The “infra” library

One interesting this about being a small developer who also does the occasional contractor assignment is that one gets exposed to a wide variety of solutions. This includes techniques and methods for particular business issues. Or developer issues as the case may be.

The “infra” library represents a collection of the best generic ideas, facilities, techniques and methods that have been collected throughout the last 30 years of my software career.

“infra” requires a companion library, the business application library, it order for it to “come alive”. It does not do anything on its own.

The critical thing about the “infra” library is that it:
a. accompanies ALL of our business applications
b. it must be opened BEFORE the specific application library

We use a “LAUNCHER” library (available in git) in our startup folder that refers to a .json file to determine which libraries get opened, and in which order, every time Omnis is opened. This LAUNCHER always opens “infra” and any other common libraries we use. The developer then opens the application library he/she is working on today. We support about 10 different application libraries nearly all of which depend on “infra”. One neat thing about Omnis is that we can have multiple business libraries open and running with just a single copy of “infra” open and everything works.

“infra” anticipates operating in a variety of architectures - fat client, ultra thin server, remote form server, web services server.

Prior to the ability of a JSON export of Omnis code we used the Omnis VCS as our code repository. On the whole we liked the VCS very much but it had some annoying bugs for our more sophisticated needs and we were keen to move on. One neat feature of the VCS that was used extensively was the sharing of classes to other projects. We had our common class base which was extended to any new project (this was the precursor to infra). However the transition to JSON structured repositories such as git or SVN had no equivalent function. So we were compelled to create infra as a distinct library that ran as a super-companion to each of our application libraries. We are much happier with this arrangement.

An introduction to tkSuper and rtSuper

These two classes are virtually identical. There has to be a copy of each because Omnis does not allow inheritance across class types. The frequently used items here are:

Variables

tlDatabases	contents determined from $Read_db_config if list has multiple lines then the list is display on the login page for user to select from e.g. live, test or dev databases.
trCurrUser	copy of logged in users “uausers” record
trGroupOrg	copy of entgrouporg record - the legal entity owning this data
tcCurrentLib	name of application library - set in Startup_task of app library
tConstants	a row with application specific values that tend to be static
toValues	access to a frequently used object class
toStrings	access to a frequently used object class
trFormColors	row with colours to use in remote forms giving each groupOrg a distinct colour theme
tFormSession	session object for a single database connection for user e.g. fat client
tFormStatement	statement object to accompany tFormSession

If you run with session pools instead of a dedicated tFormSession, that is fine. tMaster will cope with that.

Frequently accessed methods & functions

$DefineList	use instead of $definefromsqlclass - designed to cope with multi-library applications
$Loop	when looping through a list - set List.$line = 0 first supports looping through selected lines only
$Read_db_config	reads sqlite ini file on Startup to get db connection info - tlDatabases
$DynamicSession	if you don’t have a task based session, gets session from session pool
$LogEvent	record when something happens in application
$LogError	record when an error occurs - this would typically be emailed to alert support
$AppError	when an error in code execution is detected send message here
$ServerTime	gets timestamp from database server
$AboutNow	returns last value returned from $ServerTime, updated from server every 5 seconds

The Startup_Task in the application library must be subclassed from infra.tkSuper or some other subclass of infra.tkSuper.

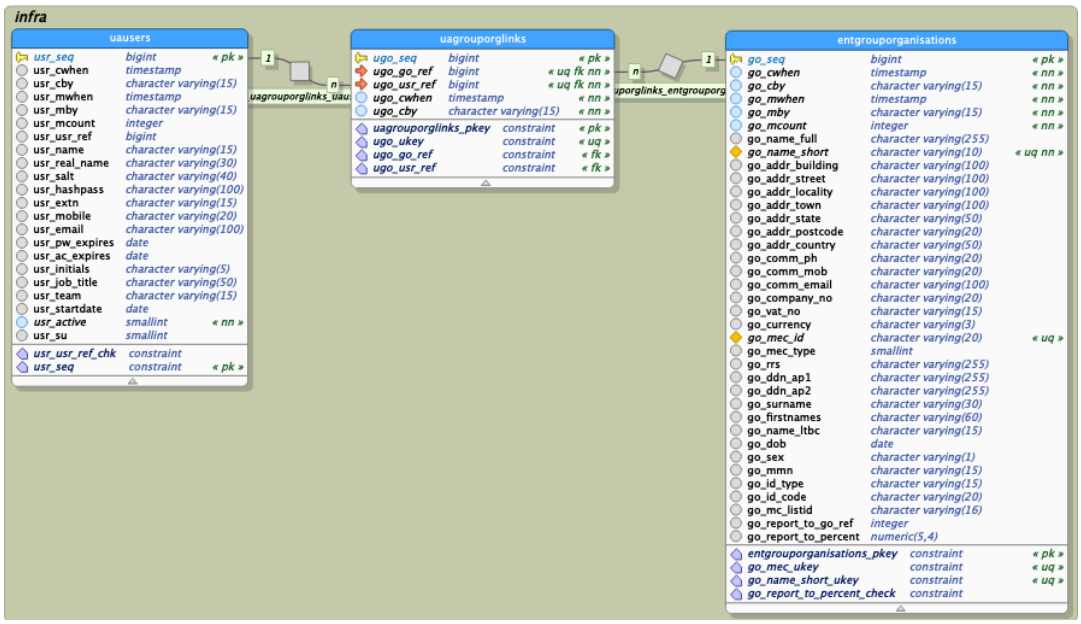
An Introduction to the “infra” library.

