Works management system preliminary design technical specification

# Introduction – high level overview

The works management system being designed is a project management system to fit the existing workflow of Northpowers Westcoast Energy and ultimately Northpowers Perth branch.

The system currently being used follows the pattern of a WBS (Work Breakdown Structure). There are currently 3 levels, the first level is the project (job), the second level is the task, and the 3rd level is the duty.

*Terms in bold in this section are table names in the design*

A **client** may have several different **projects** planned, planned, active, or complete.

A **task** (formerly sub-task) is the smallest unit of work being produced. These get allocated to crews on a **day** by day basis and require a **reschedule** when not completed on time. A **crew** can receive several tasks in one day.

A **duty** is a responsibility for a **staff** member to check off in the system and relates to a specific task. This often means that they are responsible for overseeing or organizing something. A duty often carries a lead in time, which is the number of days prior to start that the duty needs to be completed by. Sometimes additional information may be passed or may require to be collected associated to a duty.

**Resource**s are allocated either against a specific task or for a crew or overall for the day. Sometimes as in the case of external contractors it can be the duty of some member of staff to arrange this.

Clients may have different project types.

Currently the important roles played within the system in the Melbourne business, are Project Manager, Field Manager, Administrator, Scheduler, Person in Charge of Crew (foreman?). It is likely that Perth follows the same structure however it is possible for other roles to be introduced.

Different clients may have different requirements for the same project type. Due to this duties may differ by project type and by client and may potentially be assigned to different staff.

These differing requirements require a system that favors configurability over simplicity i.e. it must be generic.

# Design of generic type to suit projects, tasks, and duties

## Overview

This means a general-purpose attribute/column/field that can be attached to a table/form/view as determined by the system administrator.

The meaning of generic here is that it should be able to accommodate different database data types for whole numbers, real numbers, dates, times, text, potentially images or other media such as video in the future (maybe multiple images).

This not only refers to the storage of these within the database but means different widget types as suitable within the view/form.

## Database

The project table could have any number of generic type fields appended for any given project depending on the client project type. The data will be stored in the generic\_task table

The task table mirrors the project table implementation with regards to this except that the data will be stored in the generic\_project table.

The duty table can have only one generic field optionally assigned hence at this stage will store the data in the duty table rather than creating a separate generic\_duty table.

## Widgets

The widget that the user interacts with to enter or select the data values for the generic columns from within the relevant view/forms, will be automatically chosen by the application dependent on the data type and restrictions set by the system admin.

1. Date picker for dates
2. Not sure yet for time
3. Text box for freeform text
4. Drop down list box if less than 20 values, if more than 20 selectable values then autocomplete drop down list which will display a drop down list of up to 10 values that match what the user is typing as they type so they can either type or select
5. File selector for images

## Operation

1. A user who has a role of system administrator is able to create system wide re-usable widgets in the correct create view/form accessible from home page main menu A widget will choose its own type within. Various
   1. **Select the type of data to be stored in the database.**

This is important when querying, for example if dates are stored as text then you cant ask the database to show all records between certain dates etc.

* + 1. INT – whole numbers i.e. no decimal point
       1. Specify any minimum restriction
       2. Specify any maximum restriction
    2. FLOAT real numbers i.e. has a decimal point
       1. Specify any minimum restriction
       2. Specify any maximum restriction
       3. Specify number of decimal places
    3. TIME for valid entries for SQL for time and date see http://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html
       1. Specify any minimum restriction in SQL
       2. Specify any maximum restriction in SQL
    4. DATE
       1. Specify any minimum restriction in SQL
       2. Specify any maximum restriction in SQL
    5. TEXT
  1. **Select the validation type**
     1. None
     2. Value List
        1. This will allow direct entry or import from excel of list of acceptable values. This will create a new table in the database that can be accessed through the application. This table will be two columns, an id column and the actual data column which will be of the specified type.
        2. Check whether allow new entries or restrict to
     3. SQL Select
        1. This will allow an SQL SELECT query on the entire database with any level of complexity.
        2. Check whether allow new entries or restrict to list.
     4. PCRE (Perl compatible regular expression = pattern matching) This can enforce that any data directly entered matches a particular pattern, for example if a client uses job numbers that are 3 alpha characters followed by 5 digits then this will force user input to conform to the given pattern. There are plenty of prewritten patterns that that you copy via google e.g. for validation email address and phone numbers etc or you can write your own and them first using one of the freely available online regular expression testers.
  2. Check whether mandatory or not

1. The system admin can then assign these widgets
   1. To a specific duty type relevant to client job type and staff roll in the duty type view/form
   2. To tasks relevant to client and job type in the generic task view/form
   3. To a project, relevant to client and job type in the generic project view/form

## Categorization resources and generic attributes for the duty, task, and project tables

The purpose of categorization of course is to group similarly related items together in views/forms and reports, for easy comprehension.

Categorization is easiest created and managed visually by system admin in a tree e.g. <http://libkal.gr/yii_lab/categorydemo#>

# Design of history/audit trail/logging

## Overview

It is important to be able to simulate the“versioning” aspect of the existing excel based system. Once the project is actioned and the initial plan date has been set, then various critical dates are derived and it should not be possible to alter various aspects i.e. the critical date unless the use is a system admin.

