Battery_level. The 'Messages' pane at the bottom shows ten separate messages, each indicating that 10 rows were affected by the corresponding insert statement. A yellow status bar at the bottom right of the window indicates 'Query executed successfully.' and shows connection details: 'TOMMY\SQLEXPRESS (12.0 RTM)', 'TOMMY\Tommy (54)', 'SMURF', '00:00:00', and '0 rows'.

```
SQLQuery5.sql* X SQLQuery4.sql TOMMY\SQLEXPRESS...F - dbo.Sensor_1 SQLQuery3.sql*
[...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(1,'Bad','Low'),(2,'Bad','High'),(3,'Good','High'),(4,'Bad','Low') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(11,'Good','Medium'),(12,'Bad','High'),(13,'Bad','Medium'),(14,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(21,'Bad','Medium'),(22,'Good','Low'),(23,'Bad','Low'),(24,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(31,'Bad','Medium'),(32,'Good','Low'),(33,'Bad','High'),(34,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(41,'Good','High'),(42,'Bad','High'),(43,'Good','Low'),(44,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(51,'Good','Low'),(52,'Bad','Medium'),(53,'Bad','Medium'),(54,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(61,'Good','High'),(62,'Bad','High'),(63,'Bad','Medium'),(64,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(71,'Bad','High'),(72,'Good','Low'),(73,'Bad','Medium'),(74,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(81,'Good','High'),(82,'Good','Medium'),(83,'Bad','Medium'),(84,'Bad') [...] INSERT INTO Sensor_1([S1_ID],[Condition],[Battery_level]) VALUES(91,'Bad','Medium'),(92,'Good','High'),(93,'Good','Low'),(94,'Good','Medium')

100 % < > Messages
(10 row(s) affected)

100 % < > TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (54) | SMURF | 00:00:00 | 0 rows
Query executed successfully.
```

Measurements_S1 Insert Query

The screenshot shows a SQL Server Management Studio window with the following details:

- Query Editor:** The main pane displays a series of 10 `INSERT INTO Measurements_S1` statements, each inserting a row into the `dbo.Sensor_1` table. The statements are identical except for the values in the `VALUES` clause.
- Messages Pane:** Below the query editor, the messages pane shows 10 rows affected for each of the 10 insert statements, resulting in a total of 100 rows affected.
- Status Bar:** At the bottom, the status bar indicates "Query executed successfully." and shows connection information: TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (55) | SMURF | 00:00:00 | 0 rows.

```
SQLQuery5.sql*      SQLQuery4.sql      TOMMY\SQLEXPRESS...F - dbo.Sensor_1      SQLQuery3.sql* ×
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(1,1,11,11,11,11)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(11,11,21,21,21,21)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(21,21,31,31,31,31)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(31,31,41,41,41,41)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(41,41,51,51,51,51)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(51,51,61,61,61,61)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(61,61,71,71,71,71)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(71,71,81,81,81,81)
[ ] INSERT INTO Measurements_S1([Measurement_ID_S1],[S1_ID],[Temperature_F],[Pressure_Hg],[Dewpoint_F],[Visibility_ft]) VALUES(81,81,91,91,91,91)

100 % < > Messages
(10 row(s) affected)
100 % < < > TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (55) | SMURF | 00:00:00 | 0 rows
Query executed successfully.
```

Sensor_2 Insert Query

Measurements_S2 Insert Query

The screenshot shows a SQL Server Management Studio window with the following details:

- Query Editor:** The main pane displays a series of 10 `INSERT INTO Measurements_S2` statements, each inserting a row with specific values for columns like `S2_ID`, `Windspeed_mph`, `Nitrogen`, `Pollution_ppm`, and `Pollen_Count`.
- Messages Pane:** Below the query editor, the messages pane shows 10 rows affected for each insert statement, resulting in a total of 100 rows affected.
- Status Bar:** The bottom status bar indicates "Query executed successfully." and shows connection information: TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (58) | SMURF | 00:00:00 | 0 rows.

```
SQLQuery6.sql* X TOMMY\SQLEXPRES...Measurements_S2 SQLQuery5.sql* SQLQuery3.sql* SQLQuery4.sql*
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(1,1,26,11,11,11)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(11,11,11,21,21,21)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(21,21,21,31,31,31)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(31,31,31,41,41,41)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(41,41,41,51,51,51)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(51,51,51,61,61,61)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(61,61,61,71,71,71)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(71,71,71,81,81,81)
[ ] INSERT INTO Measurements_S2([Measurement_ID_S2],[S2_ID],[Windspeed_mph],[Nitrogen],[Pollution_ppm],[Pollen_Count]) VALUES(81,81,81,91,91,91)

100 % < > Messages
(10 row(s) affected)
100 % < < > 
Query executed successfully. TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (58) | SMURF | 00:00:00 | 0 rows
```

RP_Info Insert Query

The screenshot shows a SQL Server Management Studio window with the following details:

- Query Editor:** The title bar says "SQLQuery8.sql*". The tabs at the top include "TOMMY\SQLEXPRESS...RF - dbo.RP_Info", "SQLQuery6.sql*", "SQLQuery5.sql*", "SQLQuery3.sql*", and "SQLQuery4.sql*".
- Code Area:** The main area contains a large block of SQL INSERT statements for the "dbo.RP_Info" table. The statements are repeated 10 times, each inserting 10 rows of data.
- Messages Area:** Below the code, the "Messages" tab is selected. It displays 10 rows of output, each showing "(10 row(s) affected)".
- Status Bar:** At the bottom, it says "Query executed successfully." and shows connection information: "TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (59) | SMURF | 00:00:00 | 0 rows".

```

INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(1,1,1,1),(2,2,2,2),(3,3,3,3),(4,4,4,4),(5,5,5,5),(6,6,6,6),(7,7,7,7)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(11,11,11,11),(12,12,12,12),(13,13,13,13),(14,14,14,14),(15,15,15,15)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(21,21,21,21),(22,22,22,22),(23,23,23,23),(24,24,24,24),(25,25,25,25)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(31,31,31,31),(32,32,32,32),(33,33,33,33),(34,34,34,34),(35,35,35,35)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(41,41,41,41),(42,42,42,42),(43,43,43,43),(44,44,44,44),(45,45,45,45)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(51,51,51,51),(52,52,52,52),(53,53,53,53),(54,54,54,54),(55,55,55,55)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(61,61,61,61),(62,62,62,62),(63,63,63,63),(64,64,64,64),(65,65,65,65)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(71,71,71,71),(72,72,72,72),(73,73,73,73),(74,74,74,74),(75,75,75,75)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(81,81,81,81),(82,82,82,82),(83,83,83,83),(84,84,84,84),(85,85,85,85)
INSERT INTO RP_Info([RP_ID],[Location_ID],[S1_ID],[S2_ID]) VALUES(91,91,91,91),(92,92,92,92),(93,93,93,93),(94,94,94,94),(95,95,95,95)