For an application to run using the “infra” library a number of critical variables have to be understood.

trCurrUser upon login, stores the users “uausers” record of the verified user

trGroupOrg upon login, refer to uagrouporglinks to see which companies this user has access to. At least one must exist. If only one company then read and store the entgrouporg record in trGroupOrg. If there is more than one company the user is taken to a screen to select which company is being worked on and the selected company record is stored in trGroupOrg.

tcCurrentLib Not set in “infra” This is set within the Startup_Task of the applicable library.



entgrouporganisations

Function: Company definition record

Purpose: Multiple group organisation records support a group of interrelated companies.

Structure: This is applied to a number of key tables that represent a connection between the company and a key resource - typically records that would conventionally have no foreign keys, a foreign key to the group company, GO_SEQ, is established. This is the basis of a multi-tenanted database.

N.B. This isn’t to host thousands of businesses on one db, but to host a group of companies & subsidiaries within a single installation.

Within the “infra” library this is:

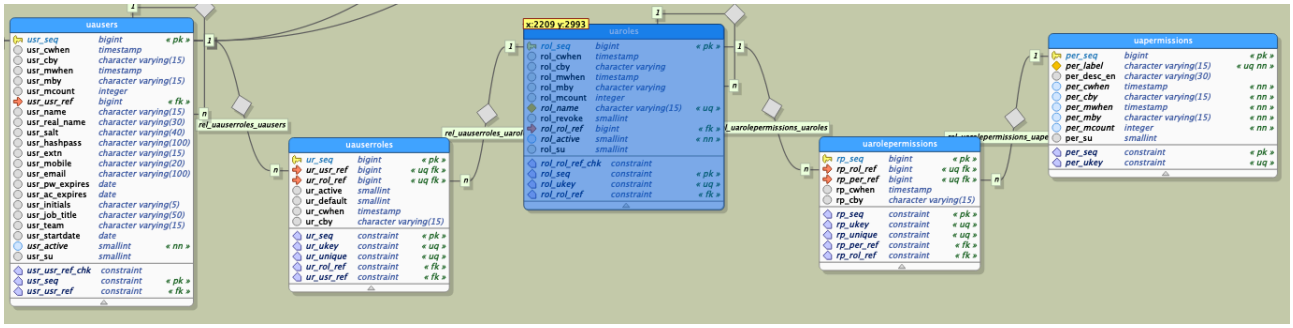
infra tables with a foreign key to entgrouporganisations

schema	table	foreign key
infra	accountingperiodlinks	acl_go_ref
infra	entgrouporganisations	go_report_to_go_ref
infra	entgrouporgnames	gon_go_ref
infra	entgrouporgstructure	gs_go_ref
infra	entorglinks	eol_go_ref

schema	table	foreign key
infra	fininvoiceouth	foh_go_ref
infra	import template	imt_go_ref
infra	sysasyncemails	ae_go_ref
infra	sysreferenceorg	rfo_go_ref
infra	sysreferenceuser	rfu_go_ref
infra	systaskstats	sts_go_ref
infra	uagrouporglinks	ugo_go_ref

users, roles and permissions

Function: User access
Purpose: Complete control over what users can/cannot do through the user interface.
Structure: 5 tables auUsers, uaRoles, uaPermissions and links tables between the three.



For each access element in the user interface, create a Permissions entry e.g. select event, insert event, update event, delete event are 4 distinct Permissions. Add additional records for workflow controls e.g. change from “enquiry” to “offer”.

Think about the roles you want to create that get assigned to users. Link specific roles to relevant permissions e.g. permission “change from enquiry to offer” is assigned to Manager role. Create the Role “Manager”, link it to “change from enquiry to offer”. Note that a role can have a Revoke flag set. Any permissions assigned to this role will be removed from a users permissions even if they have been assigned in other roles.

As you create users you assign them with Roles that give them access to the relevant windows and functions within the application.

Note – there are three related columns in each table: per_su, rol_su and usr_su. This is to denote a “super user” permission (per_su) that can only be assigned to a role with rol_su that in turn can only be assigned to a user with usr_su. This allows you to assign super user privileges but they cannot be assigned to regular users. This super user would only login occasionally to create a new entgrouporganisation record and assign the users with access to it.

An active role can be made inactive if you want to temporarily disable a certain function within the application. Simply return the role to active to resume normal functionality.

N.B. roles and permissions applied to a user will span all companies the user has access to.

