Term Project Iteration 5

Phillip Escandon
escandon@bu.edu

Project Direction / Overview

Aerospace Camera System Error Logs, Maintenance Logs, Configuration Files-

During the process of building and integrating a camera system and all of its subparts, many log files are captured but not widely used. When called upon to review the logs, only a few team members have the background and historical knowledge to review and decipher any issues as well as suggest a course of action to rectify these issues.

My proposed database will contain log files and the associated describing data for each of these camera systems. I have collected approximately 9000 of these logs dating back to 2012. My goal would be a database that could quickly create a report(s) to:

1. Give a historical background and timeline of the camera, including SW versions, Configuration of the camera system when it was delivered to a customer and common failures observed during testing.
2. Give a historical background and normalized boot sequence for a given camera system.
3. Give a historical background of *failures* and some basic configuration and state that the camera was in during a failure.

Some of these items may seem trivial,but the collection of these logs and *quickly* deciphering them has always been an issue.

This database could potentially be used by four (4) teams.

The data to be captured and used in the database are the logs that are particular to each product.
Three particular logs will be used:

Errorlog.log
Maintenance.log
Sensor configuration

1. Log Files

Error Log

In an abstract framework, the logs can be considered to consist of three sections:

- **Boot Section -** describes in terse, software terms, the boot sequence of the product.
- **Start Up Built In Test (SBIT)**- Once boot is completed, the camera synchronizes with a GPS clock and a built in test is conducted. It describes the status of the various subsystems in the product.
- **Plan-** describes the tasks allocated to the product and records the user interaction with the product until the product is shut down.

Hundreds of these logs are collected for each system. Each system has a distinct **SENSOR ID**. The logs are always called 'errorlog.log' or 'maint.log'.   Each errorlog will contain the Boot and SBIT section, but not necessarily the mission section. The maintenance log will contain Start times, end time, PlanID and SBIT Status.

Once the product is fielded for a customer and used, the Plan section will contain valid recorded data.

Current usage of these logs is minimal. Thousands of these logs exist, but they are rarely examined or mined for details and analysis. A shortened version of this log exists as a 'maintenance log' and is primarily used by the Field Service Team to quickly debug issues.

Maintenance Log

This is a simple version of the error log that only contains Plan ID, SBIT failures, startup and shutdown times. Everything in this log can be found in the Error log.

Configuration File

Each sensor has it’s very own configuration file. This contains values that were tuned during the integration phase as well as the final focus values for the various field of views.

Interest

I have seen and used these logs but feel that they are being overlooked. Sloppy warehousing, incomplete datasets and timeframes, no real direction and no plan to use these logs or versions of these logs in upcoming projects.

For one particular vexing issue, I was given a small dataset of roughly 50 files- a list of keywords and general instruction as to see if I could figure out what was happening.

After some initial parsing and cleaning, I was still left with smaller and still very obtuse files. I could see what was happening, and could verbally explain but the team required context. They required a story to go along with this data. A parsing of the files did not provide the needed story and background.

The general questions that should be answered are:

- What Sensor had an issue?
  Sensor ID
- When did it have an issue?
  Date
- What is the SW load for each component?
  SW Versioning information
- Did it pass SBIT?
  Test Report - state of the system before mission
- What was tasked in a mission?
  What was planned?
- What failed?
  What planned task did not work?
- What passed?
  What planned task did work?
- What keys were pressed on the system?
  What buttons did the user press during operation?

1.1 Use Cases

Integration Team

The database could be used to identify issues with the product prior to release and shipment of the product. This team also captures data related to tests that are conducted in the 3 month build cycle, as well as a final configuration file.

Use Case: An Integration Engineer completes her test and is ready to update the database with her configuration file. The file will be added to the database and a report will be generated that compares each field with the corresponding fields from previous sensor ids. If each value is within +- 3 standard deviations from the mean of all previous values, then it is accepted. If a value exceeds 3 STD from the mean, the value is flagged as an outlier for further analysis. This will occur for each of the 6 tests completed: Controls, Line of Sight, Thermal, Auxiliary, Imaging, Functional Baseline.

Customer Engineering / Chief Engineers Group

By keying and reflecting on a complete report, or series of reports generated by the database, a coherent story that describes successes and failures for individual systems can be generated. An engineer assigned to support country X, could quickly find all sensors delivered to X, SW versioning information and historical data. Decisions as to the exact SW load delivered and any required updates could be tracked and seen using this database. New log files could be inserted and reports generated to provide actionable direction to correct issues.

Use Case: A Chief Engineer (CE) for Greece must gather a detailed report of all systems delivered to that country. The cameras were delivered 8 years ago.

The CE would search the database for ‘Greece’ and find 8 sensors delivered throughout a two year period. A report would indicate the IDs of the sensors delivered as well as software builds.

Using the only customer logs, a rudimentary Elapsed Time of usage could be calculated for each sensor.

Product Support / Quality Control

Failures of the product could be captured in the database in the form of a test report generated from the SBIT section of the errorlog.log. The database could also be used as a final sell-off deliverable as proof that the product has indeed passed all required tests and document corrective actions that rectified failed tests. The serial numbers and of major components could be captured and used by the Production and Quality team.

Use Case: The Quality team has just informed the customer that the Functional Baseline test has passed and the system will be ready for final acceptance testing.

The quality engineer would search for ‘Greece 2’ and find the ‘status’ report indicating the sensor ID, tests conducted, Built in Test status and software versions of the AIB, RMS, SCU and GEDI computer boards. This report would go into the sell-off documentation stating that the system has been tested and has passed all requirements.

1.2 Fields

Field	What is Stores	Why it's needed
Error Log.Sensor ID Boot.SensorID SBIT.SensorID Plan.SensorID Camera.SensorID	Stores the ID of the individual camera	Each camera system is unique and has this unique identifier.
Boot.TimeDate SBIT.TimeDate Plan.TimeDate IntegrationTest.TimeDate	Time and Date of the camera operation	This is the time and date field from the logs in Unix Epoch time. This will be converted to GMT time for the database.
Plan.PlanID	Identifies the specific planned use and configuration of the camera	This identifies the specific plan that was used during the operation. This is not a unique field.
Boot.RMSVersion Boot.SCUVersion Boot.GEDIVersion Boot.AIBVersion	Software version of the four computer boards	This is the SW build version for each computer board. Usually does not change but is very helpful for historical reports.
Boot.Message SBIT.Message Plan.Message	Message string from the logs	String message from the error log. This usually gives the details of pass / fail actions.
SBIT.Status	Boolean for testing purposes	This will give us a quick overview of the status of a test / retest sequence.
Boot.TimeSync	The time that the GPS and computer times were synchronized	This gives an indication of system boots. If more than one Time Synchronization is seen, this could indicate a problem where the camera unintentionally restarted.
ErrorLog.filename Boot.filename SBIT.filename Plan.filename	Unique filename created from the errorlog	The errorlog must be renamed as a unique entity. This unique filename will be used to identify the parent of the three (Boot, SBIT, Plan) instances generated.
Boot.message	Variable length message from the	Distinct message from the errorlog

SBIT.message Plan.message	errorlog section	section that is used to give greater context.

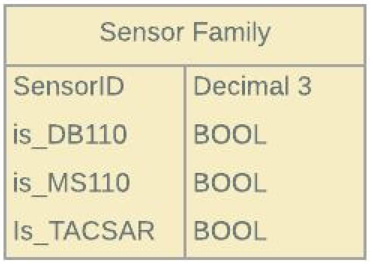
Field	What is stores	Why it's needed
RollResolver.Devicepub.offset	Offset for the Roll Resolver on camera	Historical Data - needed for comparison to other systems
RollResolverConfig.offset	Resolver configuration offset	Historical Data - needed for comparison to other systems
PitchResolver.Devicepub.offset	Offset for the Pitch Resolver on camera	Historical Data - needed for comparison to other systems
PitchResolverConfig.offset	Resolver configuration offset	Historical Data - needed for comparison to other systems
BS.nf1bm0 ConfigurationFile.Filter1	Backscan Z transform filter value	These values are constantly looked up and reused. Historical Data - needed for comparison to other systems
ConfigurationFile.Filter2 BS.nf1bm1	Backscan Z transform filter value	These values are constantly looked up and reused. Historical Data - needed for comparison to other systems
ConfigurationFile.Filter3 BS.nf1bm2	Backscan Z transform filter value	These values are constantly looked up and reused. Historical Data - needed for comparison to other systems
ConfigurationFile.Filter4 BS.df1bm0	Backscan Z transform filter value	These values are constantly looked up and reused. Historical Data - needed for comparison to other systems
ConfigurationFile.Filter5 BS.df1bm1	Backscan Z transform filter value	These values are constantly looked up and reused.

		Historical Data - needed for comparison to other systems.
ConfigurationFile.Filter6 BS.df1bm2	Backscan Z transform filter value	Historical Data - needed for comparison to other systems

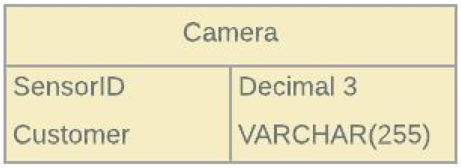
1.3 Summary and Reflection

2.1 Structural Database Rules for Entities

A Sensor Family is made up of Cameras
Camera belongs to a Sensor Family

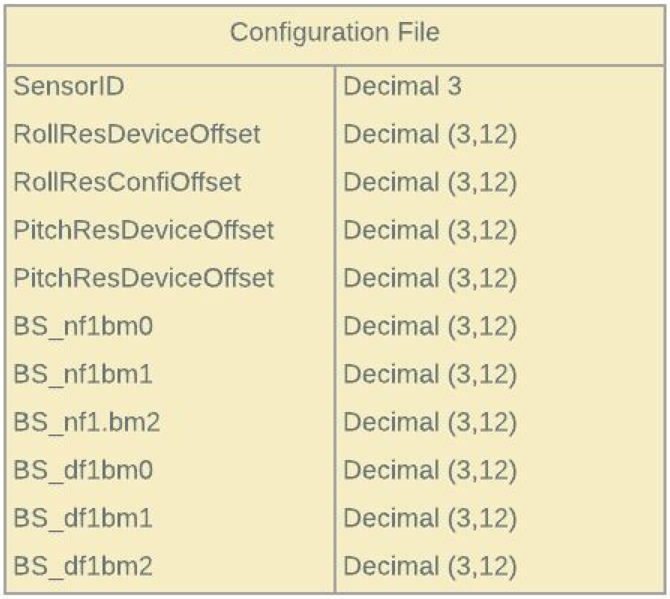


Sensor Family	
SensorID	Decimal 3
is_DB110	BOOL
is_MS110	BOOL
Is_TACSAR	BOOL

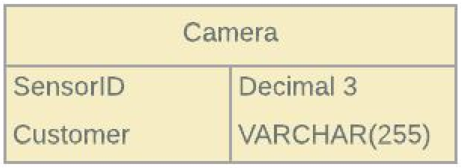


Camera	
SensorID	Decimal 3
Customer	VARCHAR(255)

A Camera may contain one configuration file.
A Configuration file belongs to one camera

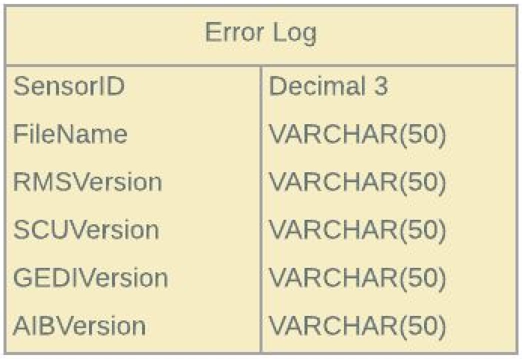


Configuration File	
SensorID	Decimal 3
RollResDeviceOffset	Decimal (3,12)
RollResConfiOffset	Decimal (3,12)
PitchResDeviceOffset	Decimal (3,12)
PitchResDeviceOffset	Decimal (3,12)
BS_nf1bm0	Decimal (3,12)
BS_nf1bm1	Decimal (3,12)
BS_nf1.bm2	Decimal (3,12)
BS_df1bm0	Decimal (3,12)
BS_df1bm1	Decimal (3,12)
BS_df1bm2	Decimal (3,12)

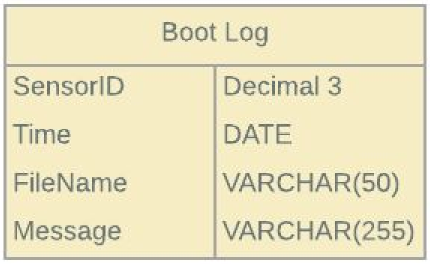


Camera	
SensorID	Decimal 3
Customer	VARCHAR(255)

An Error Log may generates one or more Boot Logs
A Boot Log is generated by one Error Log



Error Log	
SensorID	Decimal 3
FileName	VARCHAR(50)
RMSVersion	VARCHAR(50)
SCUVersion	VARCHAR(50)
GEDIVersion	VARCHAR(50)
AIBVersion	VARCHAR(50)



Boot Log	
SensorID	Decimal 3
Time	DATE
FileName	VARCHAR(50)
Message	VARCHAR(255)

An Error Log may generates one or more SBIT Logs
An SBIT Log is generated by one Error Log

Error Log	
SensorID	Decimal 3
FileName	VARCHAR(50)
RMSVersion	VARCHAR(50)
SCUVersion	VARCHAR(50)
GEDIVersion	VARCHAR(50)
AIBVersion	VARCHAR(50)

SBIT Log	
SensorID	Decimal 3
Time	DATE
FileName	VARCHAR(50)
Test	VARCHAR(12)
Status	VARCHAR(12)
Message	VARCHAR(255)

An Error log may generates one Plan Log
A Plan Log is generated by one Error Log

Error Log	
SensorID	Decimal 3
FileName	VARCHAR(50)
RMSVersion	VARCHAR(50)
SCUVersion	VARCHAR(50)
GEDIVersion	VARCHAR(50)
AIBVersion	VARCHAR(50)