```

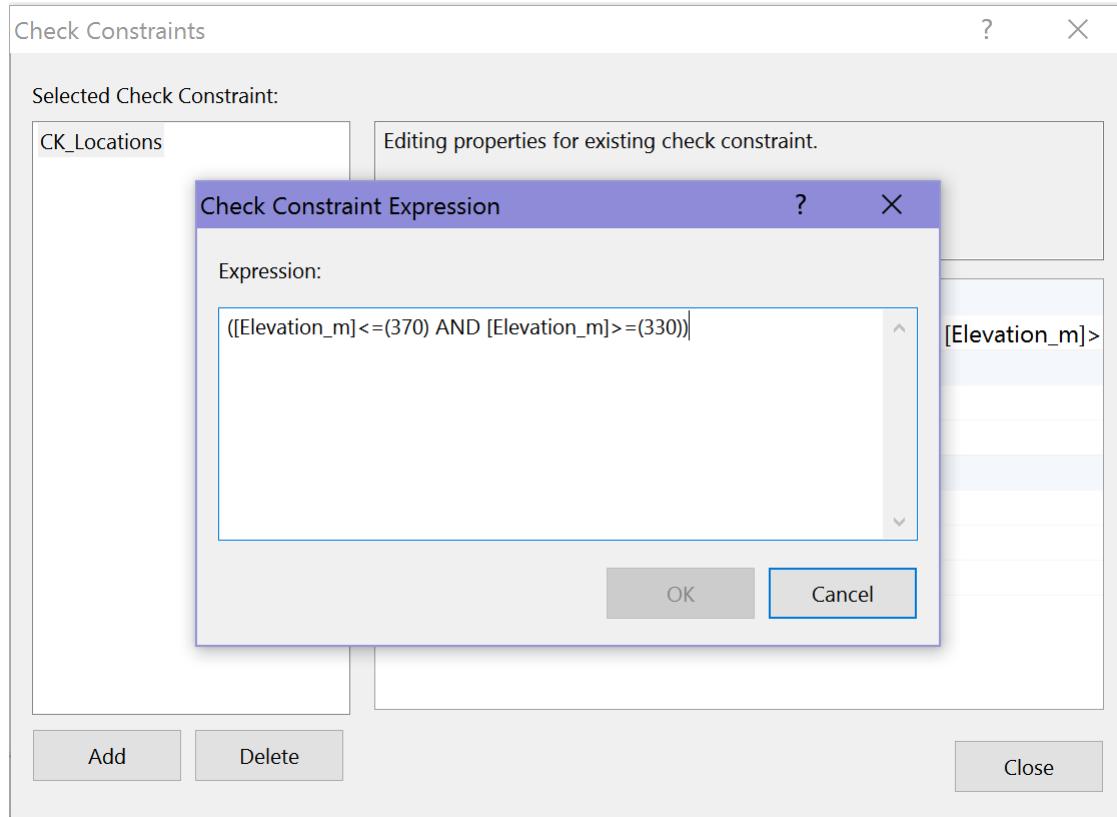
Time Insert Query

Records Insert Query

```
SQLQuery11.sql* SQLQuery9.sql* SQLQuery8.sql* TOMMY\SQLEXPRESS..RF - dbo.RP_Info SQLQuery6.sql* SQLQuery5.sql* SQLQuery3.sql* SQLQuery4.sql*  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(1,1,1,1,1),(2,2,2,2,2),(3,3,3,3,3)  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(11,11,11,11,11),(12,12,12,12,12),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(21,21,21,21,21),(22,22,22,22,22),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(31,31,31,31,31),(32,32,32,32,32),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(41,41,41,41,41),(42,42,42,42,42),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(51,51,51,51,51),(52,52,52,52,52),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(61,61,61,61,61),(62,62,62,62,62),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(71,71,71,71,71),(72,72,72,72,72),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(81,81,81,81,81),(82,82,82,82,82),  
[ ] INSERT INTO Records([Rec_ID],[Measurement_ID_S1],[Measurement_ID_S2],[RP_ID],[Time_ID]) VALUES(91,91,91,91,91),(92,92,92,92,92),  
  
100 % < Messages  
(10 row(s) affected)  
100 % <   
Query executed successfully.
```

