Databasing Report

# Data Cleansing

## The Process and Importance of Data Cleansing

* Data must be separated out and assigned to the most relevant tables at first.
* The most important part of keeping the database data clean is choosing correct ways to store the data. Integers are inexpensive and less taxing then VARCHAR and DECIMAL Values and will help keep the overhead low.
* In terms of relevancy, all the data in this assignment log files were important. However, in industry practice, potential culling of excess data could occur if more time was able to be spent with the clients to further refine requirements of the database and reduce overhead for efficiency.

## Next Steps with Data Cleansing

* Further data cleansing that could be implemented, is the splitting of arrays (like rawArray) into a more appropriate data type like JSON. However, in order to meet the time deadline this workflow had to be omitted.

# Choice of entities

## Choice of Tables

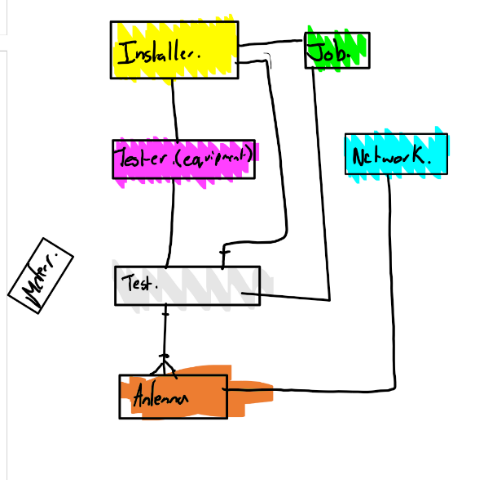
* I started with a noun analysis of the requirement document sent out in the assignment files. As a rule, nouns in the requirement brief tend to end up becoming an entity in the ERD. At first, I ended up with the following (very) basic diagram of potential tables (fig 1).
* After this, it was a matter of analysing the data files and finding avenues of normalisation and reduction.
* During this process, I came up with a list of the following ten tables:
  + Installer
  + Job
  + Meter
  + Network
  + Antenna
  + Signal Type
  + Result
  + Signal Result
  + Test Equipment

Figure Noun Analysis Table Generation

* + Test
* Many of these tables are used to form as a way of reducing redundancy of data. E.g. Per each test, many results could be generated and so on.
* Job was split out from test, as I concluded that many tests could be conducted for a specific work order.

# Choice of attributes

## Attribute Usage and Types

* Attributes where determined as a result of relevancy to specific parts of the test log files. All the lines in the log file have been split out to form an attribute in some form or another. For example, in order to achieve minimal redundancy (maximum normalisation), a lot of test output have been split out to form over several results table. In order to get a full idea of the results, tests must be read in context to the other entities (with foreign keys).
* All my attributes are simple in terms of datatypes. I have used VARCHAR for variable length data, Char for fixed length data (such as 4 letter acronyms), Date/Time for the time of the test, Auto Incrementing Integers for ID Keys and Fixed length Decimal for Float input

## Next Steps

* Some datatypes could be adjusted for better storage types. For example, raw arrays could be adjusted into a JSON format when further details about the arrays would become available.

# Choice of keys

## Key types

* For most of the database, I implemented a new primary unique identifier to reduced redundancy. These primary keys are Auto-Incrementing Integers and are unique to avoid key conflicts.
* These types of primary keys are useful as it makes entry of the field less likely to duplicate/quicker (rather than writing out a whole load of text). It also makes foreign key entry easier.
* The only exception to this was the use of work Order as the primary key. As we there was only one attribute that fitted in this table and the table could be future expanded, I opted to use the work order as the primary key.
* *(For more info on the foreign keys, please see connectivity/relationships)*

# Connectivity/Relationships

## Logical flow of data

* When deciding my relationships between tables, I used verbal statements to help me decide the type of relationship that would be required and to check my logic. For example:
  + *“A job may have many tests”*
  + *“A test will have one or many results”*
* After these relationships have been formed the interlinking foreign key references can also be formed. These are built in as an extra attribute in the table.
* These can then be translated onto the ERD easily and effectively. The logic can be further checked by building the ERD in the workbench model editor. I found through this several of my keys were in the wrong place and was able to rectify them as such.
* Another thing to keep in mind was the use of the correct datatype for these foreign keys, so that they would not throw errors when inserting the data.

# Use of composite/bridging entities

* A large proportion of my database is built around bridging tables.
* Bridging tables typically form between “Many to Many” Relationships.
* The “Test” table is a good example of one of them. While it serves as a storage place for data, it also interlinks a very complex network of tables: Job, Meter, Installer & Test Equipment via the use of foreign keys.
* Another example of this is “Results”. It interlinks the following tables: Network, Signal Results & Antenna via the use of foreign keys.
* Bridging tables/entities help reduce the complexity of a database network and stop unnecessary overhead of the host server as it tries to compute a more complex relationship network.
* It also makes queries easier, as you can target the bridging table rather than several tables.

# Extent of normalisation

## Extent Used

* The use of normalisation was very important in this Database as there is a large abundance of redundant information in each log file. Repeating this information increases overhead of a relational database and is normally mitigated using foreign keys relating to records in other tables.
* The extent of normalisation increased dramatically over the course of the assignment. I began with 6-7 Tables and then moved up to 10 as I found that data entry could be reduced and made more efficient with the use of more tables.

## Next Steps

* A fine balance needs to be drawn between normalisation and complexity. There is potential for further normalisation, but a line needs to be drawn where data entry becomes too complex. However, an element of reducing this complexity could be the use of a front-end form.

# Other “interesting” aspects of the design

* An interesting aspect of the design was the automation and design process. I mapped out a basic structure of my ERD in Visio but moved towards building it in the model editor in MySQL Workbench. Not only did this reveal potential flaws in my design logic by providing the tools to analyse and allow me to fix them (for example, the reverse of foreign key placement), but also provided a design structure for the implementation of my attributes, and their properties. After This, it also allowed me to generate a script to create the database. After auditing this, and ensured everything worked properly, data could be inserted.
  + This Automation allows for a lot of time saving, and for the administrator to focus on the design stage, rather than the tedious approach of coding and then testing later.
* Another quirky/ interesting design aspect was cardinality and multiplicity. I realised that some of my designed entities were required in the context of the test, but not in the context of a real-life relationship. For example, a meter AMS may have bought may never have been used for a test. I have allowed for the expansion of this further by introducing that multiplicity does not have to equal at least one for certain objects. These objects are generally inventory (physical) items, not abstract items. Consistency between these

# Version Control

## What Did I Use for Version Control?

* I used version control extensively throughout this assignment. I relied on retiring my files manually as they became irrelevant or outdated.
* I also used a GitHub repository as I completed elements of my assignment for tracking purposes. Repositories are useful as they can allow for multiple users to collaborate without worrying about file conflicts (conflict resolution), as well as the ability to revert to any previous repository, and to compare/mark-up changes to files.
* Lastly, I used OneDrive. This contains implemented version control and ensured my files were available across devices as I needed them.

## Why?

* Version control allows for the roll back of certain files that are incorrect or break something, while also freeing up most files from duplicates. This keeps the project file system slim and free of clutter.