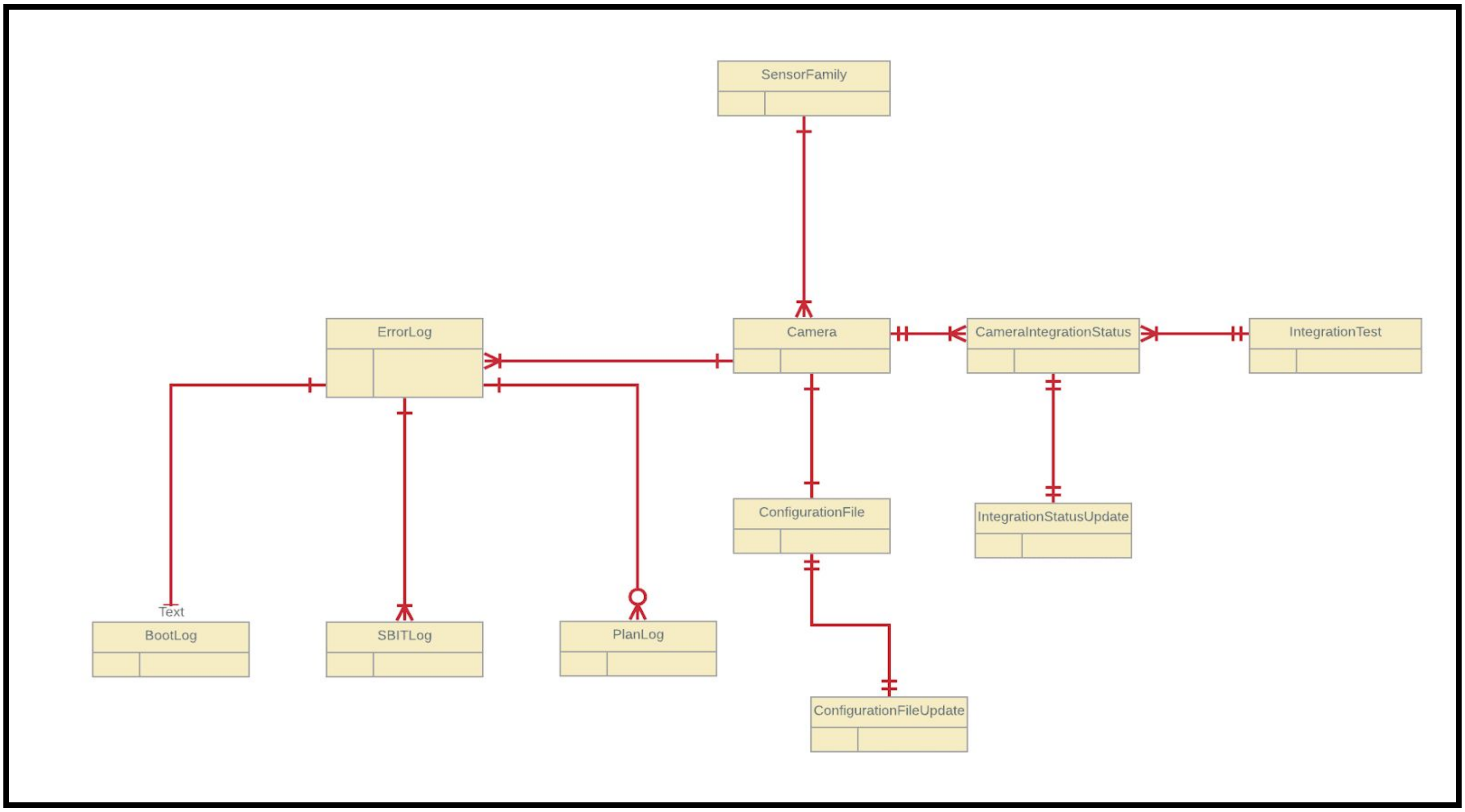
Plan Log	
SensorID	Decimal 3
Time	DATE
FileName	VARCHAR(50)
Tasks	VARCHAR(50)
Datalink	BOOL
PlanID	VARCHAR(50)
Message	VARCHAR(255)

Integration Test is composed of many subtests.
One subtest belongs to one Integration Test.

Integration Tests	
SensorID	Decimal 3
TestID	DECIMAL 3
Completion_Percent	INT
DataLocation	VARCHAR(255)

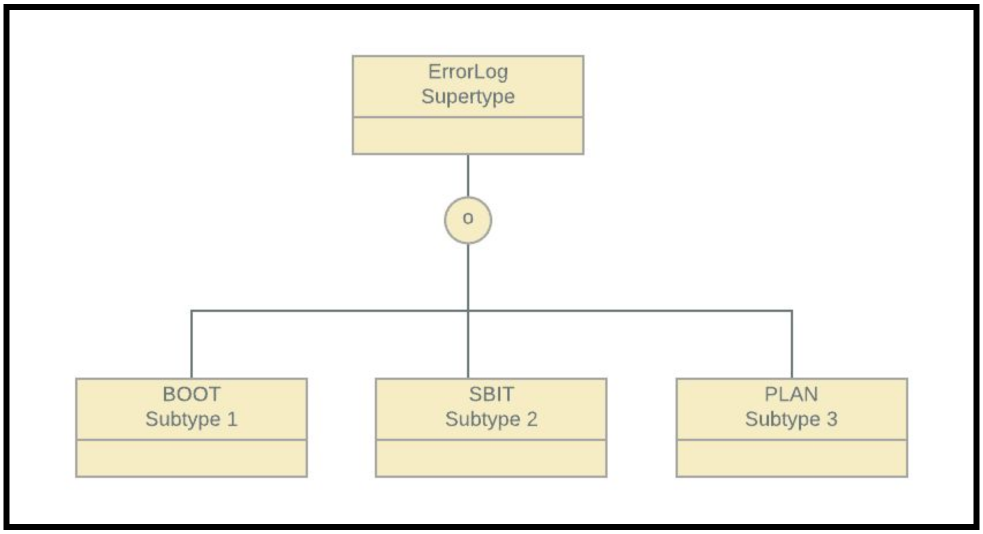
SubTests	
SensorID	Decimal 3
TestID	Decimal 3
SubtestID	Decimal 3
Status	BOOL

2.2 Conceptual ERD Diagram



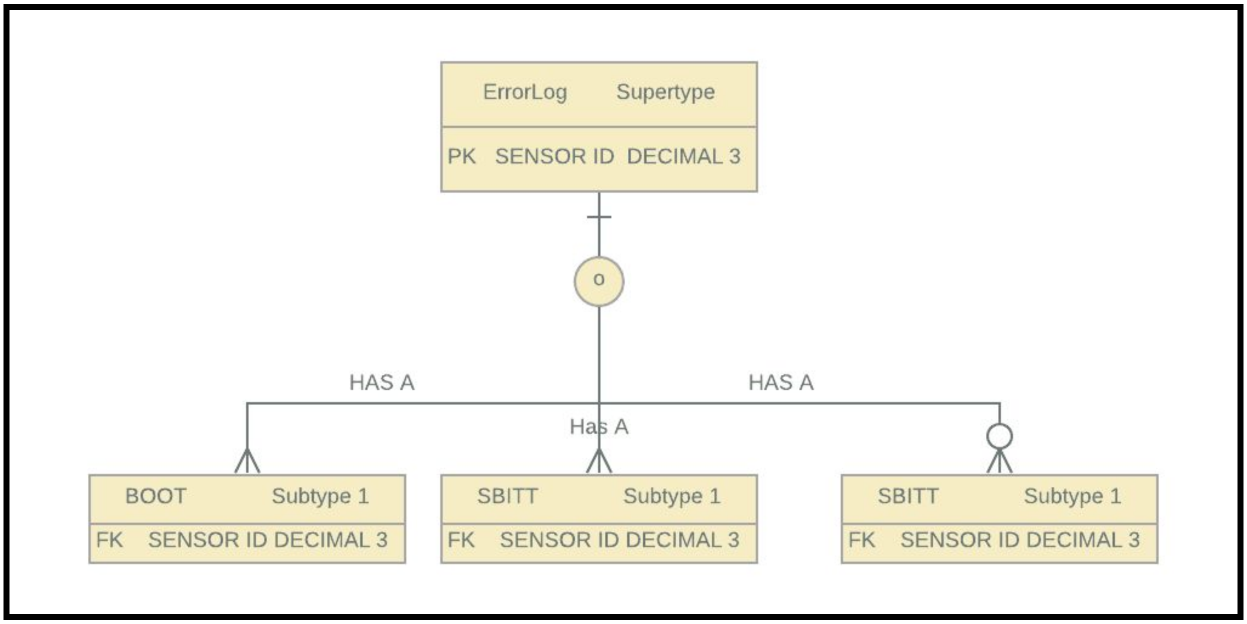
3.1 Specialization - Generalization Structural Database Rules

An Error log has a Boot log one of these or several of these.
An Error log has a SBIT log one of these or several of these.
An Error log has a Plan log,one of these or several of these.



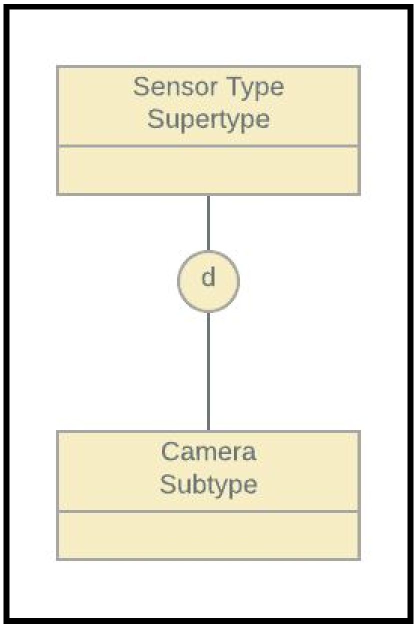
Conceptual

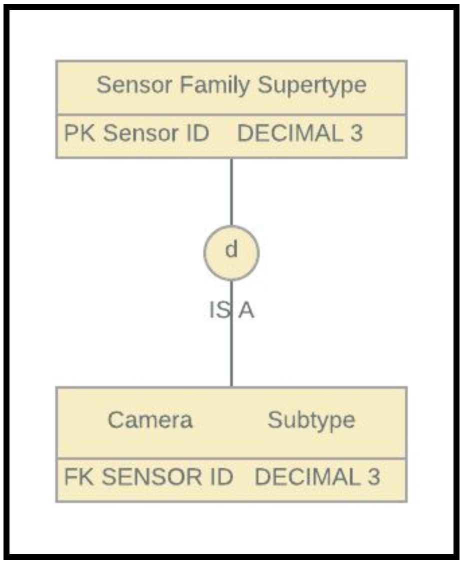
3.2 Initial DBMS Physical ERD



DBMS Physical

A Sensor Type is a Camera type and only one of these.





Conceptual

Physical

3.2 Specialization - Generalization Use Case:

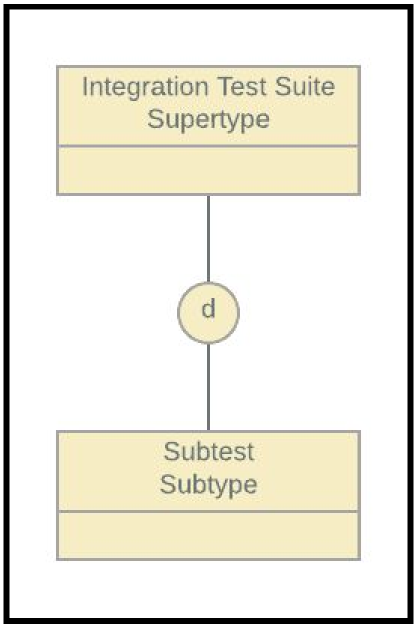
Chief Engineers (CE) Group Use case (old)

1. CE must gather information for a camera failure that occurred on a camera delivered to a customer.
2. CE would search database for customer name.
3. A report would indicate the camera IDs and software builds delivered for each camera and elapsed time of use for each camera.

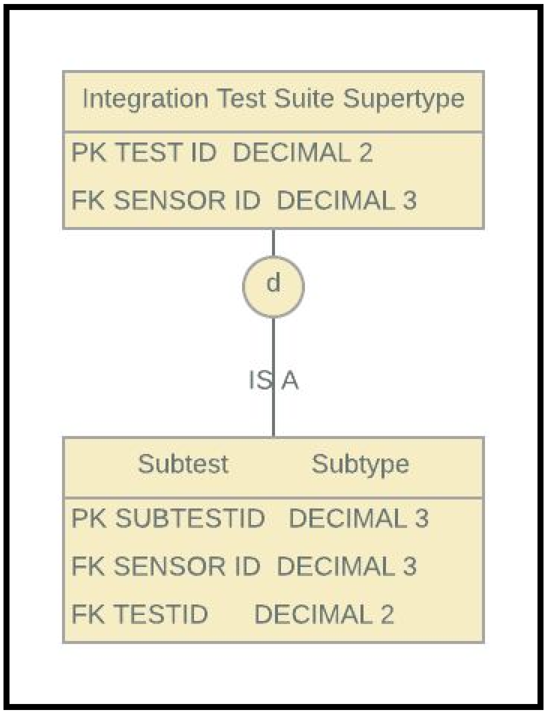
CE Group use case (new)

1. CE must gather information for a camera failure that occurred in the field
2. CE would search the database for a specific camera type for a customer name.
3. A report would indicate the camera ID and software builds delivered for each camera and elapsed time of use for each camera.

Integration Test is a Subtest and only one test.