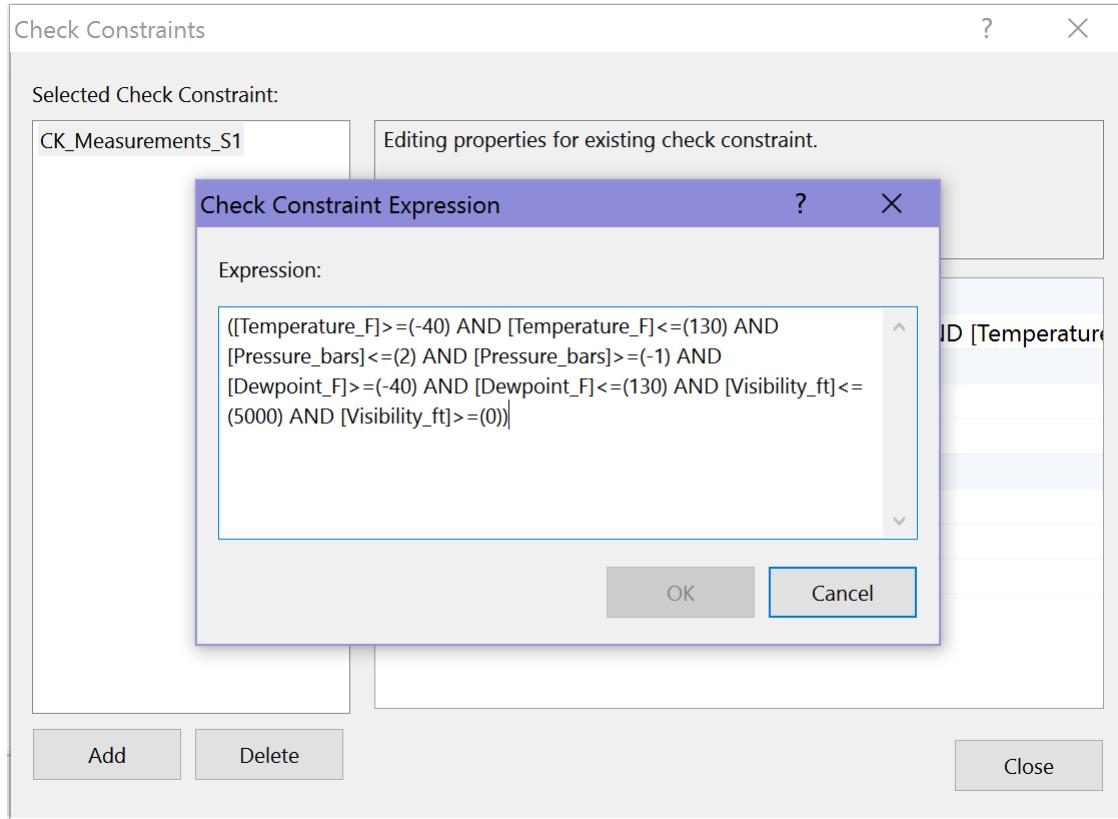
Constraints Screenshots and Rationale

Locations Constraints



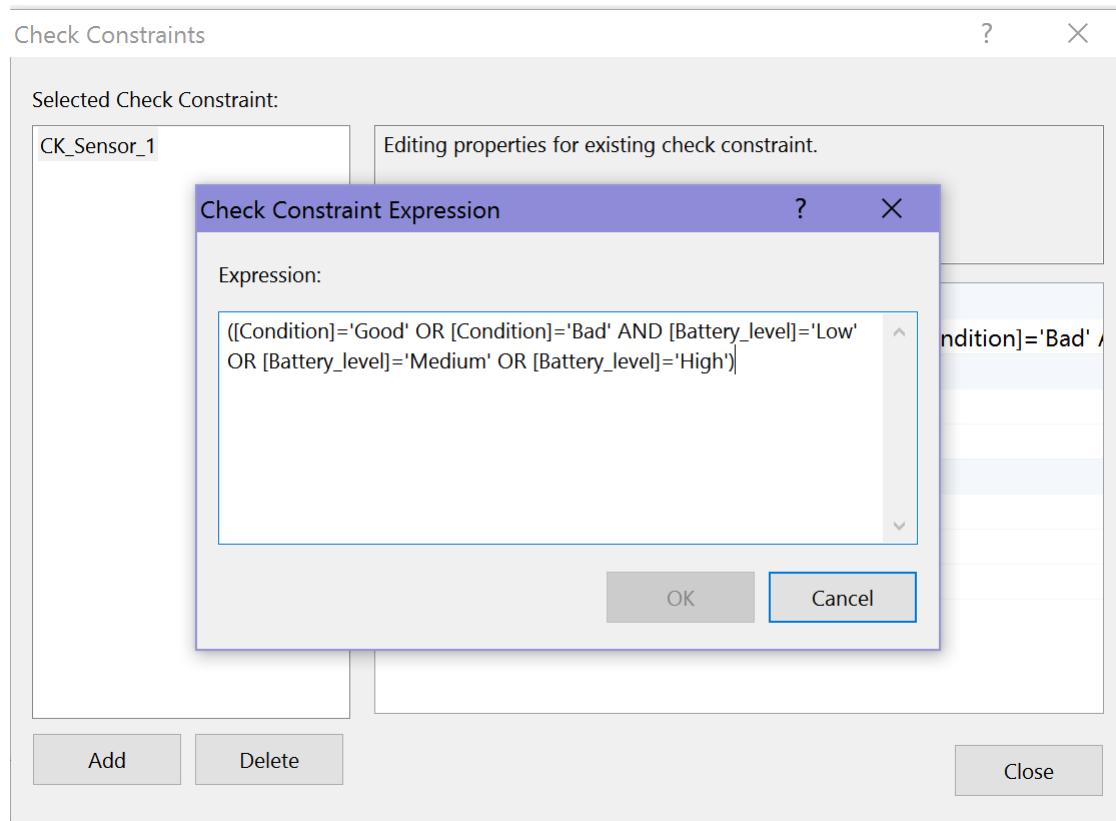
The reason for adding the constraints for elevation is to make sure that elevation is in reasonable range of value in unit of meters. When the value is not within the range, we can be certain that the data is wrong.

Measurements_S1 Constraints



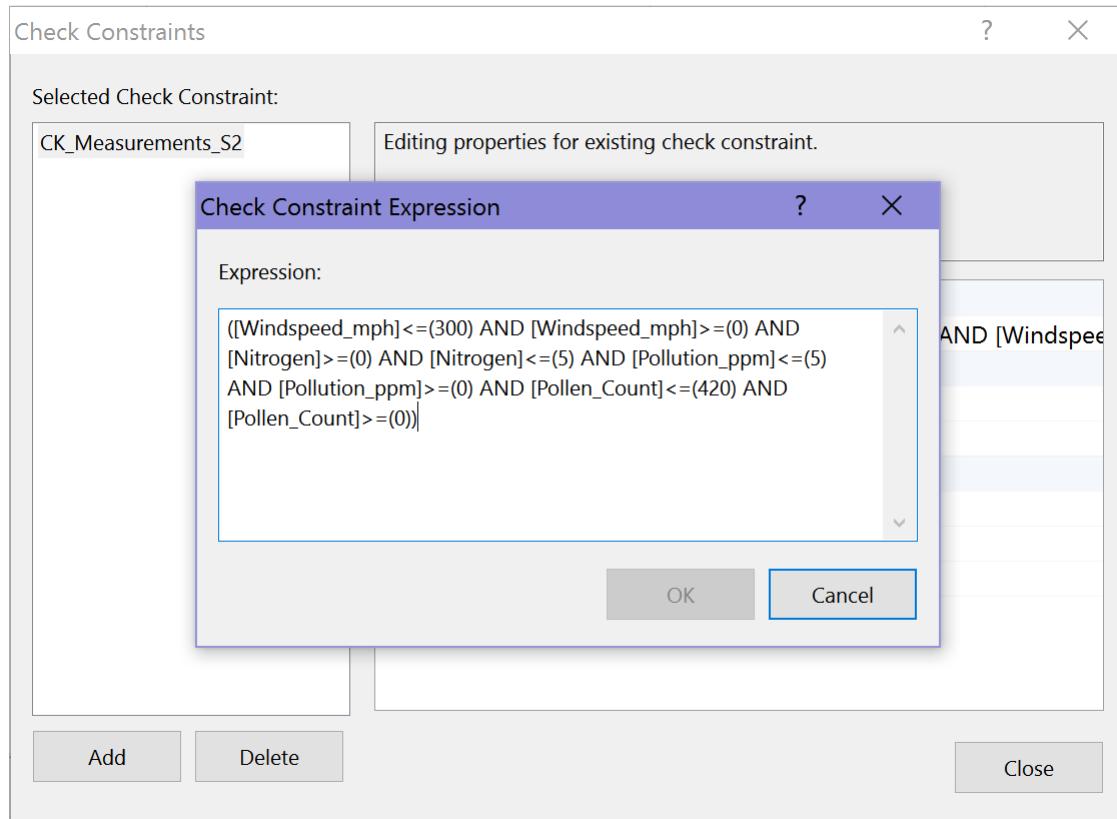
The reason for adding the constraints for temperature, pressure, dewpoint, and visibility is to make sure that these values are reasonable. When the data is off the range, we can be sure that the data is wrong. For example, the temperature in Phoenix would not exceed 130 or below -40 Fahrenheit, and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

