Term Project Iteration 3 Walkthrough

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

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.

During this parsing phase, I decided to focus on splitting up each log into three distinct sections, the boot section, SBIT section and mission section if it was available.

Previously - I have parsed these files using Python. I have subsequently moved some of the parsing to Windows Powershell, simply because it is ubiquitous with all Windows computers and I can use it when overseas working with a customer computer that might be limited in ability.

I now have a one pagefile with output that speaks to the relevant points found in the error log but am still missing some important details. 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?

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.

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.TimeSynch	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	Unique filename created from the errorlog	The errorlog must be renamed as a unique entity.

SBIT.filename Plan.filename		This unique filename will be used to identify the parent of the three (Boot, SBIT, Plan) instances generated.
Boot.message SBIT.message Plan.message	Variable length message from the errorlog section	Distinct message from the errorlog 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

Summary and Reflection

As I corrected the use case sections, I can see the database taking form and shape.
Concerns:

1. How am I going to get this data into the database?
   I have created a few powershell functions to parse the error log into three sections. Now I must convert these sections into the specific items listed under each entity. The connections are becoming real to me now. Doing this I realized a few variables I was missing in the various tables such as ‘message’.

Structural Database Rules

Entities:

1. Error Log table
2. Boot table
3. SBIT table
4. Plan table
5. Camera table
6. Configuration table
7. Integration Tests table

Participation
Plurality
An Error log may generate many Boot tables.
A Boot table is generated by one Error log.

An Error log may generate many SBIT tables.
An SBIT table is generated by one Error log.

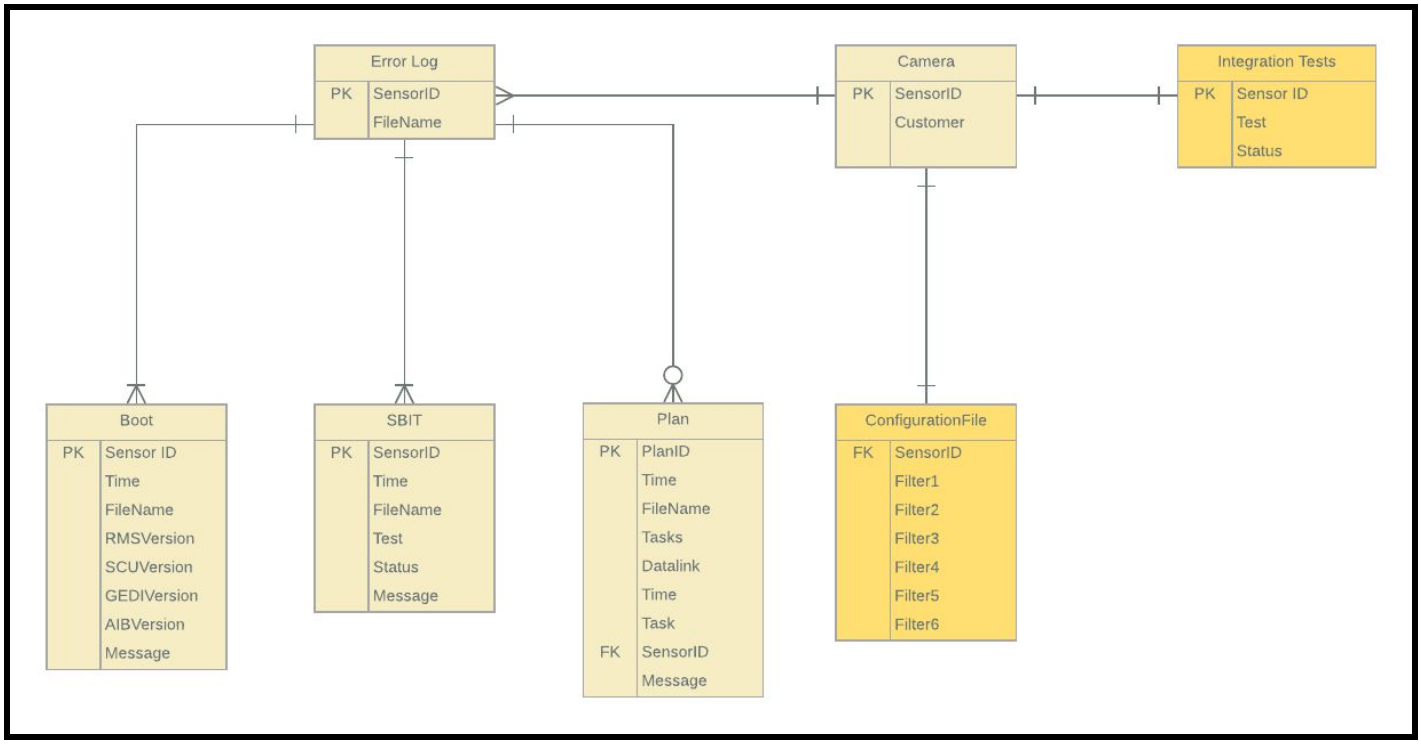
An Error log may generate many or no Plan tables.
A Plan table is generated by one Error log.