Conceptual DBMS



Physical DBMS

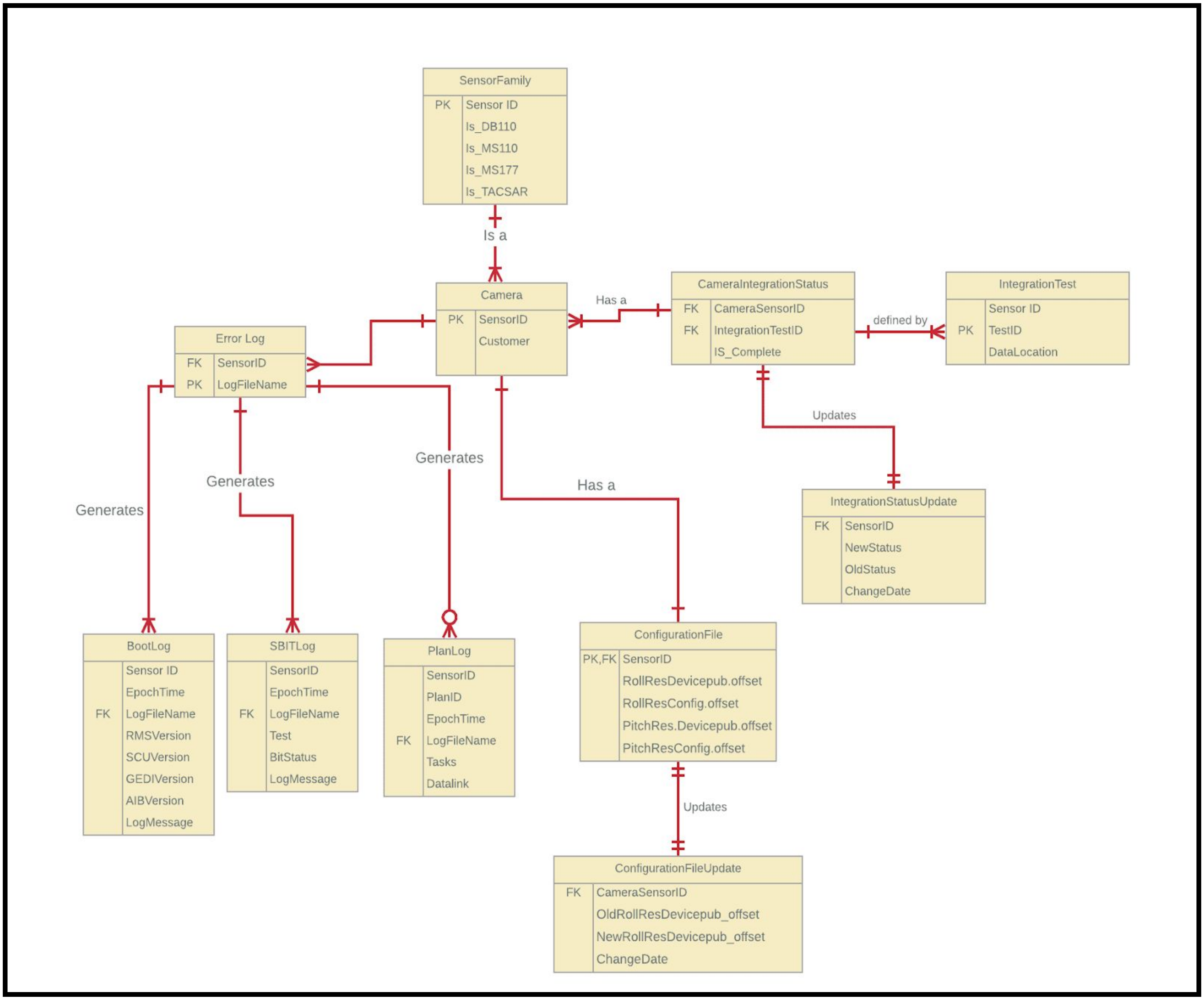
4. Full DBMS Physical ERD

Attributes

Table	Attribute	Data Type	Reasoning
Error Log	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
Error Log	FileName	Varchar(50)	Filenames will be 'date_errorlog_sensorid.log'
Boot	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
Boot	Time	Decimal(16)	Epoch Time
Boot	FileName	Varchar(50)	Filenames will be 'date_errorlog_sensorid.log'
Boot	RMSVersion	Varchar(255)	Usually a long string such as DB110_IQAF_MAIN_REL16.3_20140925
Boot	SCUVersion	Varchar(255)	Usually a long string such as DB110_IQAF_MAIN_REL16.3_20140926
Boot	GEDIVersion	Varchar(255)	Usually a long string such as DB110_IQAF_MAIN_REL16.3_20140927
Boot	AIBVersion	Varchar(255)	Usually a long string such as DB110_IQAF_MAIN_REL16.3_20140928
Boot	Message	Varchar(255)	Usually a long string such as DB110_IQAF_MAIN_REL16.3_20140929
SBIT	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
SBIT	Time	Decimal(16)	Epoch Time
SBIT	FileName	Varchar(50)	Filenames will be 'date_errorlog_sensorid.log'
SBIT	Test	Varchar(25)	SBIT Test Name
SBIT	Status	bit	0 = Fail, 1 = Pass
SBIT	Message	Varchar(255)	String with additional SBIT information
Plan	PlanID	Varchar(50)	Usually something like 'Date_pilotName' or 'Date_Area'
Plan	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
Plan	Time	Decimal(16)	Epoch Time
Plan	FileName	Varchar(50)	Filenames will be 'date_errorlog_sensorid.log'
Plan	Tasks	Decimal(3)	Maximum number < 100
Plan	Datalink	bit	True or False to indicate is DL was in use
ConfigurationFile	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
ConfigurationFile	RollResDevicepub.offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	RollResConfig.offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	PitchRes.Devicepub.offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx

ConfigurationFile	PitchResConfig.offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	BS.nf1bm0	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	BS.nf1bm1	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	BS.nf1bm2	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	BS.df1bm0	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	BS.df1bm1	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile	BS.df1bm2	Decimal(2,8)	Values tend to look like x.xxxxxxxx
Camera	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
Camera	Customer	Varchar(25)	Usually the name of a country but might be NULL if not sold
SensorFamily	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
SensorFamily	Is_DB110	bit	True or False to indicate if a certain family
SensorFamily	Is_MS110	bit	True or False to indicate if a certain family
SensorFamily	Is_MS177	bit	True or False to indicate if a certain family
SensorFamily	Is_TACSAR	bit	True or False to indicate if a certain family
IntegrationTest	SensorID		SensorID will be a number between 1 - 500
IntegrationTest	TestID	Decimal(2)	1 = Controls, 2 = LOS, 3 = Thermal, 4 = Auxillary, 5 = Imaging
IntegrationTest	SubtestID	Decimal(1,1)	Numerical representation of a subtest such as '3.2' or '2.2'
IntegrationTest	Completion_Percentage	INT	Up to 100%
IntegrationTest	DataLocation	Varchar(255)	Expect this to be a URL to a sharepoint location
Subtest	TestID	Decimal(1)	Numerica representation of Test 1 - Test 5
Subtest	SubtestID	Decimal(1,1)	Numerical representation of a subtest such as '3.2' or '2.2'
Subtest	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
Subtest	Status	bit	0 = Fail, 1 = Pass
SubtestChange	ChangeID	Decimal(12)	Primary Key
SubtestChange	SensorID	Decimal(3)	Foreign Key
SubtestChange	SubtestID	Decimal(3,1)	Numerical representation of a subtest such as '3.2' or '2.2'
SubtestChange	NewStatus	bit	New Status of the testid
SubtestChange	OldStatus	bit	Old Status of the testid
SubtestChange	ChangeDate	DATE	Date of the status Change
ConfigurationFileUpdate	SensorID	Decimal(3)	SensorID will be a number between 1 - 500
ConfigurationFileUpdate	OldRollResDevicepub_offs	Decimal(2,8)	Values tend to look like x.xxxxxxxx

	et		
ConfigurationFileUpdate	NewRollResDevicepub_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFileUpdate	OldRollResConfig_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFileUpdate	NewRollResConfig_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFileUpdate	OldPitchRes_Devicepub_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFileUpdate	NewPitchRes_Devicepub_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFileUpdate	OldPitchResConfig_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFileUpdate	NewPitchResConfig_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx



Physical ERD Implementation

4.1 Normalization

Error Log Normalization:

PK : LogFileName

FK: SensorID

Candidate Keys: All

1NF - PK Identified, no repeating groups

2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

Error Log
SensorID
LogFileName

IntegrationTest Normalization

PK : TestID

FK:

Candidate Keys: All

1NF - PK Identified, no repeating groups

2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

IntegrationTest
SensorID
TestID
TestName
DataLocation

BootLog Table Normalization:

PK : LogFileName

FK: LogFileName

Candidate Keys: All

1NF - PK Identified, no repeating groups

2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

BootLog
SensorID
EpochTime
LogFileName
Message

SBITLog Table Normalization:

PK : LogFileName

FK: LogFileName

Candidate Keys: All

1NF - PK Identified, no repeating groups

2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

SBITLog
SensorID
EpochTime
LogFileName
Test
BitStatus
Message

PlanLog Table Normalization:

PK : LogFileName

FK: LogFileName

Candidate Keys: All

1NF - PK Identified, no repeating groups

2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

PlanLog
SensorID
EpochTime
LogFileName
Tasks
Datalink
PlanID

Configuration File Table Normalization

PK : SensorID

FK: SensorID

Candidate Keys: All

1NF - PK Identified, no repeating groups

2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

ConfigurationFile
SensorID
RollResDevicepub_offset
RollResConfig_offset
PitchRes.Devicepub_offset
PitchResConfig_offset

Sensor Family Table Normalization:

PK : SensorID

FK: SensorID

Candidate Keys: All

- **1NF - PK Identified, no repeating groups**
- **2NF - No Attributes dependent on the PK**
- **3NF - No Attributes dependent on other attributes**

SensorFamily
SensorID
Is_DB110
Is_MS110
Is_MS177

Camera Table Normalization:

PK : SensorID

FK: SensorID

Candidate Keys: All

- **1NF - PK Identified, no repeating groups**
- **2NF - No Attributes dependent on the PK**
- **3NF - No Attributes dependent on other attributes**

Camera
SensorID
Customer

CameraIntegrationStatus Table Normalization:

PK : NONE

FK: CameraSensorID, TestID

This table is a join table used to break up the Camera - IntegrationTest Many to Many relationship.

This table enables a Camera - CameraIntegrationStatus - IntegrationTest relationship is devolved into a many to one to many relationship.

CameraIntegrationStatus
CameraSensorID
IntegrationTestID

IntegrationStatusUpdate Table Normalization:

PK : SensorID
FK: SensorID
Candidate Keys: All

1NF - PK Identified, no repeating groups

2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

IntegrationStatusUpdate
SensorID
NewStatus
OldStatus
ChangeDate

ConfigurationFileUpdate Table Normalization:

PK : SensorID
FK: SensorID
Candidate Keys: All

1NF - PK Identified, no repeating groups

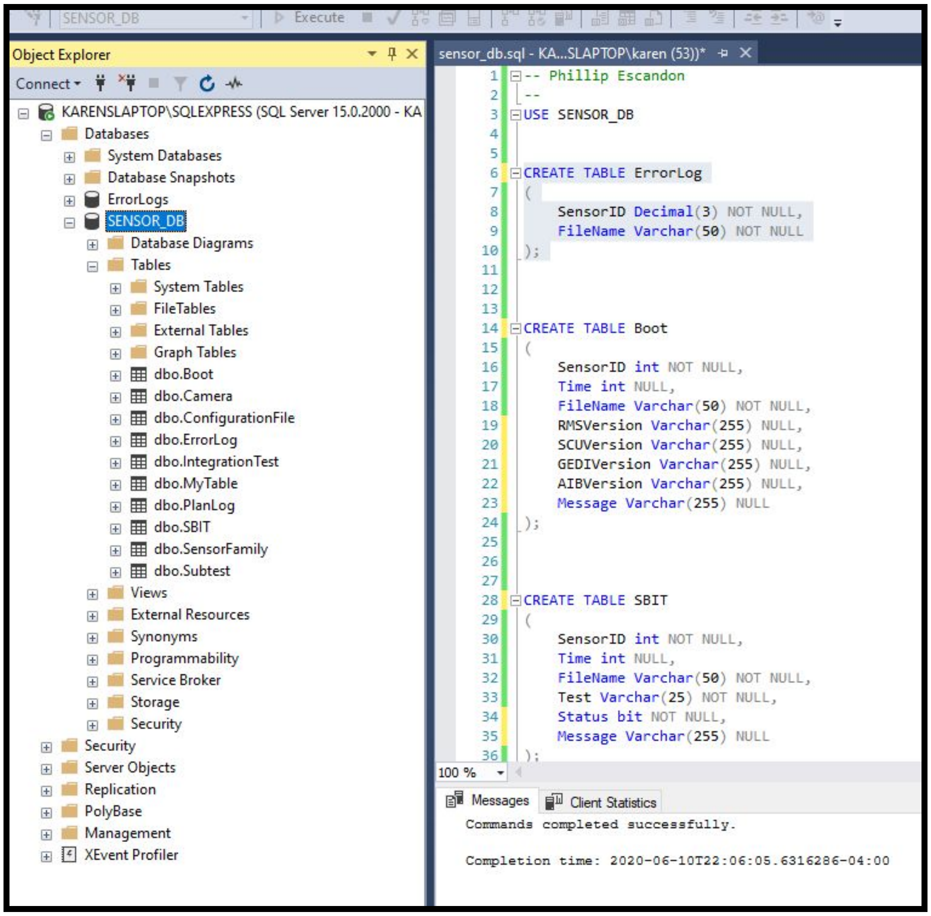
2NF - No Attributes dependent on the PK

3NF - No Attributes dependent on other attributes

ConfigurationFileUpdate
CameraSensorID
OldRollResDevicepub_offset
NewRollResDevicepub_offset
ChangeDate

4.2 Tables and Constraints

See attached SQL File.



```sql
-- Phillip Escandon
--
USE SENSOR_DB


CREATE TABLE ErrorLog
(
    SensorID Decimal(3) NOT NULL,
    FileName Varchar(50) NOT NULL
);



CREATE TABLE Boot
(
    SensorID int NOT NULL,
    Time int NULL,
    FileName Varchar(50) NOT NULL,
    RMSVersion Varchar(255) NULL,
    SCUVersion Varchar(255) NULL,
    GEDIVersion Varchar(255) NULL,
    AIBVersion Varchar(255) NULL,
    Message Varchar(255) NULL
);



CREATE TABLE SBIT
(
    SensorID int NOT NULL,
    Time int NULL,
    FileName Varchar(50) NOT NULL,
    Test Varchar(25) NOT NULL,
    Status bit NOT NULL,
    Message Varchar(255) NULL
);
```

```
Commands completed successfully.

Completion time: 2020-06-10T22:06:05.6316286-04:00
```

4.3 Index Placement and Creation

Primary Keys:

Primary Keys	Description
ErrorLog.LogFileName	The LogFileName is UNIQUE because it is traditionally composed to the missionID_Date, which varies from country and pilot.
Boot.LogFileName	The Logfile is UNIQUE.
SBIT.LogFileName	The Logfile is UNIQUE.
PlanLog.LogFileName	The Logfile is UNIQUE.

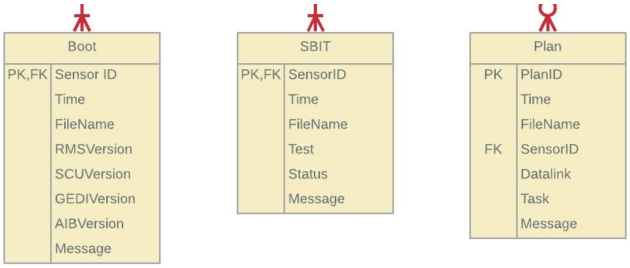
ConfigurationFile.SensorID	SensorID uniquly idenfies the Configuration File
Camera.SensorID	Uniquly identifies the Camera
SensorFamily.SensorID	Uniquely Identifies the SensorFamily
IntegrationTest.TestID	Uniquely Identifies the Test and Subtest
IntegrationStatusUpdate.SensorID	Identifies a status update for a Sensor indicated by the sensorID.

Foreign Keys:

Foreign Keys	
ErrorLog.SensorID	This FK references is referencing the Camera table SensorID.
PlanLog.LogFileName	This FK references the ErrorLog table LogFileName, which is it's primary key.
BootLog.FileName	This FK references the ErrorLog table LogFileName, which is it's primary key.
SBITLog.Filename	This FK references the ErrorLog table LogFileName, which is it's primary key.
CameraIntegrationStatus.CameraSensorID	This FK references the Camera tables SensorID which is it's primary key. This table is a JOIN table and does not have a Primary Key
CameraIntegrationStatus.IntegrationTestID	This FK references the IntegrationTest TestID which is it's primary key.This table is a JOIN table and does not have a Primary Key

Use Cases:

Inspecting my ERD I could see that Time was a consistent attribute across the logs.



Use Case:

An Engineer wanted to inspect all logs that occurred on two specific days.

Pseudocode:

```
Select SBIT Messages and Plan IDs WHERE TIME > (some date).
```

The entire script was executed - on re-execution it was verified that the index was already created.

Use Case 2:

The history of test status is important. An engineer is interested in common *failures between* SBIT and Subtests.