Sensor_1 Constraints



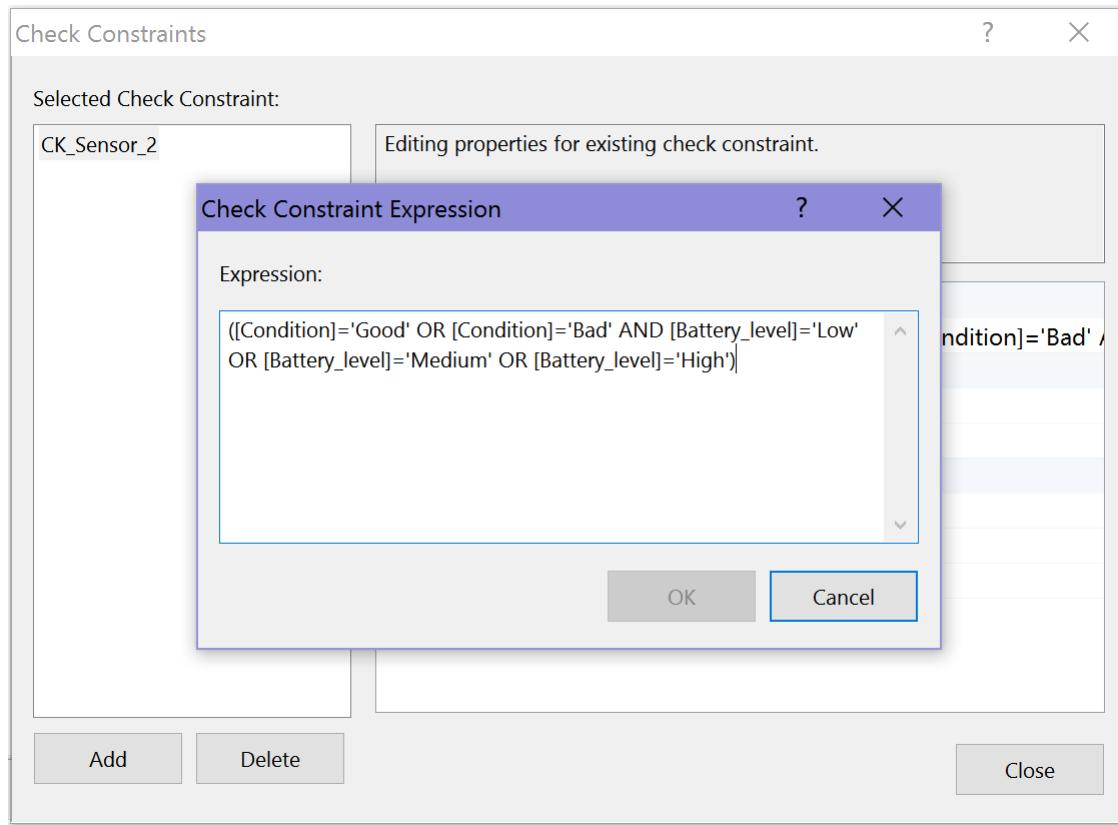
The reason to add the constraints for sensors is to make sure there are only two state for Condition: Good or Bad. Same logic goes to Battery level. There are only three fixed states for battery level: low, medium and high.

Measurements_S2 Constraints



The reason for adding the constraints for wind speed, nitrogen, pollution, and pollen count is to make sure that these values are 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 be 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

The screenshot shows a SQL query window titled "SQLQuery11.sql*". The query is:

```
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
```

The status bar at the bottom indicates "Command(s) completed successfully."

Select and Join

The screenshot shows a SQL Server Management Studio (SSMS) window. At the top, there are tabs for multiple queries: SQLQuery11.sql*, SQLQuery8.sql*, SQLQuery2.sql*, SQLQuery7.sql*, SQLQuery4.sql, and SQLQuery3.sql. The main pane displays the following T-SQL code:

```
SET STATISTICS TIME ON;
SELECT * FROM Measurements_S1
JOIN Sensor_1
    ON Measurements_S1.S1_ID = Sensor_1.S1_ID
SET STATISTICS TIME OFF;
```

Below the code, the results pane shows the output:

(100 row(s) affected)

SQL Server Execution Times:
CPU time = 32 ms, elapsed time = 30 ms.

In the status bar at the bottom, it says "Query executed successfully." and shows connection information: TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (58) | SMURF | 00:00:01 | 100 rows.

Index on Measurements Primary Key

The screenshot shows a SQL Server Management Studio (SSMS) window titled "SQLQuery11.sql*". The query window contains the following T-SQL code:

```
SET STATISTICS TIME ON;
SELECT * FROM Measurements_S1
JOIN Sensor_1
    ON Measurements_S1.S1_ID = Sensor_1.S1_ID
SET STATISTICS TIME OFF;
```

Below the query window, the results pane displays the output:

100 % < >
Results Messages

(100 row(s) affected)

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

Index on Measurements Foreign Key

The screenshot shows a SQL Server Management Studio (SSMS) interface. The title bar says "SQLQuery11.sql*". The main area contains the following SQL code:

```
SET STATISTICS TIME ON;
SELECT * FROM Measurements_S1
JOIN Sensor_1
    ON Measurements_S1.S1_ID = Sensor_1.S1_ID
SET STATISTICS TIME OFF;
```

Below the code, the results pane shows:

100 % < >
Results Messages

(100 row(s) affected)

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

Index on Sensor Primary Key

The screenshot shows a SQL Server Management Studio (SSMS) window titled "SQLQuery11.sql*". The query window contains the following T-SQL code:

```
SET STATISTICS TIME ON;
SELECT * FROM Measurements_S1
JOIN Sensor_1
    ON Measurements_S1.S1_ID = Sensor_1.S1_ID
SET STATISTICS TIME OFF;
```