A Camera may generate many Error logs.
An Error log is generated by one Camera.

Each Camera has one configuration file.
Each configuration file is for one camera.

Each Camera has one Integration test suite.
Each Test Suite has one camera.

ERD Diagram



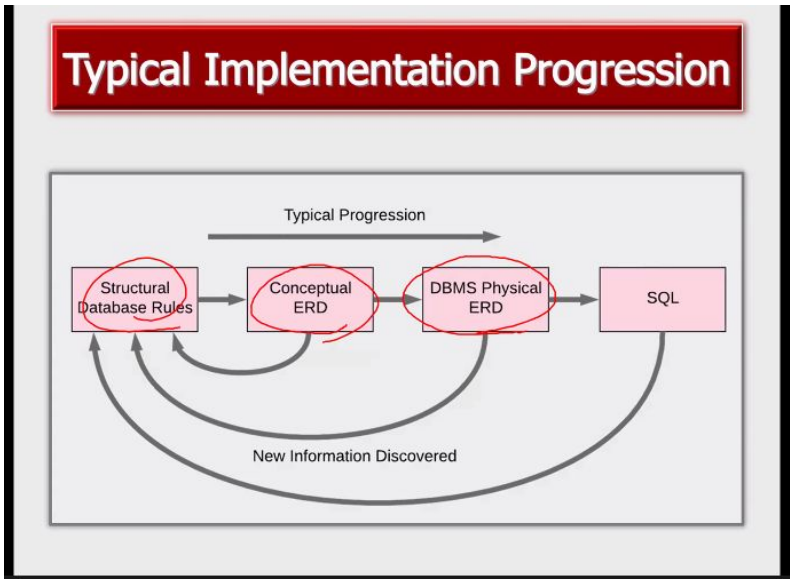
The two yellow entities represent entities that I will attempt to create and add upon if time permits.
While the error log entity typically generates only one Boot Section, and one SBIT section, there are times when the camera system is turned off and back on during operation, and you subsequently have another Boot Section and SBIT section in the same error log, so it was important to capture that.

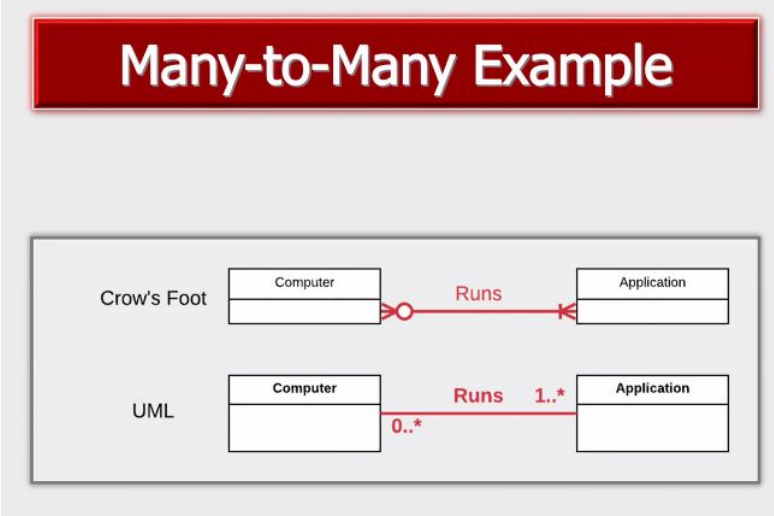
How do i decide if a key should be a primary key or a foreign key