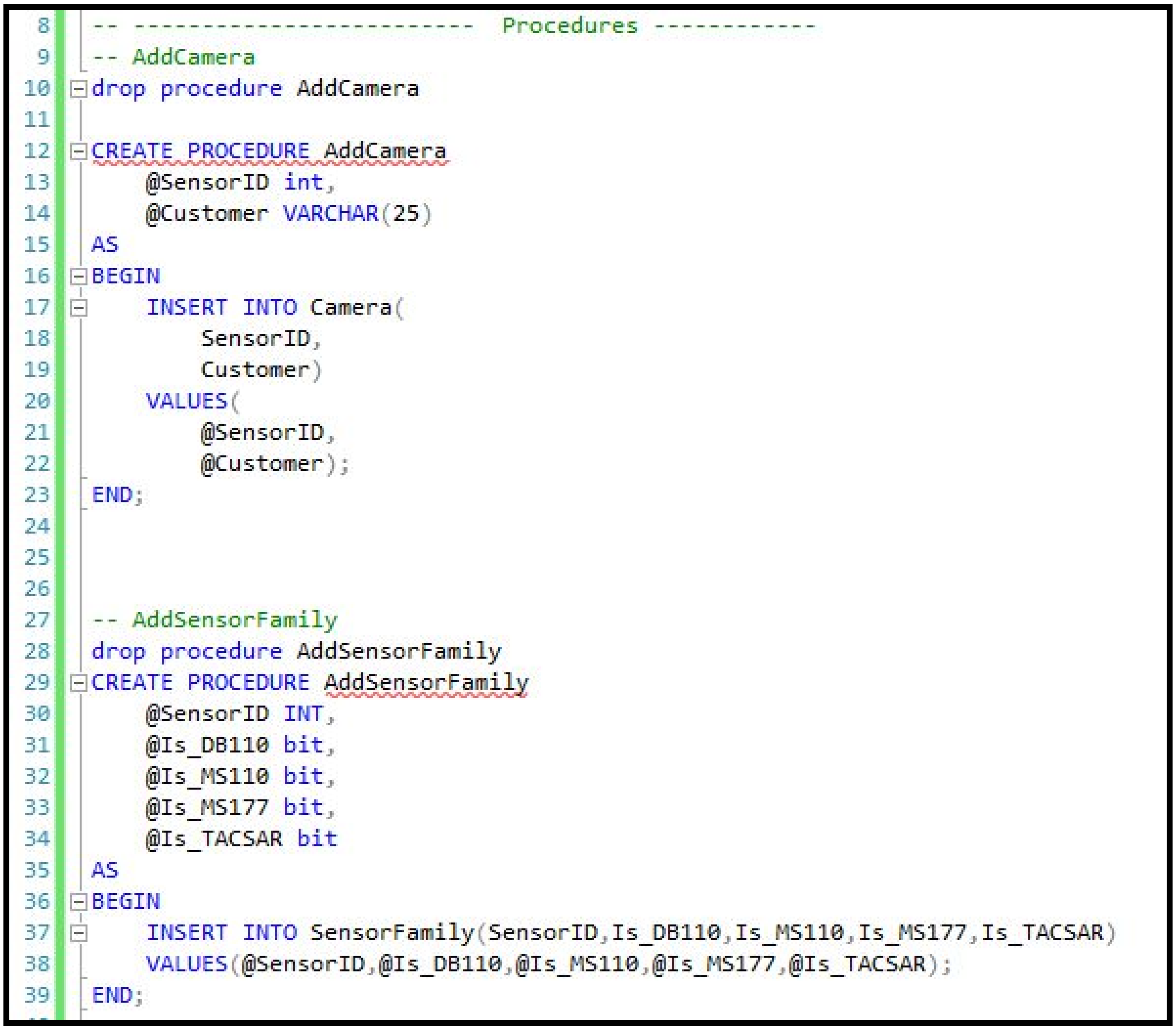
The Status attribute is a possible candidate for indexing.

5.1 Reusable Transaction - Oriented Store Procedures

All procedures are located in the SensorProcedures.sql file.

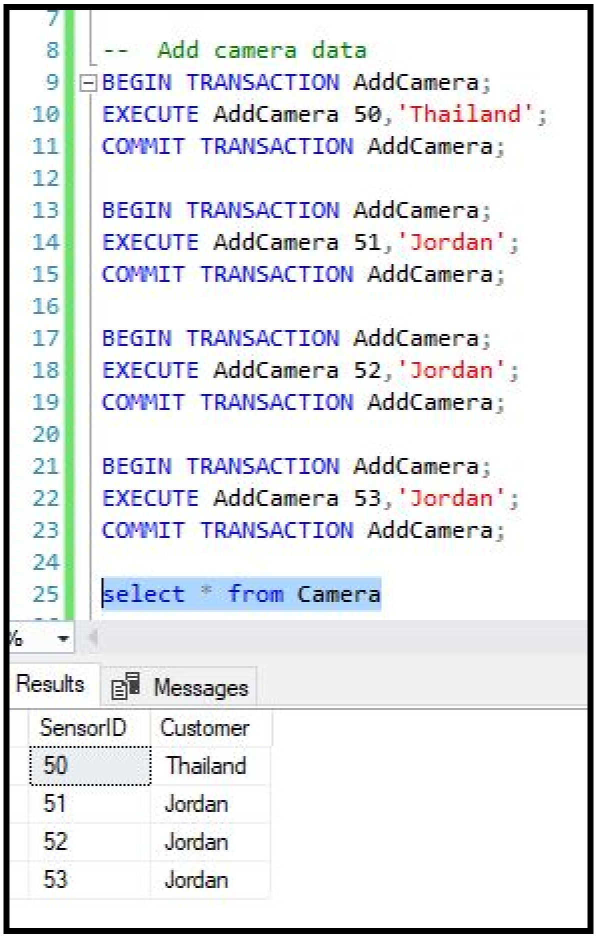
Use cases for populating database with data

1. New Sensor comes from the Production floor into the Integration Lab.
   This begins the tracking of the sensor.
   a. CREATE PROCEDURE AddSensorFamily();
   b. CREATE PROCEDURE AddCamera();



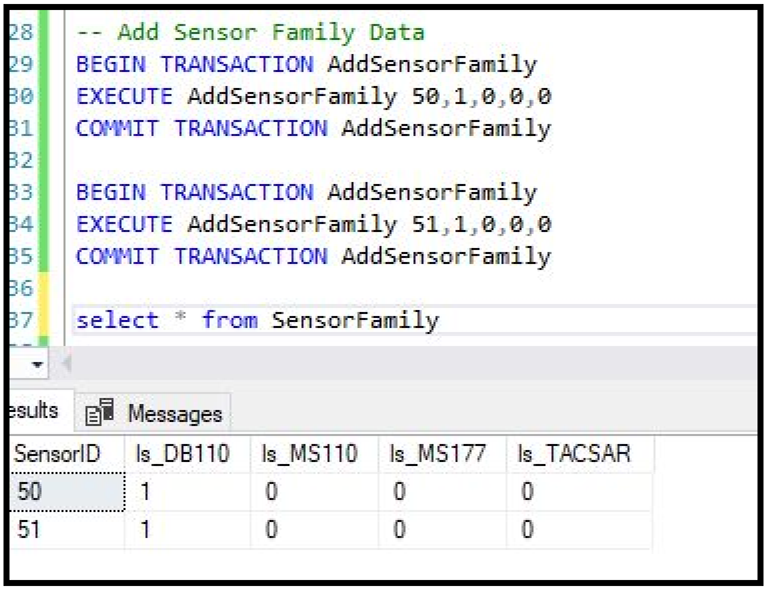
```
-- -------------------------  Procedures ------------
-- AddCamera
drop procedure AddCamera

CREATE PROCEDURE AddCamera
    @SensorID int,
    @Customer VARCHAR(25)
AS
BEGIN
    INSERT INTO Camera(
        SensorID,
        Customer)
    VALUES(
        @SensorID,
        @Customer);
END;



-- AddSensorFamily
drop procedure AddSensorFamily
CREATE PROCEDURE AddSensorFamily
    @SensorID INT,
    @Is_DB110 bit,
    @Is_MS110 bit,
    @Is_MS177 bit,
    @Is_TACSAR bit
AS
BEGIN
    INSERT INTO SensorFamily(SensorID,Is_DB110,Is_MS110,Is_MS177,Is_TACSAR)
    VALUES(@SensorID,@Is_DB110,@Is_MS110,@Is_MS177,@Is_TACSAR);
END;
```

Stored Procedure Execution:



```sql
--  Add camera data
BEGIN TRANSACTION AddCamera;
EXECUTE AddCamera 50,'Thailand';
COMMIT TRANSACTION AddCamera;

BEGIN TRANSACTION AddCamera;
EXECUTE AddCamera 51,'Jordan';
COMMIT TRANSACTION AddCamera;

BEGIN TRANSACTION AddCamera;
EXECUTE AddCamera 52,'Jordan';
COMMIT TRANSACTION AddCamera;

BEGIN TRANSACTION AddCamera;
EXECUTE AddCamera 53,'Jordan';
COMMIT TRANSACTION AddCamera;

select * from Camera
```

SensorID	Customer
50	Thailand
51	Jordan
52	Jordan
53	Jordan

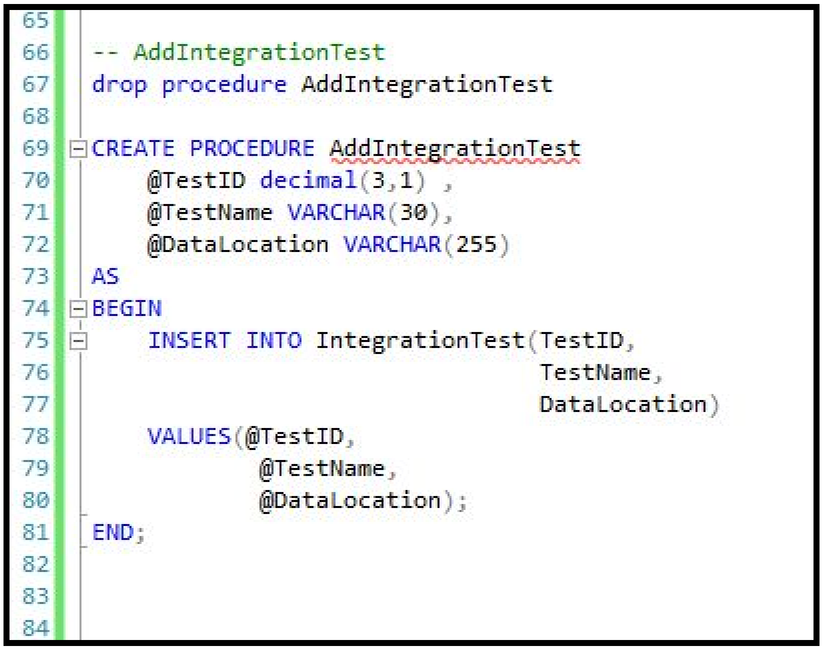


```sql
-- Add Sensor Family Data
BEGIN TRANSACTION AddSensorFamily
EXECUTE AddSensorFamily 50,1,0,0,0
COMMIT TRANSACTION AddSensorFamily

BEGIN TRANSACTION AddSensorFamily
EXECUTE AddSensorFamily 51,1,0,0,0
COMMIT TRANSACTION AddSensorFamily

select * from SensorFamily
```

SensorID	Is_DB110	Is_MS110	Is_MS177	Is_TACSAR
50	1	0	0	0
51	1	0	0	0

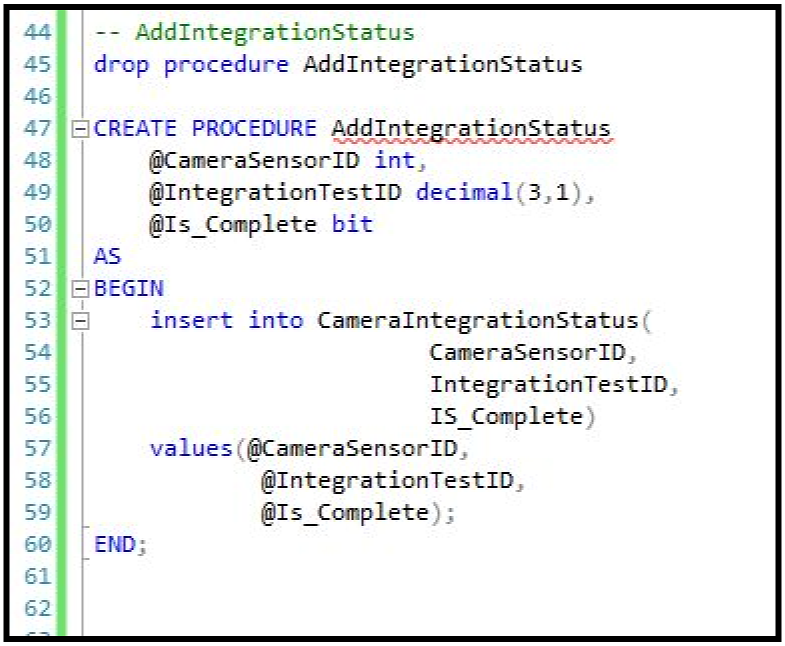
2. After a Test Readiness Review and all Avionics are available, the camera will begin an Integration Test Suite. An IntegrationStatus table will be populated as well.

    a. CREATE PROCEDURE AddIntegrationTest();



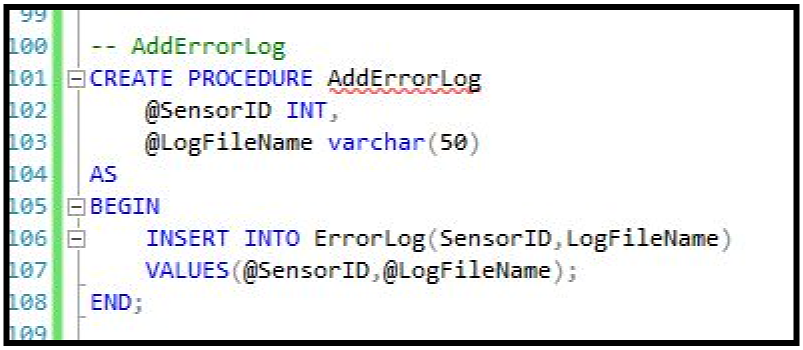
```
-- AddIntegrationTest
drop procedure AddIntegrationTest

CREATE PROCEDURE AddIntegrationTest
    @TestID decimal(3,1) ,
    @TestName VARCHAR(30),
    @DataLocation VARCHAR(255)
AS
BEGIN
    INSERT INTO IntegrationTest(TestID,
                                TestName,
                                DataLocation)
    VALUES(@TestID,
            @TestName,
            @DataLocation);
END;
```

    b. CREATE PROCEDURE AddIntegrationStatus

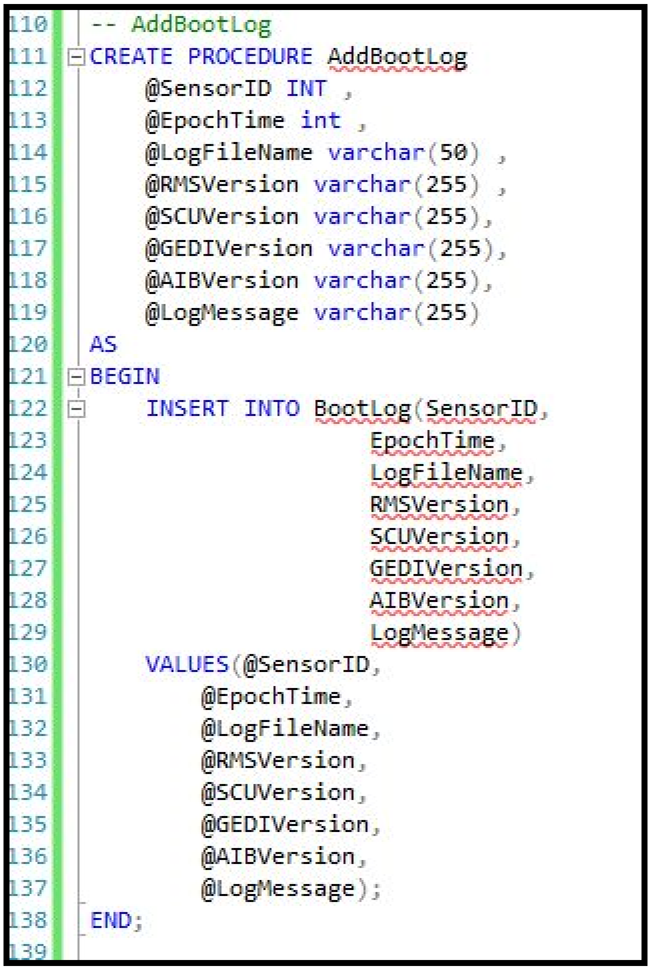


```
-- AddIntegrationStatus
drop procedure AddIntegrationStatus

CREATE PROCEDURE AddIntegrationStatus
    @CameraSensorID int,
    @IntegrationTestID decimal(3,1),
    @Is_Complete bit
AS
BEGIN
    insert into CameraIntegrationStatus(
                        CameraSensorID,
                        IntegrationTestID,
                        IS_Complete)
    values(@CameraSensorID,
            @IntegrationTestID,
            @Is_Complete);
END;
```