Below the query window, there is a status bar with zoom controls (100 %), navigation arrows, and tabs for "Results" and "Messages". The results pane displays the output of the query:

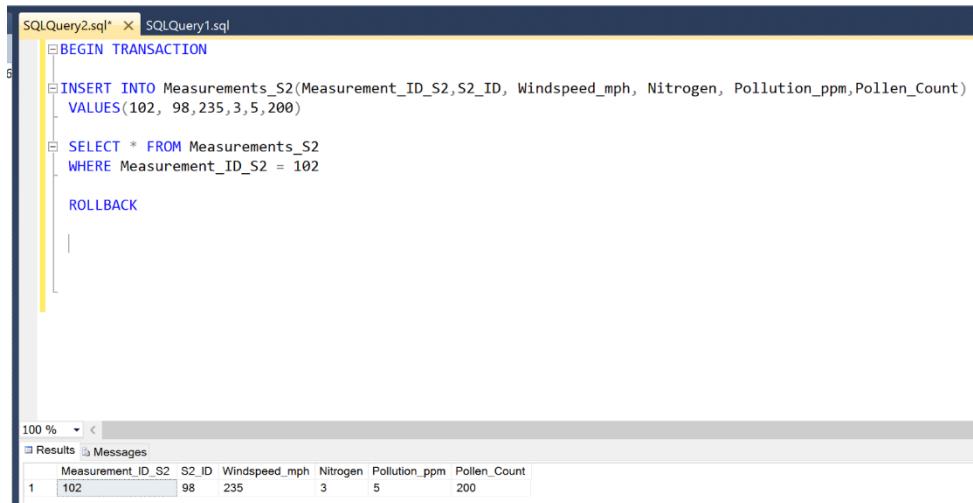
(100 row(s) affected)

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

PART IV: Transaction and Lock Management

Screenshots

Insert (before)



The screenshot shows the SQL Server Management Studio interface. A transaction is open with the following script:

```
BEGIN TRANSACTION
INSERT INTO Measurements_S2(Measurement_ID_S2,S2_ID, Windspeed_mph, Nitrogen, Pollution_ppm,Pollen_Count)
VALUES(102, 98,235,3,5,200)
SELECT * FROM Measurements_S2
WHERE Measurement_ID_S2 = 102
ROLLBACK
```

The results pane shows a single row inserted into the Measurements_S2 table:

	Measurement_ID_S2	S2_ID	Windspeed_mph	Nitrogen	Pollution_ppm	Pollen_Count
1	102	98	235	3	5	200

Insert (After)



The screenshot shows the SQL Server Management Studio interface. A query is run to select data from the Measurements_S2 table where Measurement_ID_S2 equals 102.

```
SELECT * FROM Measurements_S2
WHERE Measurement_ID_S2 = 102
```

The results pane shows the same single row that was inserted earlier:

	Measurement_ID_S2	S2_ID	Windspeed_mph	Nitrogen	Pollution_ppm	Pollen_Count
1	102	98	235	3	5	200

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 % <

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

```

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

COMMIT

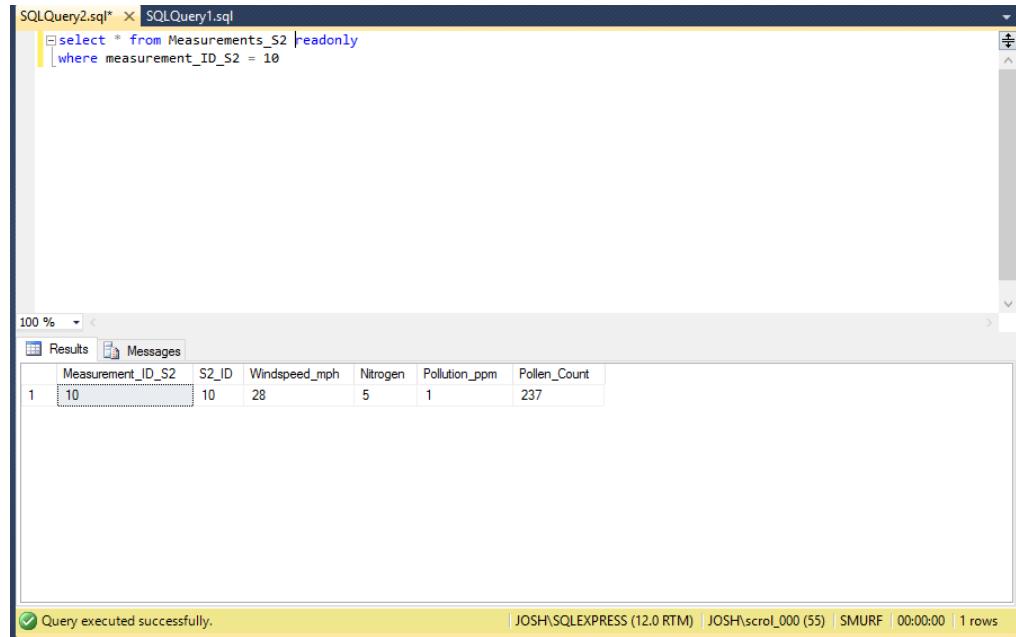
SELECT * FROM Measurements_S2
WHERE Measurement_ID_S2 = 100

```

100 % <

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

Select With Read-Only



The screenshot shows the SQL Server Management Studio interface. A query window titled 'SQLQuery2.sql* > SQLQuery1.sql' contains the following code:

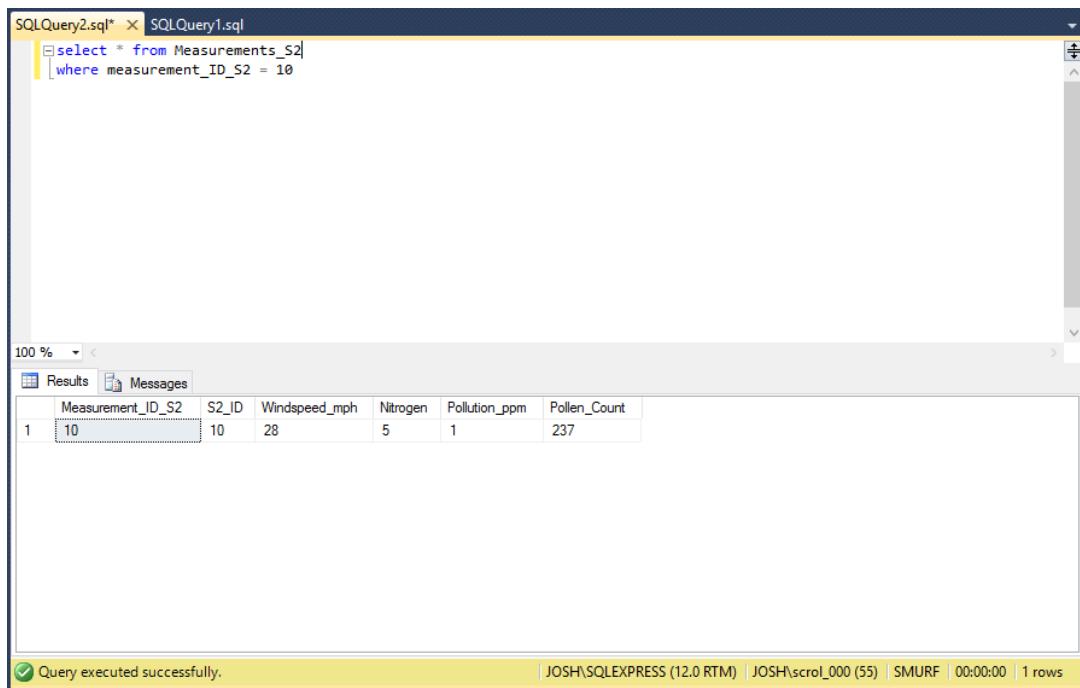
```
select * from Measurements_S2 | readonly  
| where measurement_ID_S2 = 10
```

The results pane shows a single row of data:

	Measurement_ID_S2	S2_ID	Windspeed_mph	Nitrogen	Pollution_ppm	Pollen_Count
1	10	28	5	1	237	

The status bar at the bottom indicates: 'Query executed successfully.' and 'JOSH\SQLEXPRESS (12.0 RTM) | JOSH\scrol_000 (55) | SMURF | 00:00:00 | 1 rows'

Select Without Read-Only



The screenshot shows the SQL Server Management Studio interface. A query window titled 'SQLQuery2.sql* > SQLQuery1.sql' contains the following code:

```
select * from Measurements_S2  
| where measurement_ID_S2 = 10
```

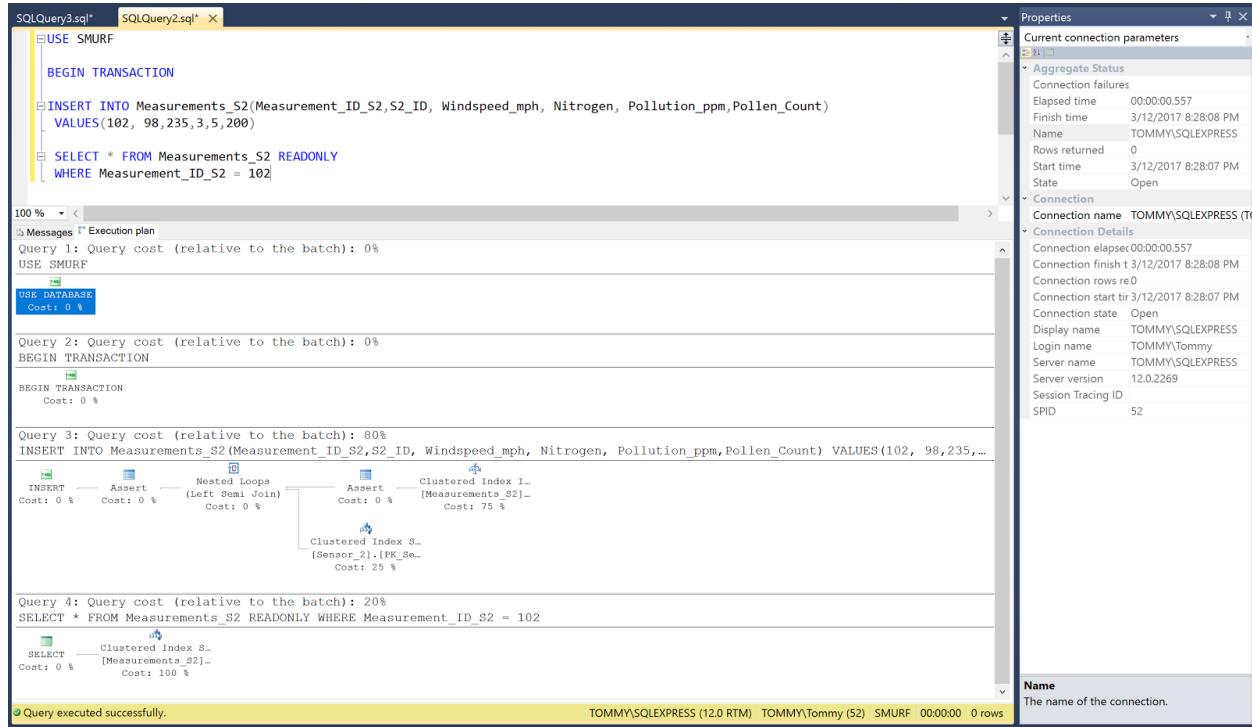
The results pane shows a single row of data:

	Measurement_ID_S2	S2_ID	Windspeed_mph	Nitrogen	Pollution_ppm	Pollen_Count
1	10	28	5	1	237	

The status bar at the bottom indicates: 'Query executed successfully.' and 'JOSH\SQLEXPRESS (12.0 RTM) | JOSH\scrol_000 (55) | SMURF | 00:00:00 | 1 rows'

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* X

```

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

Clustered Index U...
[Measurements_S2]...
Cost: 80 %

Clustered Index S...
[Sensor_2].[PK_Sen...]
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

Clustered Index S...
[Measurements_S2]...
Cost: 100 %

Query executed successfully.

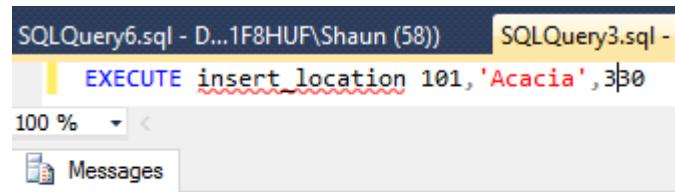
TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (52) | SMURF | 00:00:00 | 0 rows

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
```