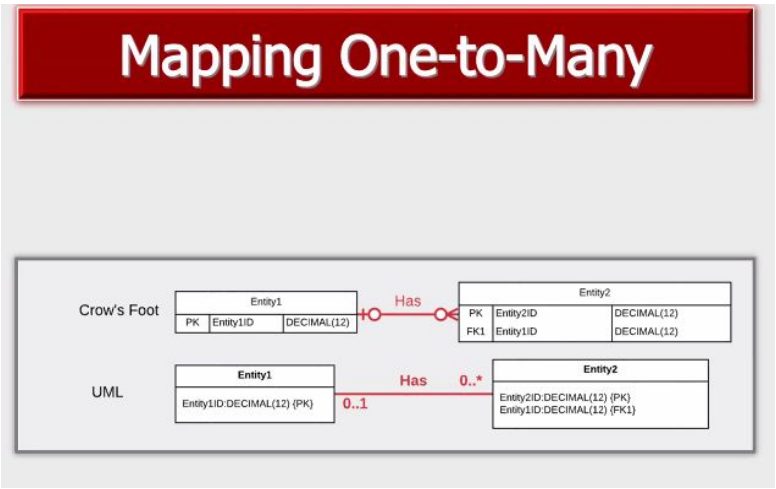
Week 3
Define a
‘Is a’ - Specialization - Generalization
Generalization - ‘Cat’
Specialization - ‘Lion’ or ‘Tiger’
Put this in conceptual first.

Extended ERDs
‘Has as’

So the Error log would be a ‘Super Type’
Look at EERD Symbol Summary



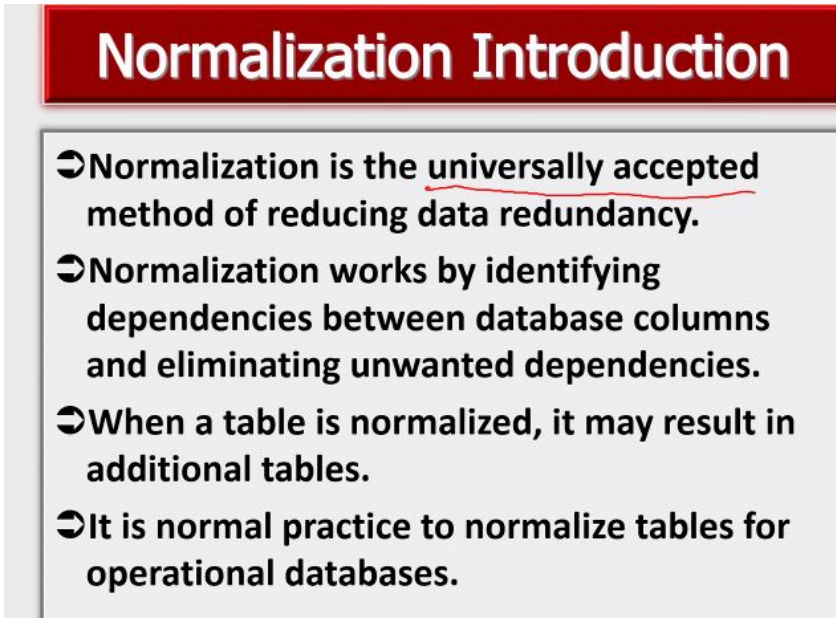


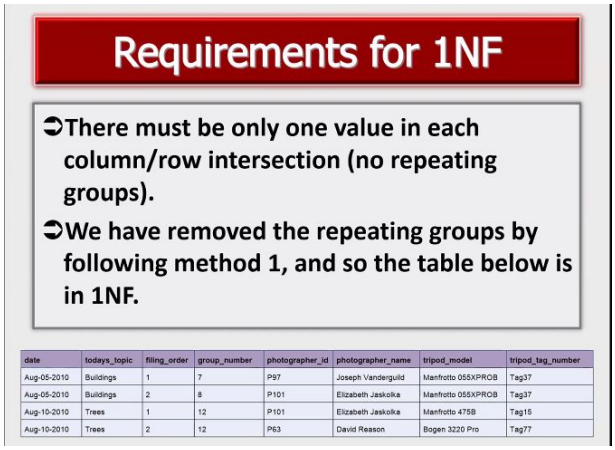


The foreign key must be in the 'MANY entity.