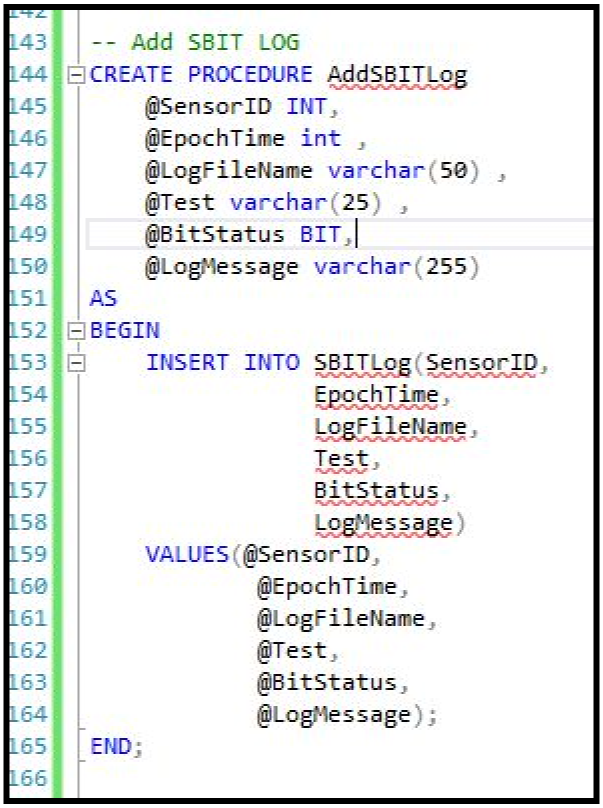
3. At the end of the first test, the camera is in the safe state where the camera is allowed to fully boot. The first errorlog will be generated.
    a. CREATE PROCEDURE addErrorLog();
    b. CREATE PROCEDURE addBootlLog();
    c. CREATE PROCEDURE addSBITLog();



```
-- AddErrorLog
CREATE PROCEDURE AddErrorLog
    @SensorID INT,
    @LogFileName varchar(50)
AS
BEGIN
    INSERT INTO ErrorLog(SensorID,LogFileName)
    VALUES(@SensorID,@LogFileName);
END;
```

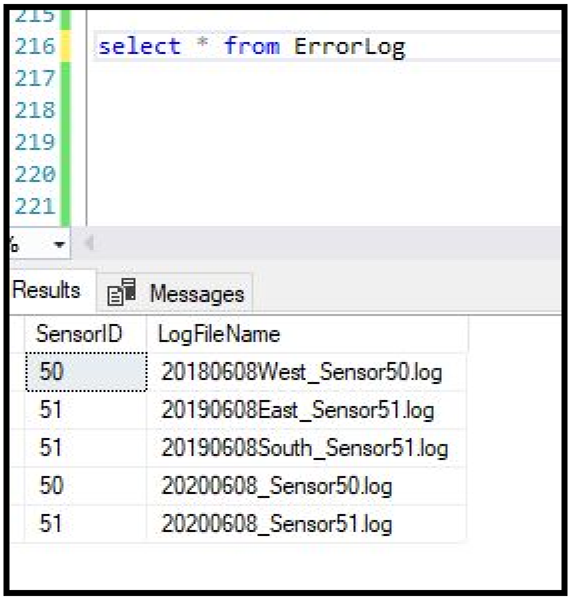


```
-- AddBootLog
CREATE PROCEDURE AddBootLog
    @SensorID INT ,
    @EpochTime int ,
    @LogFileName varchar(50) ,
    @RMSVersion varchar(255) ,
    @SCUVersion varchar(255),
    @GEDIVersion varchar(255),
    @AIBVersion varchar(255),
    @LogMessage varchar(255)
AS
BEGIN
    INSERT INTO BootLog(SensorID,
                        EpochTime,
                        LogFileName,
                        RMSVersion,
                        SCUVersion,
                        GEDIVersion,
                        AIBVersion,
                        LogMessage)
    VALUES(@SensorID,
        @EpochTime,
        @LogFileName,
        @RMSVersion,
        @SCUVersion,
        @GEDIVersion,
        @AIBVersion,
        @LogMessage);
END;
```



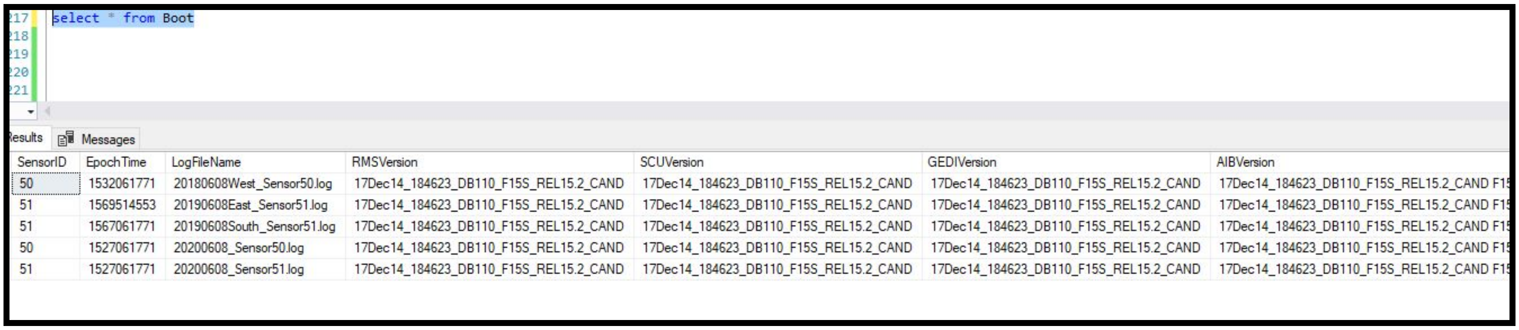
```
-- Add SBIT LOG
CREATE PROCEDURE AddSBITLog
    @SensorID INT,
    @EpochTime int ,
    @LogFileName varchar(50) ,
    @Test varchar(25) ,
    @BitStatus BIT,
    @LogMessage varchar(255)
AS
BEGIN
    INSERT INTO SBITLog(SensorID,
                        EpochTime,
                        LogFileName,
                        Test,
                        BitStatus,
                        LogMessage)
    VALUES(@SensorID,
            @EpochTime,
            @LogFileName,
            @Test,
            @BitStatus,
            @LogMessage);
END;
```

Data inserted into the Errorlog,Boot,and SBIT tables.



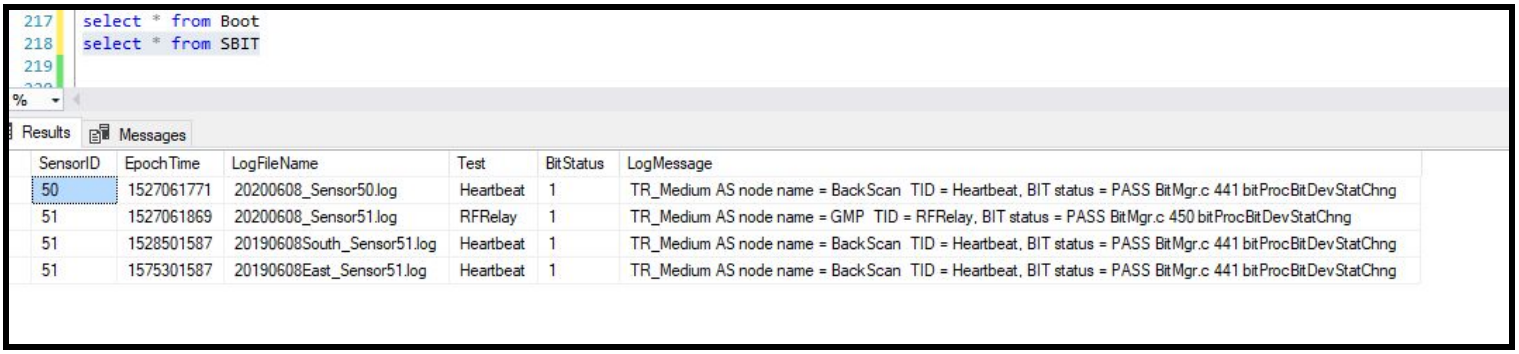
```
select * from ErrorLog
```

SensorID	LogFileName
50	20180608West_Sensor50.log
51	20190608East_Sensor51.log
51	20190608South_Sensor51.log
50	20200608_Sensor50.log
51	20200608_Sensor51.log



```sql
select * from Boot
```

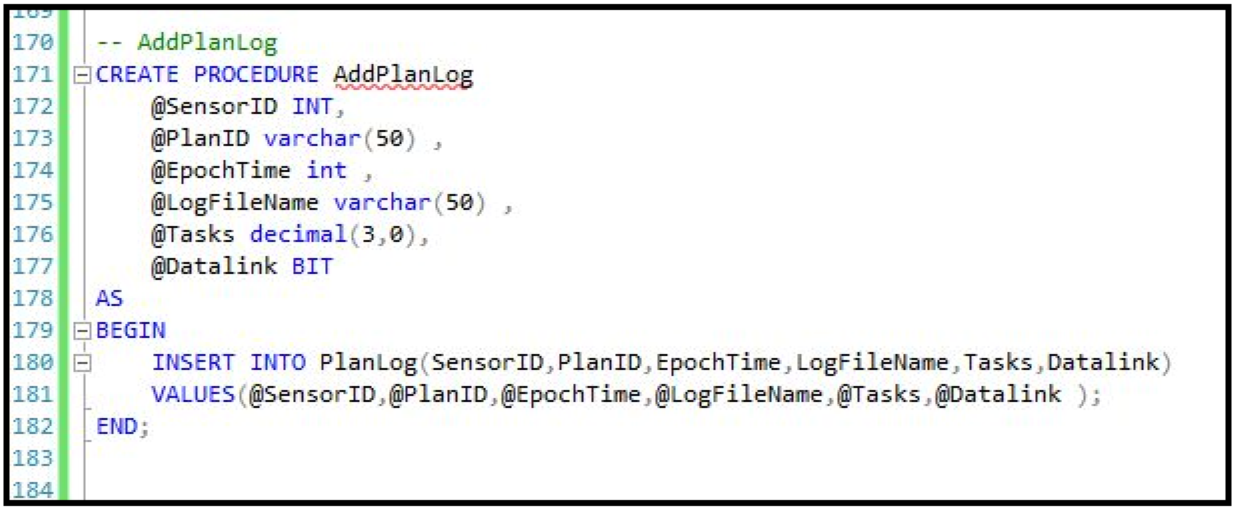
SensorID	EpochTime	LogFileName	RMSVersion	SCUVersion	GEDIVersion	AIBVersion
50	1532061771	20180608West_Sensor50.log	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND F15
51	1569514553	20190608East_Sensor51.log	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND F15
51	1567061771	20190608South_Sensor51.log	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND F15
50	1527061771	20200608_Sensor50.log	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND F15
51	1527061771	20200608_Sensor51.log	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND	17Dec14_184623_DB110_F15S_REL15.2_CAND F15



```sql
select * from Boot
select * from SBIT
```

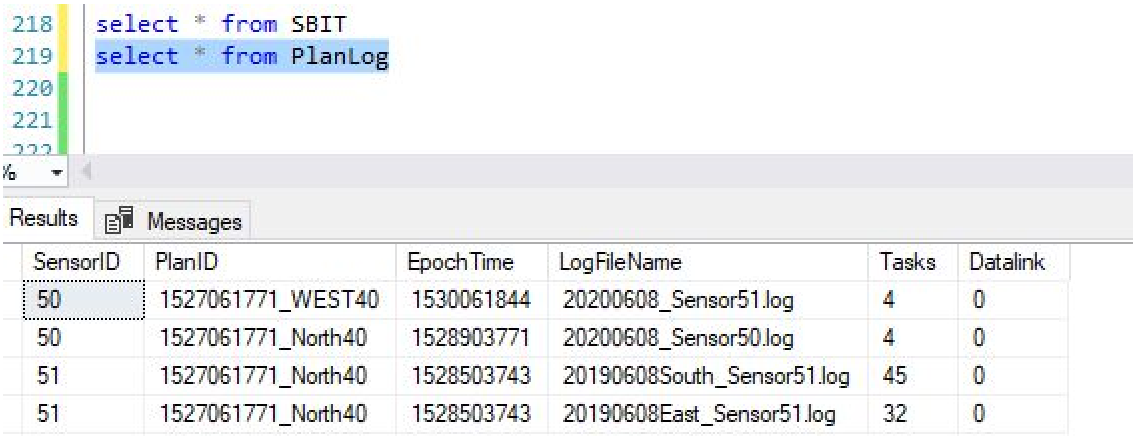
SensorID	EpochTime	LogFileName	Test	BitStatus	LogMessage
50	1527061771	20200608_Sensor50.log	Heartbeat	1	TR_Medium AS node name = BackScan TID = Heartbeat, BIT status = PASS BitMgr.c 441 bitProcBitDevStatChng
51	1527061869	20200608_Sensor51.log	RFRelay	1	TR_Medium AS node name = GMP TID = RFRelay, BIT status = PASS BitMgr.c 450 bitProcBitDevStatChng
51	1528501587	20190608South_Sensor51.log	Heartbeat	1	TR_Medium AS node name = BackScan TID = Heartbeat, BIT status = PASS BitMgr.c 441 bitProcBitDevStatChng
51	1575301587	20190608East_Sensor51.log	Heartbeat	1	TR_Medium AS node name = BackScan TID = Heartbeat, BIT status = PASS BitMgr.c 441 bitProcBitDevStatChng