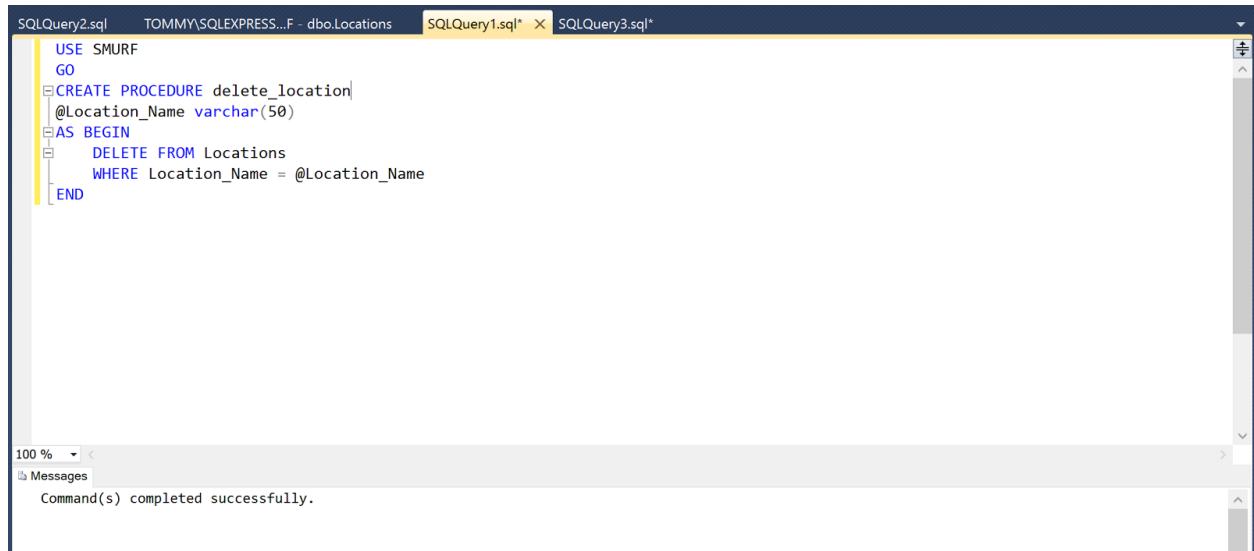
The screenshot shows the SQL Server Management Studio interface. The top title bar displays 'SQLQuery6.sql - D...1F8HUF\Shaun (58)' and 'SQLQuery3.sql -'. Below the title bar, a query window contains the following T-SQL code:

```
EXECUTE insert_location 101, 'Acacia', 330
```

The code is highlighted with syntax coloring. At the bottom of the window, there is a status bar showing '100 %' and a 'Messages' button.

Below the window, the output message '(1 row(s) affected)' is displayed.

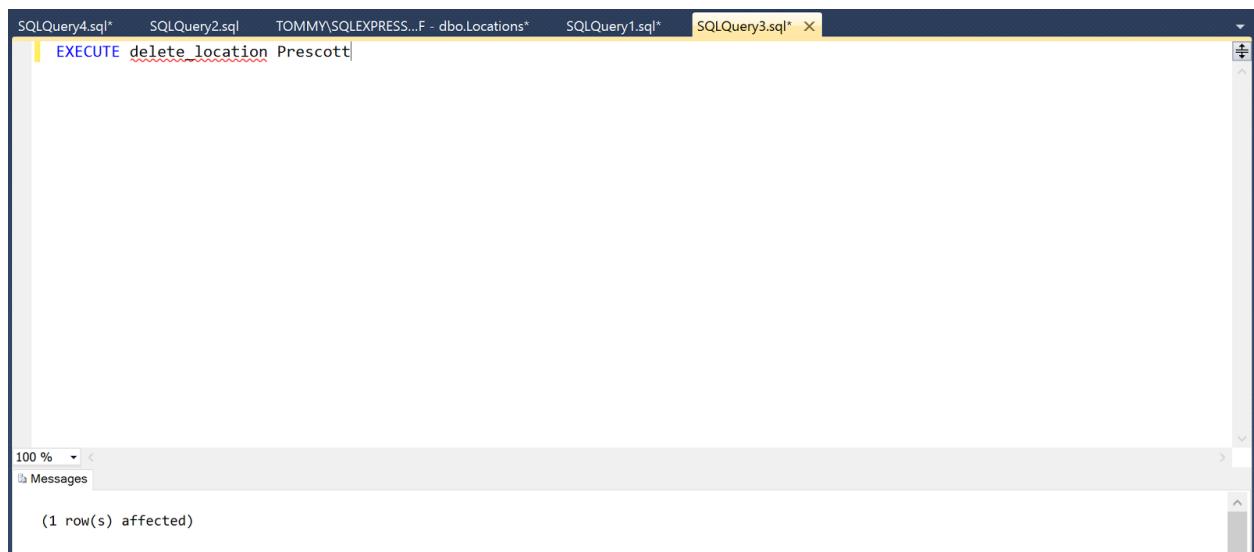
Stored Procedure: Delete



The screenshot shows the SQL Server Management Studio interface with four tabs at the top: SQLQuery2.sql, TOMMY\SQLEXPRESS...F - dbo.Locations, SQLQuery1.sql*, and SQLQuery3.sql*. The SQLQuery1 tab is active, displaying the following T-SQL code:

```
USE SMURF
GO
CREATE PROCEDURE delete_location
    @Location_Name varchar(50)
AS BEGIN
    DELETE FROM Locations
    WHERE Location_Name = @Location_Name
END
```

In the Messages pane at the bottom, it says "Command(s) completed successfully."



The screenshot shows the SQL Server Management Studio interface with five tabs at the top: SQLQuery4.sql*, SQLQuery2.sql, TOMMY\SQLEXPRESS...F - dbo.Locations*, SQLQuery1.sql*, and SQLQuery3.sql*. The SQLQuery4 tab is active, displaying the following T-SQL command:

```
EXECUTE delete_location Prescott
```

In the Messages pane at the bottom, it says "(1 row(s) affected)".