People in their given roles still need to be able to check off their duties etc.

Really this means that no date fields should be able to be altered. This could be called issuing or versioning or perhaps another name.

It is also required to be able to keep an audit history of who made what changes when.

## Database

By using a datetime field called version or issued in the project table, the application can block any changes from in crucial areas if this flag is set.

This flag can be used in junction with the logging facility about to be described to ultimately keep track of different versions should the system admin clear this flag and allow a new version to be issued.

Add a staffid column to each table in the database and have a model base class append this this on all create, and updates.

Make an exact copy of the database called log but remove all foreign keys, replace primary keys and unique constraints with non unique indexes.

Add a logged TIMESTAMP column to each log table that has DEFAULT CURRENT\_TIMESTAMP.

Add a logdelete bool flag to each table with default 0.

Use triggers after update, insert and delete to insert a copy of insert, updated or deleted row into the associated log table. The delete trigger should set a deleted flag

The logging operation will be taken care of within the database itself using triggers on update insert and delete.

After a delete action the application code in the model base class should follow with updating the loguser value to the current userid

It will then be possible to view the database through the application at a particular point in time. This may require a little bit of trickery but the main application code will not need to change. When selecting by primary key only the most recent row in the log table should be returned instead of the main database. When selecting from non primary and non unique then only the latest update of each match should be returned.

Due to this, it may be possible to implement undo/redo operations also though need to ensure for each undo and redo that there are no conflicts with other users simultaneous operations, however due to this potentially becoming confusing and conflicting in a multi-user environment then this will not be added at this stage.

# Design of access control

There are two types of roles to consider when designing the Role Based Access Control System.

1. Staff rolls e.g. admin, project manager, field manger, line mechanic etc.
2. Application roles
   1. System admin
   2. Project manager
   3. Scheduler
   4. User

AuthItem contains all roles (both Staff and application). AuthAssignment is the pivot table that asigns (staff) to a role.

A user (member of staff )can belong to many different roles.

In order to be as intuitive as possible different roles should be able to access different part of the system.

In the EER diagram, colors indicate application roles

1. red – no users have direct access to these
2. orange – system admin
   1. create
   2. update
   3. delete
3. green - system admin – cannot alter application roles
   1. create - business rule of return FALSE against application roles
   2. update - business rule of return FALSE against application roles
   3. delete - business rule of return FALSE against application roles
4. yellow –project manager
   1. create
   2. update - business rule to enforce must be a project that this user created
   3. delete - business rule to enforce must be a project that this user created
5. brown – the scheduler
   1. create
   2. update
   3. delete
6. blue – user - default
   1. update - business rule to enforce this user must be assigned this duty

Each role defined above inherits the roles below it, that is the role above is a parent of the role below.

All authorized users are able review clients, projects, and tasks.

Various reports will need to be created including something equivalent to daily brief, and planning sheet. Reports will also have access control applied in a similar manner to above.

## Application notes

Delete functionality requires special handling. In assembly, and client\_to\_task\_type\_to\_duty\_type pivot tables, because the table contains some fields that are non primary key foreign keys, editing of those fields could corrupt historical information hence the deleted flag is replaced with and ai id field which should be used by as the referenced column from foreign keys.

Would like to apply this also to AuthAssignment but may need to alter some other classes to cope with this so have delayed for now. Another options would be to store historical AuthAssignment records in another table and have a pivot table to project from this instead of AuthAssignment – fk on id. This would mean no crud for AuthAssignment but maybe for AuthAssignmentHistory. AuthAssignmentHistory to be maintained by triggers like logging.

The next level of delete functionality is simple, if access is granted then attempt delete. If a foreign key blocks because of NO ACTION or RESTRICT then mark as deleted.

Application code should be careful to return values when searching by primary key where deleted is 0 except for those tables above where the last

Another issue in this is system admin altering values in these tables e.g. the assembly table quantity – what if the quantity of an item is changed at a few date, then all related past items would also show an increase which isn’t correct. The state should be what was created at the time. Though we want to block duplicates in those tables there may be a case for an increment field to be part of the primary key hence allowing duplicates. Related tables that aren’t that don’t form part of the primary key that link e.g. assembly\_to\_task should link to the auto increment primary key. I think this is the full proof solution!

In CModel if one of the orange tables then only mark as deleted if there are existing related records. Also don’t retriver

## First normal violations in database design

Numerous tables technically violate first normal due to allowing nulls in some columns, however the attributes are truly optional or perhaps are for later entry hence my feeling are that these are correctly normalized and due to the debatable nature of Nulls in normalization it could be argued that nearly all tables are 3rd normal (apart from null violations).

The generic table badly violates this but unfortunately databases don’t easily support polymorphism. Originally this table was further normalized however concerns were that if user wanted to alter a generic type from say int to float etc then this would be more difficult to achieve in the application – hence this small violation is for application simplicity and realistically is the more practical way to do it.