Only three possible patterns:
Mapping
One to One
One to Many
Many to Many

Normalization





Step 3: Identify Functional Dependencies

- ➲date ➔ todays_topics
- ➲date,filing_order ➔ all attributes (candidate key)
- ➲date,tripod_tag_number ➔ tripod_model
- ➲photographer_id ➔ photographer_name

date	todays_topic	filing_order	group_number	photographer_id	photographer_name	tripod_model	tripod_tag_number
Aug-05-2010	Buildings	1	7	P97	Joseph Vanderguild	Manfrotto 055XPROB	Tag37
Aug-05-2010	Buildings	2	8	P101	Elizabeth Jaskolka	Manfrotto 055XPROB	Tag37
Aug-10-2010	Trees	1	12	P101	Elizabeth Jaskolka	Manfrotto 475B	Tag15
Aug-10-2010	Trees	2	12	P63	David Reason	Bogen 3220 Pro	Tag77

Step 4: Identify Candidate Keys

- ➲ We must identify all candidate keys, because the definition of normal forms 2NF, 3NF, and BCNF depend upon this.
- ➲ In this case, we only have the combination of date and filing order as the one candidate key.
- ➲ Note that not all relational tables have a candidate key.
- ➲ If a table does not have a candidate key, once it is in 1NF, it is also in 2NF and 3NF (but not necessarily BCNF).

date	todays_topic	filing_order	group_number	photographer_id	photographer_name	tripod_model	tripod_tag_number
Aug-05-2010	Buildings	1	7	P97	Joseph Vanderguild	Manfrotto 055XPROB	Tag37
		2	8	P101	Elizabeth Jaskolka	Manfrotto 055XPROB	Tag37
Aug-10-2010	Trees	1	12	P101	Elizabeth Jaskolka	Manfrotto 475B Pro	Tag15
		2	12	P63	David Reason	Bogen 3229 Pro	Tag77

Review what Candidate Key is and how to use it. What's the difference

Step 5: Identify Partial Dependencies

- A partial dependency can be of the form:
  A,B➔C,D
  B➔C
- Note that A,B must be a candidate key to qualify as a "partial" dependency. If determinant is not a candidate key, 2NF is not concerned with it.
- The table contains a partial dependency:
  date,filing_order➔all attributes
  date ➔ todays_topic
- This partial dependency means that once we know the date, we also know that day's topic.

date	todays_topic	filing_order	group_number	photographer_id	photographer_name	tripod_model	tripod_tag_number
Aug-05-2010	Buildings	1	7	P97	Joseph Vanderguild	Manfrotto 055XPROB	Tag37
Aug-05-2010	Buildings	2	8	P101	Elizabeth Jaskolka	Manfrotto 055XPROB	Tag37
Aug-10-2010	Trees	1	12	P101	Elizabeth Jaskolka	Manfrotto 475B	Tag15
Aug-10-2010	Trees	2	12	P63	David Reason	Bogen 3220 Pro	Tag77

Step 6: Removing Partial Dependencies

- A new table is created which contains the determinant and the determined attributes.
- The determinant also remains in the original table, but the determined attributes do not.
- In this example, since date ➔ todays_topic, date and todays_topic are moved into their own table, and date remains in the original table.

date	todays_topic
Aug-05-2010	Buildings
Aug-10-2010	Trees

date	filing_order	group_number	photographer_id	photographer_name	tripod_model	tripod_tag_number
Aug-05-2010	1	7	P97	Joseph Vanderguild	Manfrotto 055XPROB	Tag37
Aug-05-2010	2	8	P101	Elizabeth Jaskolka	Manfrotto 055XPROB	Tag37
Aug-10-2010	1	12	P101	Elizabeth Jaskolka	Manfrotto 475B	Tag15
Aug-10-2010	2	12	P63	David Reason	Bogen 3220 Pro	Tag77

Now the table is 2NF.

Step 7: Identify Transitive Dependencies

- In a transitive dependency, A → B → C.
- Tables with transitive dependencies may still suffer from update anomalies.
- Example:
  - If it rains, the ground is wet.
  - If the ground is wet, it is muddy.
  - RAIN → WET_GROUND →MUD

Step 8: Remove Transitive Dependencies

- Recall the functional dependency: photographer_id → photographer_name
- The transitive dependency in this example is: date,filing_order → photographer_id → photographer_name

Transitive Dependencies Removed

date	todays_topic
Aug-05-2010	Buildings
Aug-10-2010	Trees

photographer_id	photographer_name
P97	Joseph Vanderguild
P101	Elizabeth Jaskolka
P63	David Reason

date	filing_order	group_number	photographer_id	tripod_model	tripod_tag_number
Aug-05-2010	1	7	P97	Manfrotto 055XPROB	Tag37
Aug-05-2010	2	8	P101	Manfrotto 055XPROB	Tag37
Aug-10-2010	1	12	P101	Manfrotto 475B	Tag15
Aug-10-2010	2	12	P63	Bogen 3220 Pro	Tag77

Now two lookup tables have been created.. Date / Topic & Photographer ID / Name

Table is now in 3NF.