4. A Plan will be used during one of the tests. This errorlog will contain a ‘Plan Log’ section.
    a. CREATE PROCEDURE addPlanLog();



```sql
-- AddPlanLog
CREATE PROCEDURE AddPlanLog
    @SensorID INT,
    @PlanID varchar(50) ,
    @EpochTime int ,
    @LogFileName varchar(50) ,
    @Tasks decimal(3,0),
    @Datalink BIT
AS
BEGIN
    INSERT INTO PlanLog(SensorID,PlanID,EpochTime,LogFileName,Tasks,Datalink)
    VALUES(@SensorID,@PlanID,@EpochTime,@LogFileName,@Tasks,@Datalink );
END;
```

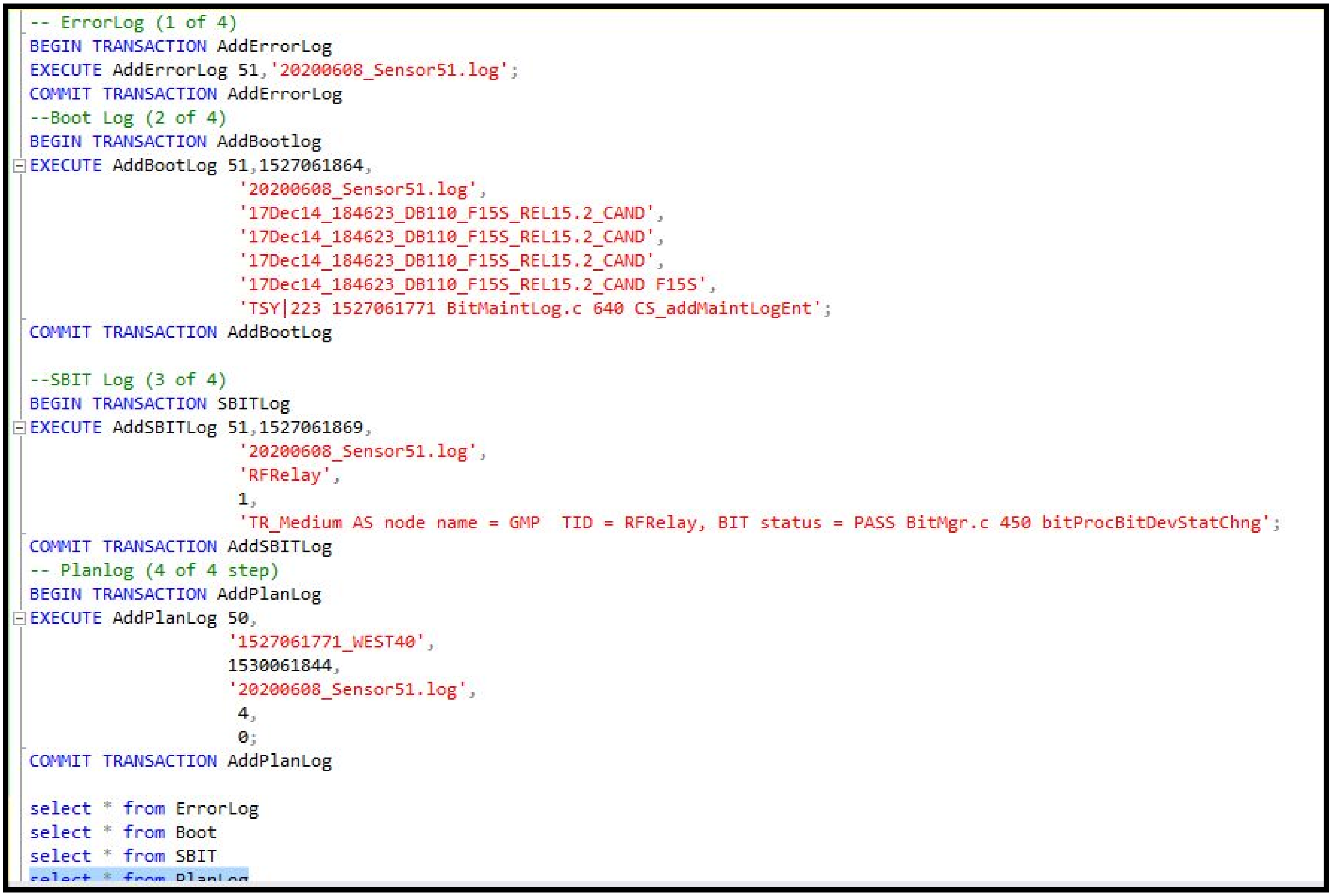
Now to add data via the procedure.



```sql
select * from SBIT
select * from PlanLog
```

SensorID	PlanID	EpochTime	LogFileName	Tasks	Datalink
50	1527061771_WEST40	1530061844	20200608_Sensor51.log	4	0
50	1527061771_North40	1528903771	20200608_Sensor50.log	4	0
51	1527061771_North40	1528503743	20190608South_Sensor51.log	45	0
51	1527061771_North40	1528503743	20190608East_Sensor51.log	32	0

Since the sequence of Logfile insertions is linear, meaning that the ErrorLog begets the BootLog, SBITLog and sometimes the PlanLog, all together the calls look like the image below.



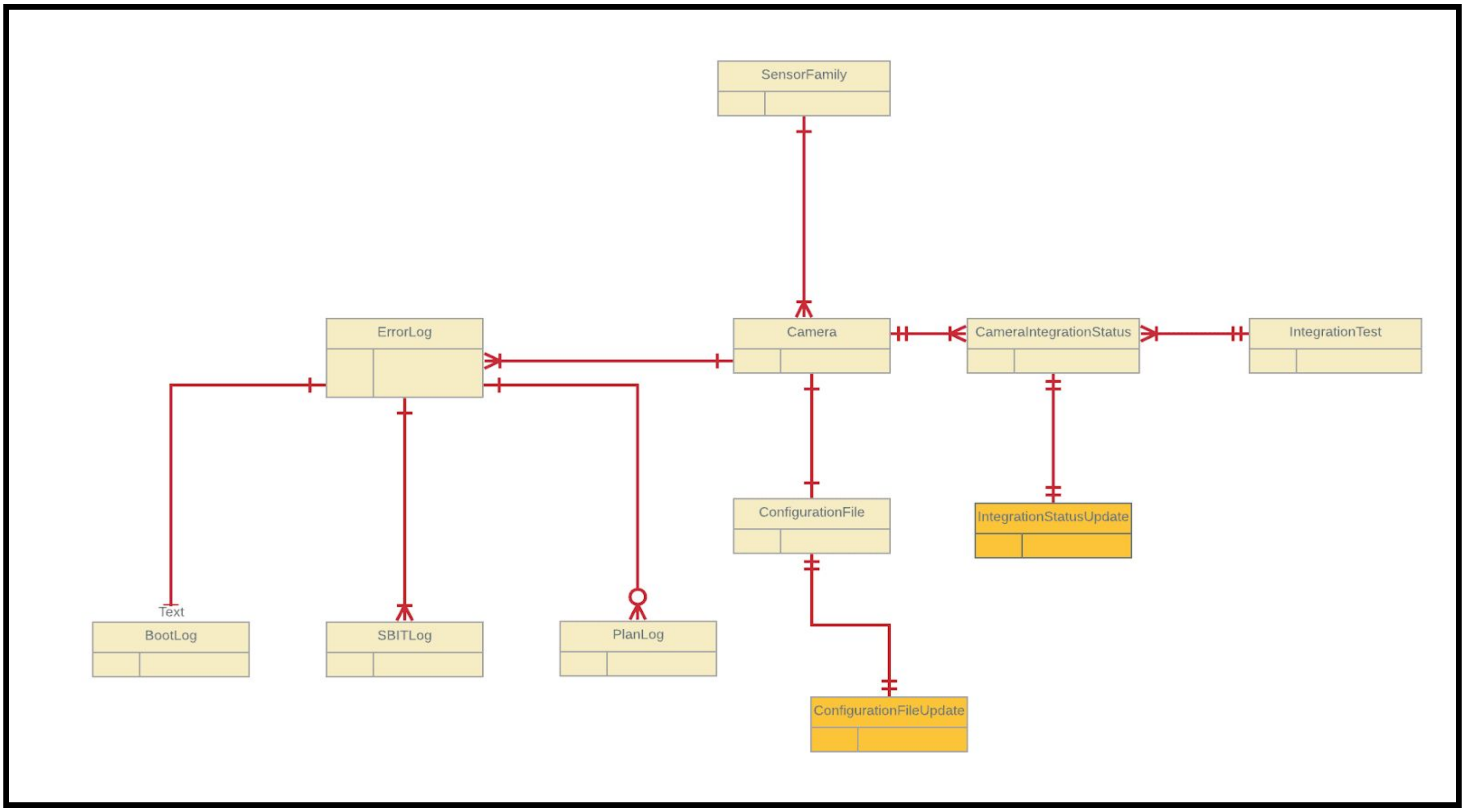
```sql
-- ErrorLog (1 of 4)
BEGIN TRANSACTION AddErrorLog
EXECUTE AddErrorLog 51,'20200608_Sensor51.log';
COMMIT TRANSACTION AddErrorLog
--Boot Log (2 of 4)
BEGIN TRANSACTION AddBootlog
EXECUTE AddBootLog 51,1527061864,
                    '20200608_Sensor51.log',
                    '17Dec14_184623_DB110_F15S_REL15.2_CAND',
                    '17Dec14_184623_DB110_F15S_REL15.2_CAND',
                    '17Dec14_184623_DB110_F15S_REL15.2_CAND',
                    '17Dec14_184623_DB110_F15S_REL15.2_CAND F15S',
                    'TSY|223 1527061771 BitMaintLog.c 640 CS_addMaintLogEnt';
COMMIT TRANSACTION AddBootLog

--SBIT Log (3 of 4)
BEGIN TRANSACTION SBITLog
EXECUTE AddSBITLog 51,1527061869,
                    '20200608_Sensor51.log',
                    'RFRelay',
                    1,
                    'TR_Medium AS node name = GMP  TID = RFRelay, BIT status = PASS BitMgr.c 450 bitProcBitDevStatChng';
COMMIT TRANSACTION AddSBITLog
-- Planlog (4 of 4 step)
BEGIN TRANSACTION AddPlanLog
EXECUTE AddPlanLog 50,
                   '1527061771_WEST40',
                   1530061844,
                   '20200608_Sensor51.log',
                    4,
                    0;
COMMIT TRANSACTION AddPlanLog

select * from ErrorLog
select * from Boot
select * from SBIT
select * from PlanLog
```

5.2 History Table

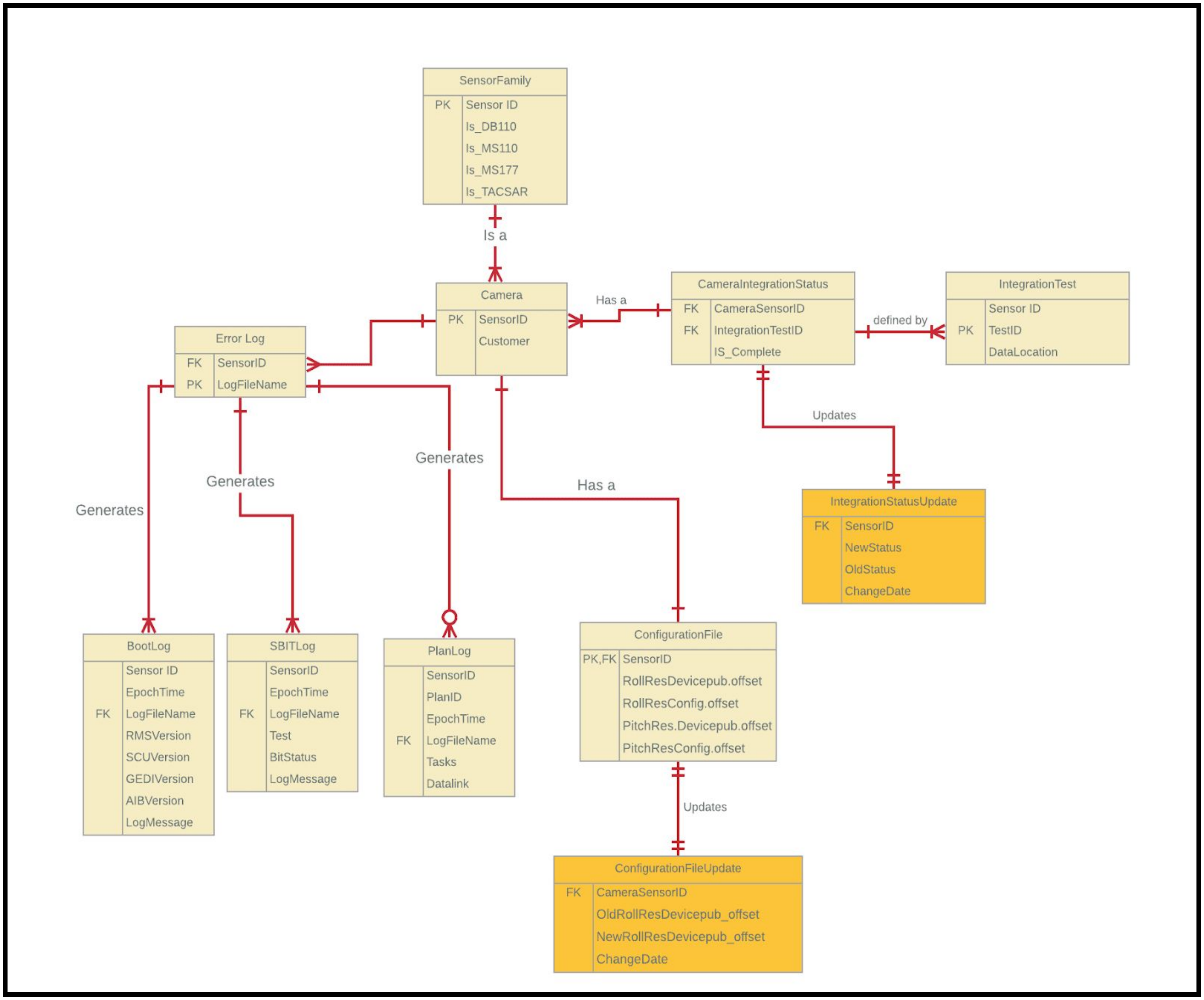
The two entities I will track with the history table are the IntegrationTestUpdate as well as the ConfigurationFileUpdate.

To implement the new status change entity I have added to the Conceptual ERD and Physical ERDs.