Stored Procedure: Update

The screenshot shows the SQL Server Management Studio interface. In the top tab bar, the active window is 'SQLQuery1.sql*'. The code pane contains the following T-SQL script:

```
USE SMURF
GO
CREATE PROCEDURE update_location
    @Elevation_m float,
    @Location_Name varchar(50)
AS BEGIN
    UPDATE Locations
        SET Elevation_m = @Elevation_m
        WHERE Location_Name = @Location_Name
END
```

In the 'Messages' pane below, it shows:

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

At the bottom of the screen, a status bar indicates:

```
100 %  <  Messages  TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (54) | SMURF | 00:00:00 | 0 rows
```

A yellow status bar at the bottom also displays:

```
Query executed successfully.
```

The screenshot shows the SQL Server Management Studio interface. The active window is 'SQLQuery1.sql*'. The code pane contains the following T-SQL command:

```
EXECUTE update_location 340, 'Langford'
```

In the 'Messages' pane below, it shows:

```
(1 row(s) affected)
```

At the bottom of the screen, a status bar indicates:

```
100 %  <  Messages  TOMMY\SQLEXPRESS (12.0 RTM) | TOMMY\Tommy (54) | SMURF | 00:00:00 | 0 rows
```

UDF Scalar Function

The screenshot shows a SQL query window titled "SQLQuery1.sql*". The code creates a scalar function named "dbo.GetLocationID" that takes a parameter "@Location_Name" and returns an integer. It selects the "Location_ID" from the "Locations" table where the "Location_Name" matches the parameter. The "GO" command is used to execute the script.

```
CREATE FUNCTION dbo.GetLocationID(@Location_Name varchar(50))
RETURNS INT
BEGIN
    RETURN (SELECT Location_ID FROM Locations
    WHERE Location_Name = @Location_Name)
END
GO
```

Messages pane: Command(s) completed successfully.

The screenshot shows a SQL query window titled "SQLQuery1.sql*". The code uses the "SMURF" database and executes a query that calls the "dbo.GetLocationID" function with the argument "Langford". It also includes a "FROM Locations" clause. The results pane displays a table with one column, showing the value 1 repeated 20 times.

```
USE SMURF
SELECT dbo.GetLocationID('Langford')
FROM Locations
```

Results pane:

(No column name)
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

Messages pane: Query executed successfully.

Cursor

SQLQuery8.sql - D...1F8HUF\Shaun (54))*

```

/* 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

	Location_ID	Location_Name	Elevation_m
1	1	Langford	360
2	2	Rocca San Felice	360
3	3	Arles	360

Messages

Trigger: Insert to Logging Table

New project * - ApexSQL Trigger

Home Advanced View Resources

Save Execute ABC Connect Undo Redo Cut Copy Paste Messages Find Replace Word wrap Line numbers

Table Location

```

1  /* ===== */
2  DESCRIPTION: Triggers Creation Script
3
4  Tables:
5    Locations
6    Triggers:
7      [dbo].[tr_i_AUDIT_Locations] [dbo].[tr_u_AUDIT_Locations] [dbo].[tr_d_AUDIT_Locations]
8
9  DATABASE: TOMMY\SQLEXPRESS.SMURF
10 DATE: 3/28/2017 7:13:38 PM
11 =====*/
12 USE [SMURF]
13 GO
14 --
15 -- Legal: You may freely edit and modify this template and make copies of it.

```

Messages

```

Trigger:tr_u_AUDIT_Locations
Create UPDATE trigger [dbo].[tr_u_AUDIT_Locations] for Table [dbo].[Locations]
Trigger created: tr_u_AUDIT_Locations
Trigger tr_u_AUDIT_Locations has been marked as Last
Trigger dropped: tr_d_AUDIT_Locations
Create DELETE trigger [dbo].[tr_d_AUDIT_Locations] for Table [dbo].[Locations]
Trigger created: tr_d_AUDIT_Locations
Trigger tr_d_AUDIT_Locations has been marked as Last

Script operation completed
Script executed successfully

```

Nullable

Find Replace

Lookups

TOMMY\SQLEXPRESS (12.0) SMURF TOMMY\Tommy Line: 1 / 604

SQLQuery10.sql X SQLQuery3.sql* SQLQuery9.sql* SQLQuery8.sql SQLQuery7.sql SQLQuery4.sql* SQLQuery2.sql TOMMY\SQLEXPRESS...F - dbo_LOCATIONS

```

***** Script for SelectTopNRows command from SSMS *****
SELECT TOP 1000 [AUDIT_LOG_DATA_ID]
,[AUDIT_LOG_TRANSACTION_ID]
,[PRIMARY_KEY_DATA]
,[COL_NAME]
,[OLD_VALUE_LONG]
,[NEW_VALUE_LONG]
,[NEW_VALUE_BLOB]
,[NEW_VALUE]
,[OLD_VALUE]
,[PRIMARY_KEY]
,[DATA_TYPE]
,[KEY1]
,[KEY2]
,[KEY3]
,[KEY4]
FROM [SMURF].[dbo].[AUDIT_LOG_DATA]

```

100 %

Results Messages

AUDIT_LOG_DATA_ID	AUDIT_LOG_TRANSACTION_ID	PRIMARY_KEY_DATA	COL_NAME	OLD_VALUE_LONG	NEW_VALUE_LONG	NEW_VALUE_BLOB	NEW_VALUE	OLD_VALUE	PRIMARY_K
1	1	[Location_ID]=101	Location_ID	NULL	101	NULL	101	NULL	[Location_ID]
2	1	[Location_ID]=101	Location_Name	NULL	Prescott	NULL	Prescott	NULL	[Location_ID]
3	1	[Location_ID]=101	Elevation_m	NULL	345	NULL	345	NULL	[Location_ID]

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

```

C:\ Command Prompt
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . .

Ethernet adapter Ethernet 36:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . .

Ethernet adapter Ethernet 37:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . .

Ethernet adapter VirtualBox Host-Only Network:
Connection-specific DNS Suffix . .
Link-local IPv6 Address . . . . . : fe80::75f9:3d7c:8f2d:bac%10
IPv4 Address. . . . . : 192.168.56.1
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :

Wireless LAN adapter Local Area Connection* 4:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . .

Wireless LAN adapter Wi-Fi 2:
Connection-specific DNS Suffix . .
Link-local IPv6 Address . . . . . : fe80::746a:abc0:1234:edfd%4
IPv4 Address. . . . . : 172.24.52.239
Subnet Mask . . . . . : 255.255.192.0
Default Gateway . . . . . : 172.24.0.1

Ethernet adapter Bluetooth Network Connection:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . .

Tunnel adapter isatap.{2F085973-FDF4-4524-8EC0-C9FE95EB5BC1}:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . .

Tunnel adapter Local Area Connection* 2:
Connection-specific DNS Suffix . .
IPv6 Address. . . . . : 2001:0:9d38:6abd:cb9:d893:7667:4610
Link-local IPv6 Address . . . . . : fe80::cb9:d893:7667:4610%13
Default Gateway . . . . . : ::

Tunnel adapter isatap.{0FA46B8D-A19C-4984-9863-5F2CF178140C}:
Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . . .

C:\Users\Tommy>cd C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL
C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL>

```