Requirements of 3NF

➲A table in 3NF must first meet all requirements for 2NF.

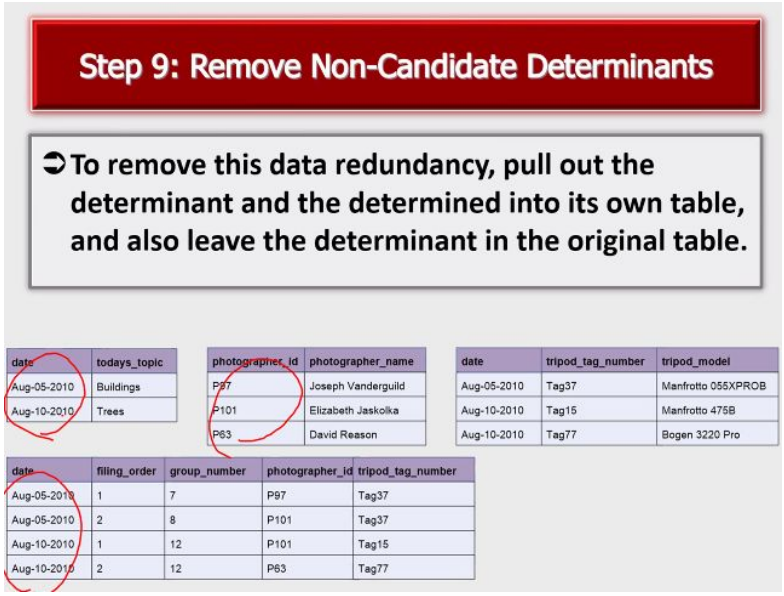
Step 9: Identify Non-Candidate Determinants

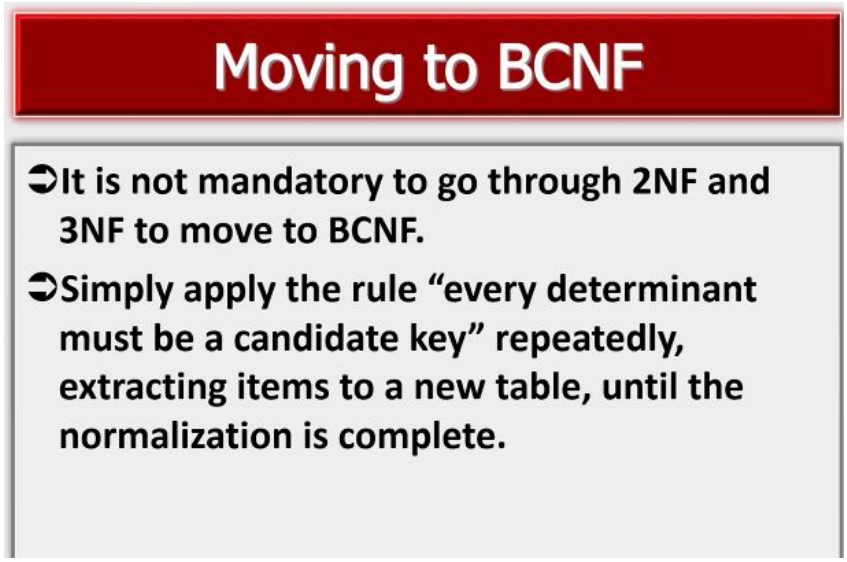
➲ This example has a determinate which is not a candidate key:
date,tripod_tag_number ➔ tripod_model

date	todays_topic
Aug-05-2010	Buildings
Aug-10-2010	Trees

photographer_id	photographer_name
P97	Joseph Vanderguild
P101	Elizabeth Jaskolka
P63	David Reason

date	filing_order	group_number	photographer_id	tripod_model	tripod_tag_number
Aug-05-2010	1	7	P97	Manfrotto 055XPROB	Tag37
Aug-05-2010	2	8	P101	Manfrotto 055XPROB	Tag37
Aug-10-2010	1	12	P101	Manfrotto 475B	Tag15
Aug-10-2010	2	12	P63	Bogen 3220 Pro	Tag77





To find a determinant - A -> (DETERMINES) B

Like the

Always just go to the BCNF and normalize this.

WATCH the VIDEO AGAIN - DOWNLOAD AND WATCH!!
WWWWWWW