Conceptual ERD:

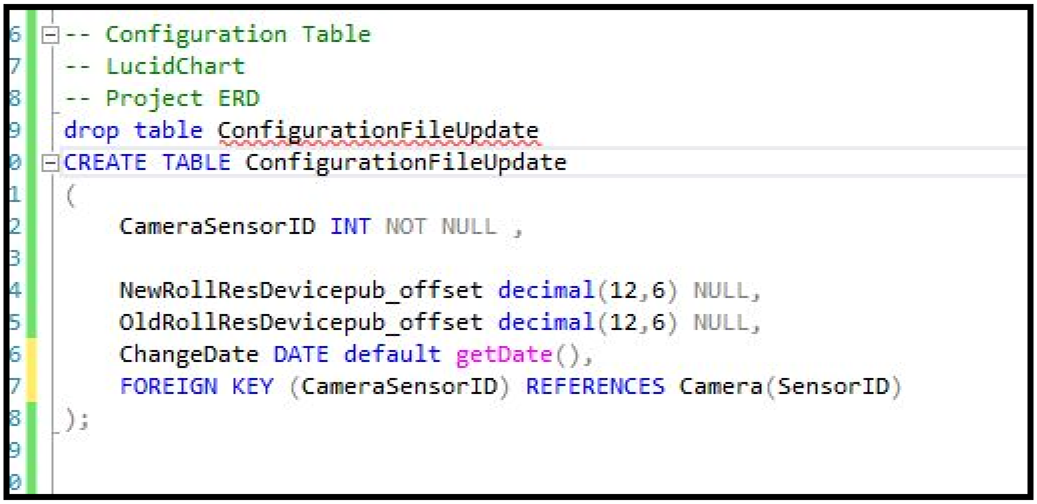


The Physical ERD is shown below:



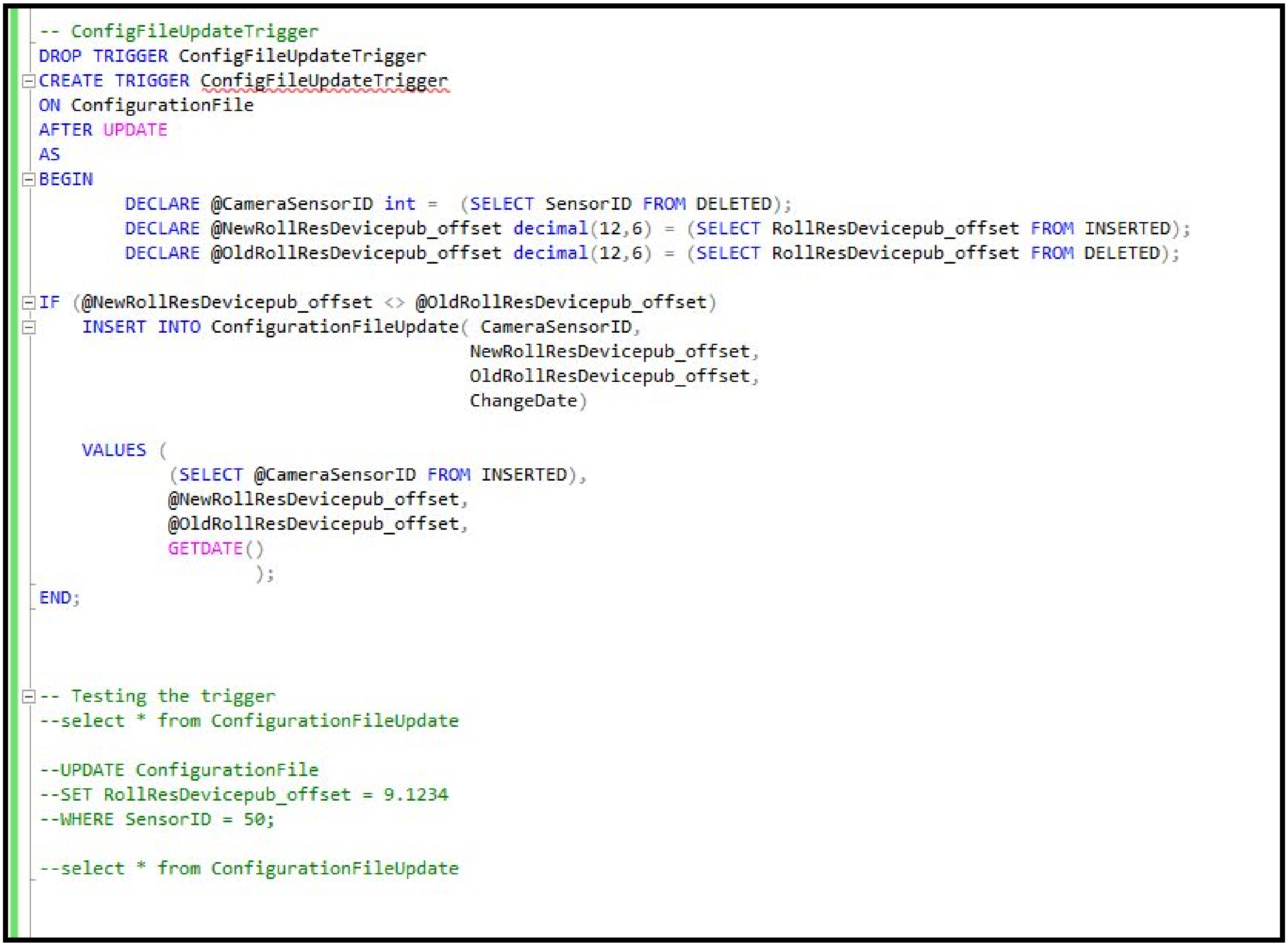
Attributes Table

ConfigurationFile Update	SensorID	INT	SensorID will be a number between 1 - 500
ConfigurationFile Update	OldRollResDevicepub_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile Update	NewRollResDevicepub_offset	Decimal(2,8)	Values tend to look like x.xxxxxxxx
ConfigurationFile Update	DateChange	Date	Date that the table was updated



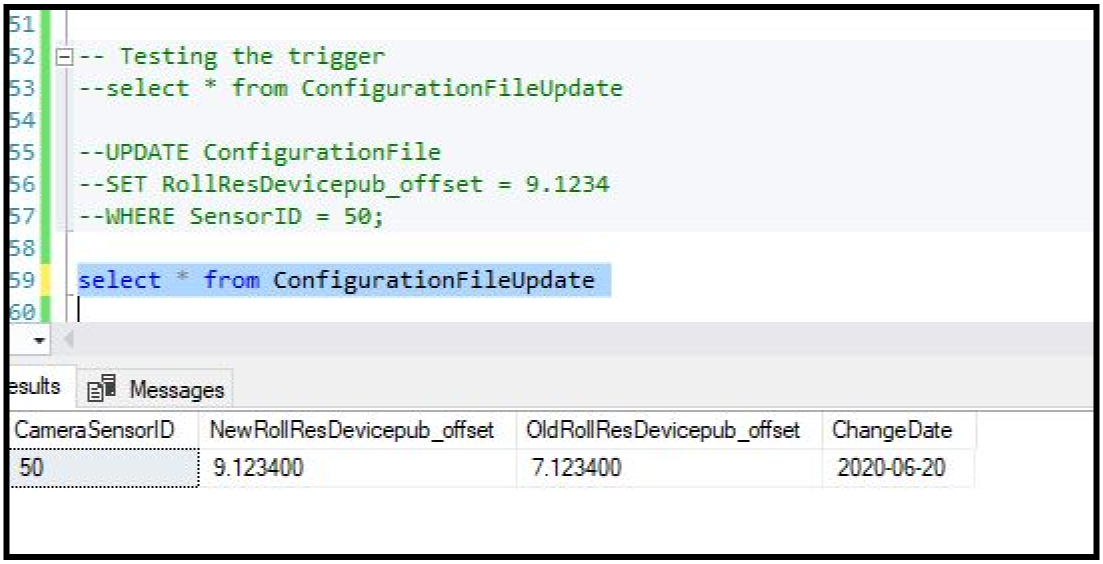
```
-- Configuration Table
-- LucidChart
-- Project ERD
drop table ConfigurationFileUpdate
CREATE TABLE ConfigurationFileUpdate
(
    CameraSensorID INT NOT NULL ,

    NewRollResDevicepub_offset decimal(12,6) NULL,
    OldRollResDevicepub_offset decimal(12,6) NULL,
    ChangeDate DATE default getDate(),
    FOREIGN KEY (CameraSensorID) REFERENCES Camera(SensorID)
);
```

Trigger for ConfigurationFile Modifications



```
-- ConfigFileUpdateTrigger
DROP TRIGGER ConfigFileUpdateTrigger
CREATE TRIGGER ConfigFileUpdateTrigger
ON ConfigurationFile
AFTER UPDATE
AS
BEGIN
        DECLARE @CameraSensorID int =  (SELECT SensorID FROM DELETED);
        DECLARE @NewRollResDevicepub_offset decimal(12,6) = (SELECT RollResDevicepub_offset FROM INSERTED);
        DECLARE @OldRollResDevicepub_offset decimal(12,6) = (SELECT RollResDevicepub_offset FROM DELETED);

IF (@NewRollResDevicepub_offset <> @OldRollResDevicepub_offset)
    INSERT INTO ConfigurationFileUpdate( CameraSensorID,
                                        NewRollResDevicepub_offset,
                                        OldRollResDevicepub_offset,
                                        ChangeDate)

    VALUES (
            (SELECT @CameraSensorID FROM INSERTED),
            @NewRollResDevicepub_offset,
            @OldRollResDevicepub_offset,
            GETDATE()
                    );
END;



-- Testing the trigger
--select * from ConfigurationFileUpdate

--UPDATE ConfigurationFile
--SET RollResDevicepub_offset = 9.1234
--WHERE SensorID = 50;

--select * from ConfigurationFileUpdate
```

Followed by the Trigger Test:



```

-- Testing the trigger
--select * from ConfigurationFileUpdate

--UPDATE ConfigurationFile
--SET RollResDevicepub_offset = 9.1234
--WHERE SensorID = 50;

select * from ConfigurationFileUpdate

```

```
CameraSensorID  NewRollResDevicepub_offset  OldRollResDevicepub_offset  ChangeDate
50              9.123400                    7.123400                    2020-06-20
```

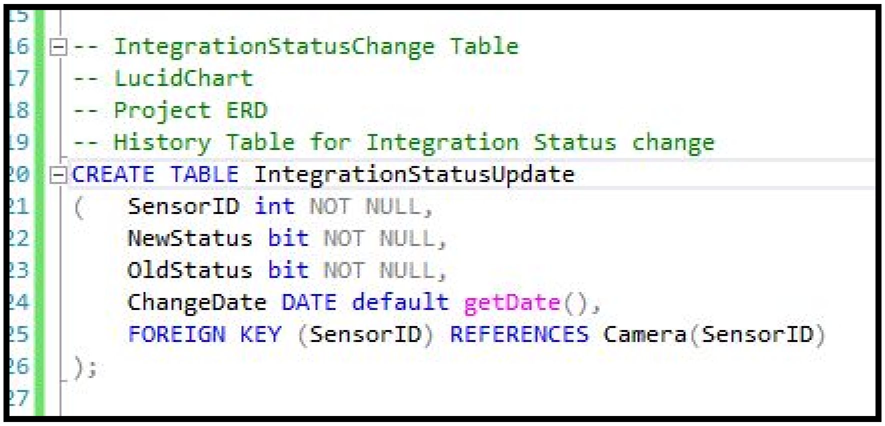
Capturing History of the Database.

One of the major entities that will change quite a bit is the CameraIntegrationStatus table. This table will change at the end of every test as well as after the first flight after delivery.

The Conceptual and Physical ERDs are the same referenced above. The two tables used to capture history are indicated by orange fill..

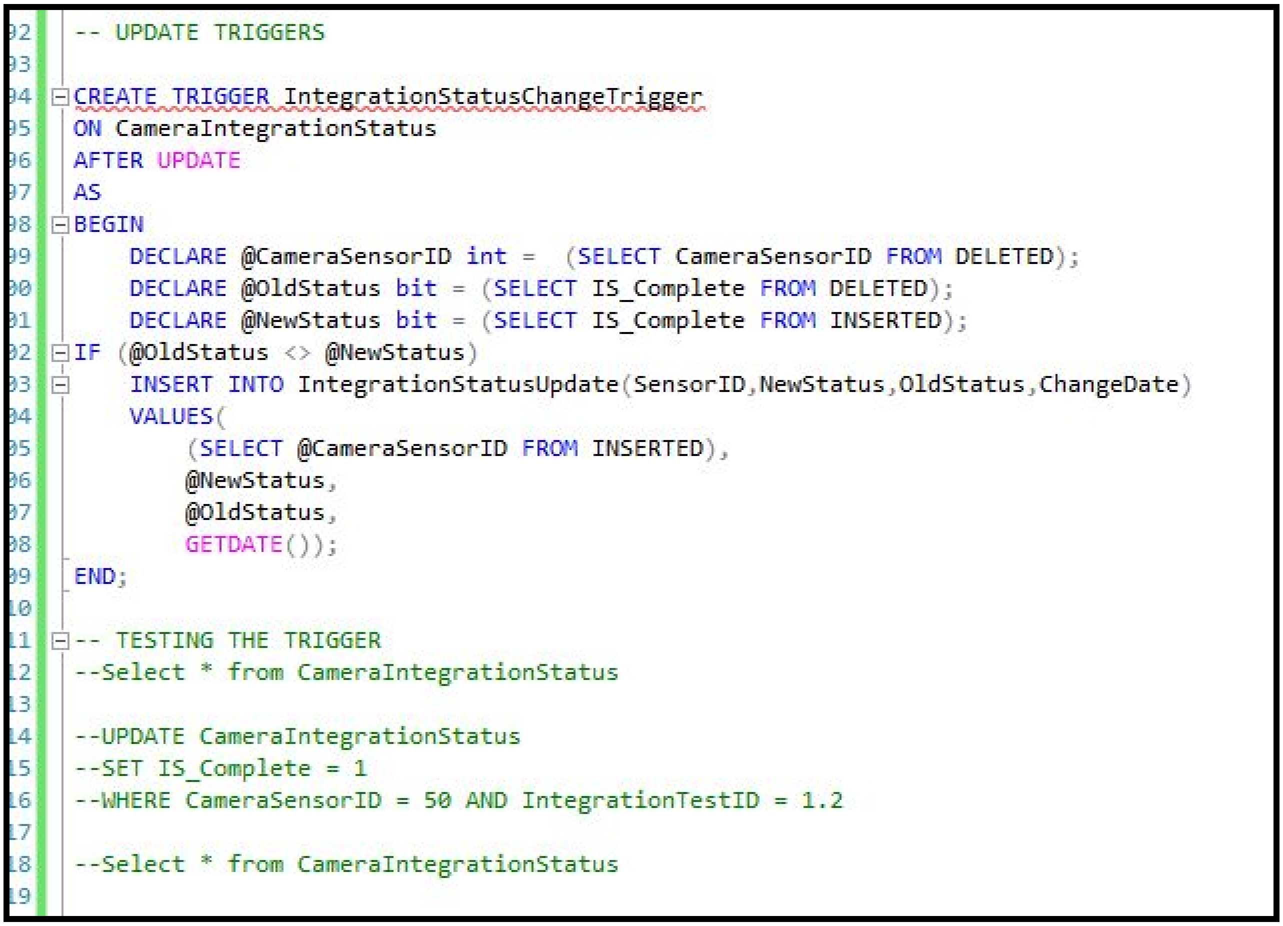
The attributes table:

IntegrationStatus Update	SensorID	INT	SensorID will be a number between 1 - 500
IntegrationStatus Update	NewStatus	BIT	Is Complete Status from CameraIntegrationTest
IntegrationStatus Update	OldStatus	BIT	Is Complete Status from CameraIntegrationTest
IntegrationStatus Update	ChangeDate	Date	Date that the table was updated



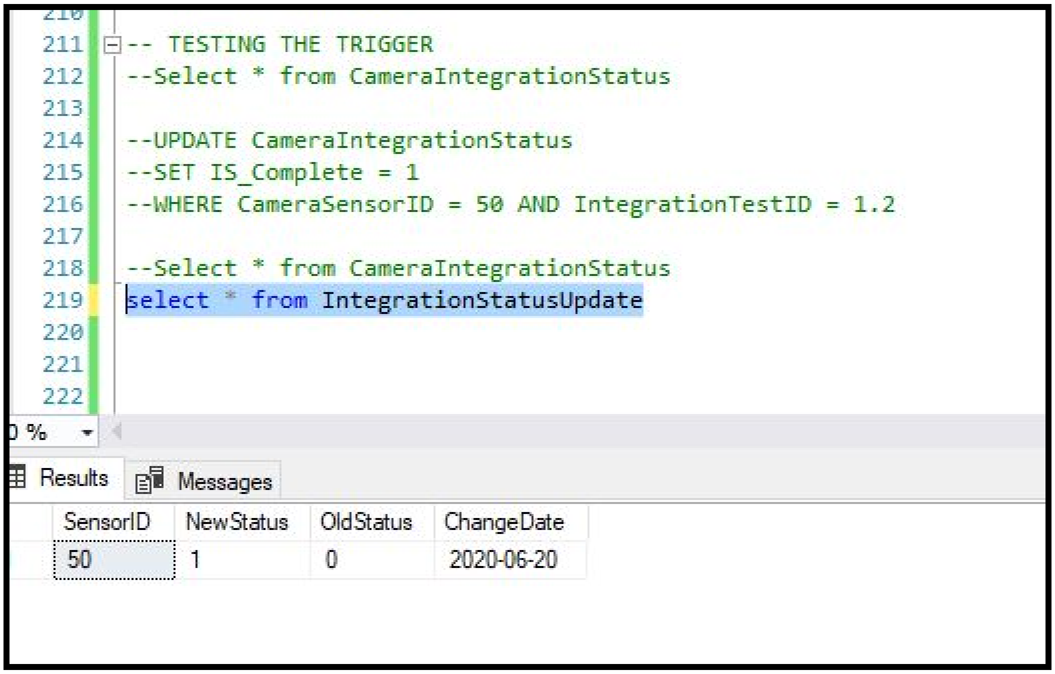
```
-- IntegrationStatusChange Table
-- LucidChart
-- Project ERD
-- History Table for Integration Status change
CREATE TABLE IntegrationStatusUpdate
(   SensorID int NOT NULL,
    NewStatus bit NOT NULL,
    OldStatus bit NOT NULL,
    ChangeDate DATE default getDate(),
    FOREIGN KEY (SensorID) REFERENCES Camera(SensorID)
);
```

Trigger for the IntegrationStatusUpdate table.



```
-- UPDATE TRIGGERS

CREATE TRIGGER IntegrationStatusChangeTrigger
ON CameraIntegrationStatus
AFTER UPDATE
AS
BEGIN
    DECLARE @CameraSensorID int =  (SELECT CameraSensorID FROM DELETED);
    DECLARE @OldStatus bit = (SELECT IS_Complete FROM DELETED);
    DECLARE @NewStatus bit = (SELECT IS_Complete FROM INSERTED);
IF (@OldStatus <> @NewStatus)
    INSERT INTO IntegrationStatusUpdate(SensorID,NewStatus,OldStatus,ChangeDate)
    VALUES(
        (SELECT @CameraSensorID FROM INSERTED),
        @NewStatus,
        @OldStatus,
        GETDATE());
END;

-- TESTING THE TRIGGER
--Select * from CameraIntegrationStatus

--UPDATE CameraIntegrationStatus
--SET IS_Complete = 1
--WHERE CameraSensorID = 50 AND IntegrationTestID = 1.2

--Select * from CameraIntegrationStatus
```

The trigger executed successfully.



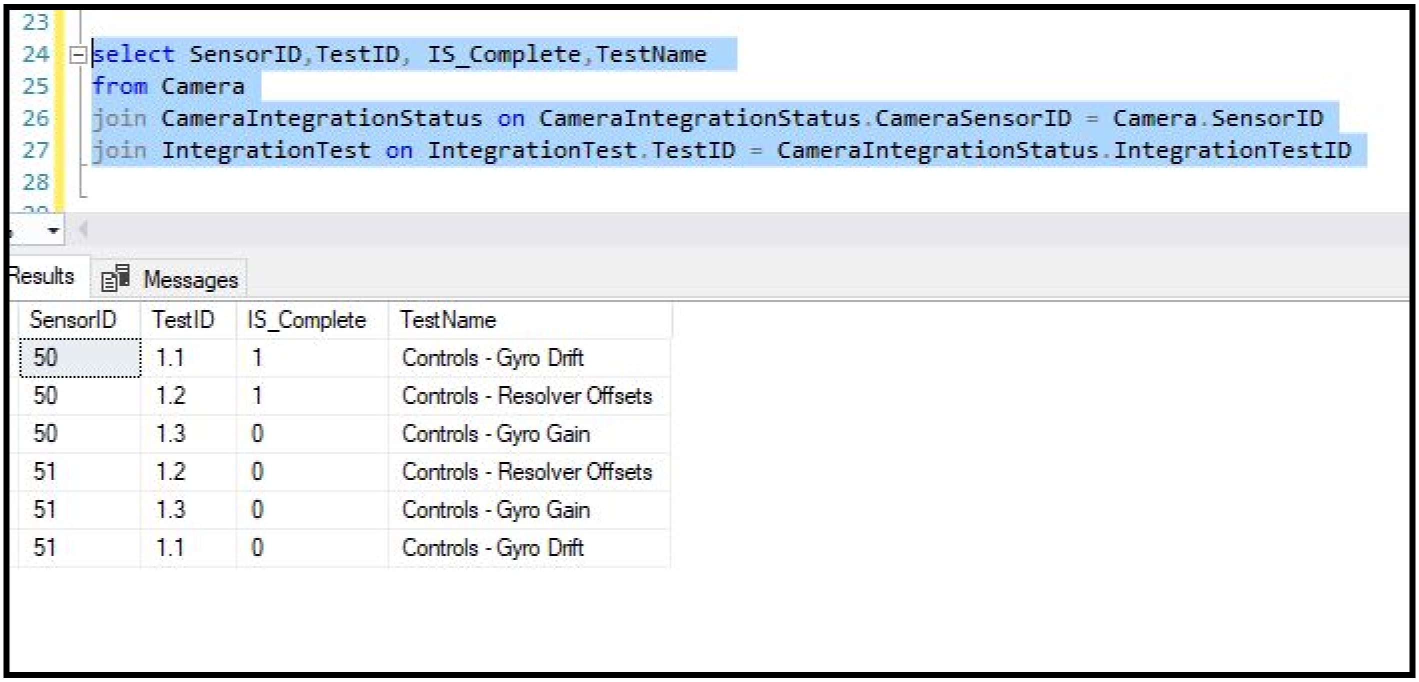
```
-- TESTING THE TRIGGER
--Select * from CameraIntegrationStatus

--UPDATE CameraIntegrationStatus
--SET IS_Complete = 1
--WHERE CameraSensorID = 50 AND IntegrationTestID = 1.2

--Select * from CameraIntegrationStatus
select * from IntegrationStatusUpdate
```

SensorID	NewStatus	OldStatus	ChangeDate
50	1	0	2020-06-20

5.3 Questions and Queries

We would like a daily report of the status of the Camera Sensors as they go through the Integration Test Cycle. By joining two tables - the Camera table and IntegrationTest table, via the CameraIntegrationStatus table, we can see the progress of the Sensor through Integration and Test.

1. Joins of at least two tables



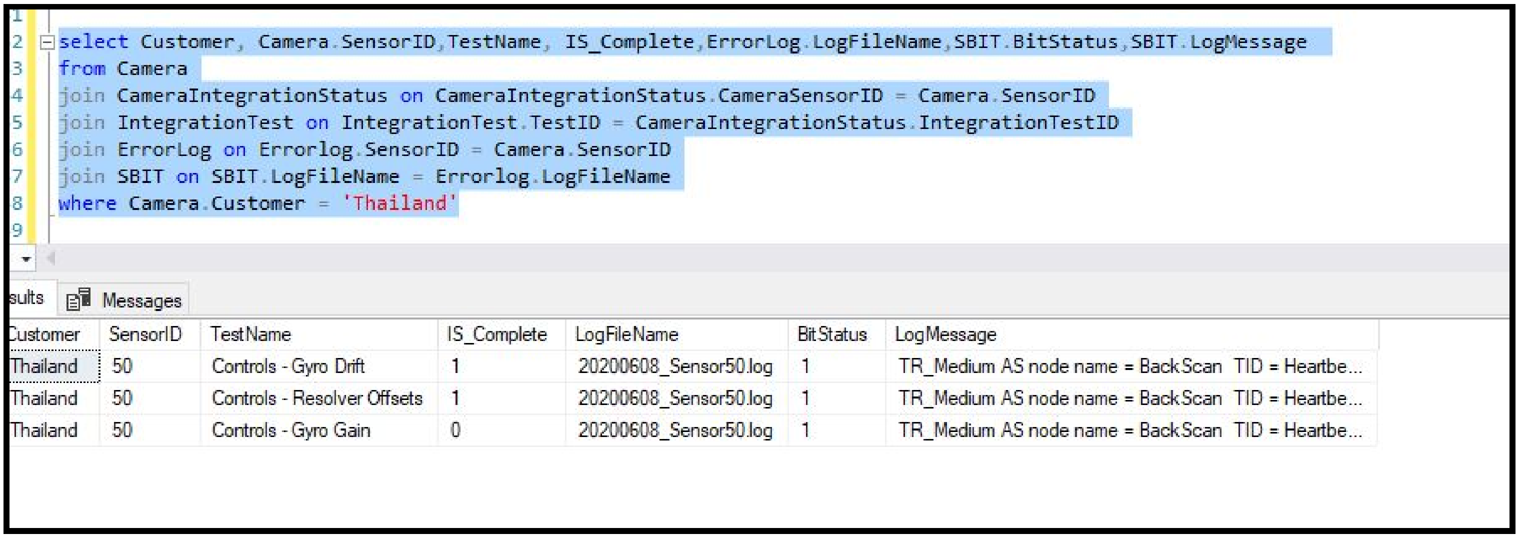
```
select SensorID,TestID, IS_Complete,TestName
from Camera
join CameraIntegrationStatus on CameraIntegrationStatus.CameraSensorID = Camera.SensorID
join IntegrationTest on IntegrationTest.TestID = CameraIntegrationStatus.IntegrationTestID
```

SensorID	TestID	IS_Complete	TestName
50	1.1	1	Controls - Gyro Drift
50	1.2	1	Controls - Resolver Offsets
50	1.3	0	Controls - Gyro Gain
51	1.2	0	Controls - Resolver Offsets
51	1.3	0	Controls - Gyro Gain
51	1.1	0	Controls - Gyro Drift

The team would like to know if our Thailand Customer has had any issues with their Cameras. The Program Managers do not recall what sensors were delivered and have asked for assistance.

2. A Join of four or more tables

Here we are joining the Camera, IntegrationTest, ErrorLog and SBIT log data displaying the SBIT messages.

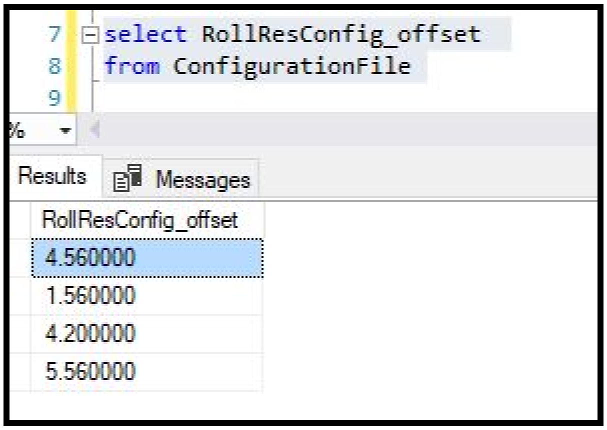


```
select Customer, Camera.SensorID,TestName, IS_Complete,ErrorLog.LogFileName,SBIT.BitStatus,SBIT.LogMessage
from Camera
join CameraIntegrationStatus on CameraIntegrationStatus.CameraSensorID = Camera.SensorID
join IntegrationTest on IntegrationTest.TestID = CameraIntegrationStatus.IntegrationTestID
join ErrorLog on Errorlog.SensorID = Camera.SensorID
join SBIT on SBIT.LogFileName = Errorlog.LogFileName
where Camera.Customer = 'Thailand'
```

Customer	SensorID	TestName	IS_Complete	LogFileName	BitStatus	LogMessage
Thailand	50	Controls - Gyro Drift	1	20200608_Sensor50.log	1	TR_Medium AS node name = BackScan TID = Heartbe...
Thailand	50	Controls - Resolver Offsets	1	20200608_Sensor50.log	1	TR_Medium AS node name = BackScan TID = Heartbe...
Thailand	50	Controls - Gyro Gain	0	20200608_Sensor50.log	1	TR_Medium AS node name = BackScan TID = Heartbe...

3. Another use case that would be very interesting to me is the following:

   I want to look at the Configuration Table and inspect the widgetOffset value. A test engineer just computed this value but is questioning if the value is correct.



```
select RollResConfig_offset
from ConfigurationFile
```

RollResConfig_offset
4.560000
1.560000
4.200000
5.560000

Very easy and trivial to see all the values. The use case is that I would like to find any OUTLIERS in this dataset. What is within +- 3 sigma of the mean? What is normal? What is in family?

I do not have the knowledge of the SQL math functions, but this might be a case where Matlab or Python could be used to connect to the database and quickly find.

I am very happy and very satisfied with this project. I will be able present this preliminary design to my team for a round table discussion and possibly get this fleshed out in more detail. It could very well be given to someone else to pursue, so this documentation is very important to me.

Because I do not have a background in the SQL environment, the file organization aspect was difficult for me, but I settled on my approach.
I have 3 files for the project.

SensorTables.sql: Contains all code to create the tables in the database.
SensorProcedures.sql: Contains all procedures and the two triggers that are implemented in the database. I had issues with my triggers, but I was eventually able to debug them and get them to work correctly.
SensorDB_Data.sql : All of the dummy data used for the Sensor database.
SensorDB_ExampleQueries: Queries used in the questions above.

Regards,
Phillip