LOGS

There are a number of logs within “infra”.
syslogerrors – when an application error occurs it is logged here

syslogevents - when a routine operation is run (typically autonomously) then a record of that operation running is recorded here.
ualogaccess - whenever a user logs in it is recorded in this table. A user generated logout will also be logged.
systaskstats - whenever a remote task is invoked (typically for web service calls) it is logged here. The scope of this log includes remote tasks involving remote forms. If the remote form involves a login then two log entries will be made - ualogaccess and systaskstats.

SEMAPHORES

This is not a low level semaphore facility. If you have a scenario where a user is to alter a logical record (e.g. a mortgage application) that spans multiple tables (e.g. insurance, documentation received, status changes, log of correspondence) then a soft semaphore can be placed in this table.

Before editing the mortgage application the pseudo code looks like this:

Is there a semaphore on this mortgage application record?
If so, report to user they cannot modify and inform who has the lock.
If not, insert a record in the semaphores table.
Make data available to user for modification.

On OK or cancel:
Delete the record in the semaphores table

On login and ordered logout
Delete all records associated with trCurrUser in the semaphores table.

ASYNC_EMAILS

If you want your application error handler to send an email to the IT department to alert them of an issue, you probably don't want execution to be slowed with sending an email. You can, instead, place the message as a record in this table which is picked up by an external process to send the email. N.B. this is particularly relevant if the customer is remote and you are the "IT department" supporting your application. If this is an in-house application then a direct email to your in-house SMTP server will be fast and reliable enough to omit this step.

AUTOSAVE

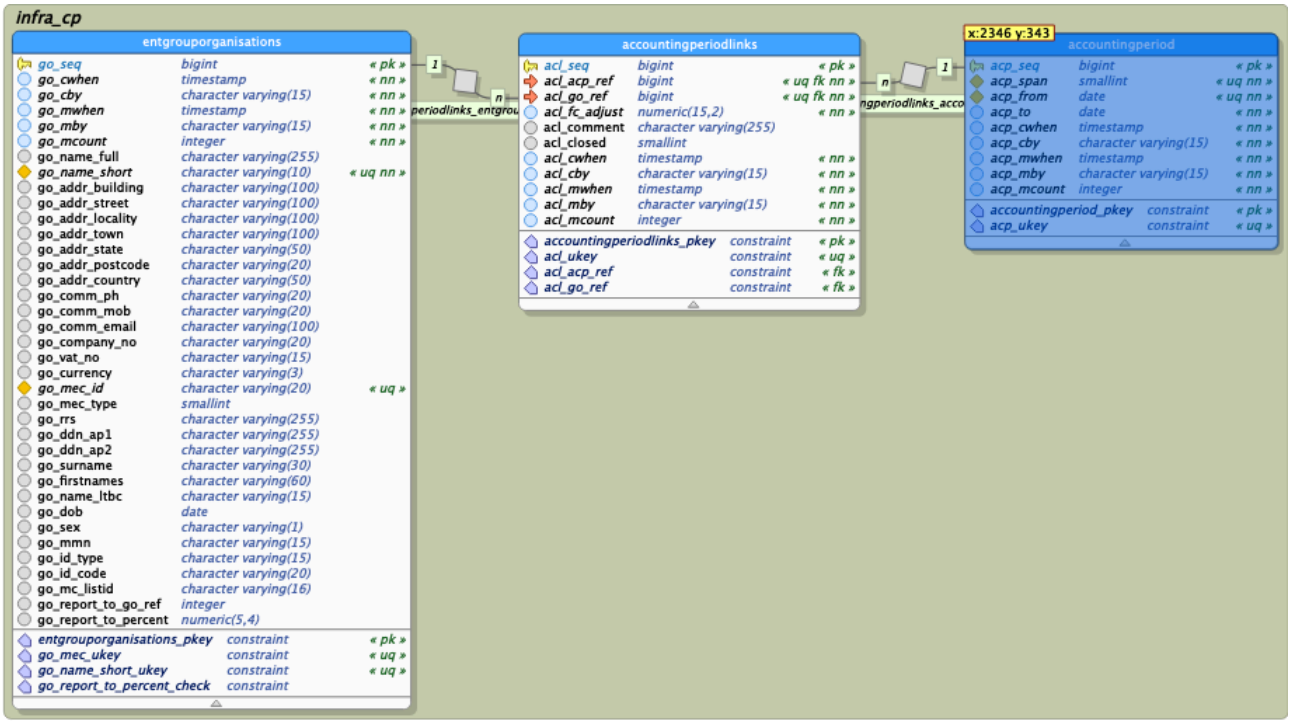
We have a situation where a user can edit a large list of records, including children records of the records in a list. These records have a crude "versioning" aspect to them so Saving one's results is done on an infrequent basis. However in the situation of a failure we did not want users to lose several minutes of work because these records are very intricate. So we implemented a means by which the contents of data in memory are saved to this table after every tab. When the user Saves the contents properly the record in this table is deleted. If the app fails the user in any way, when they return to this edit facility and refer to the same record, we offer them the means to restore the latest copy from memory.

FEEDBACK

If you offer a facility for users of your application to provide comments that get back to the developer, those messages are stored here. You may or may not send an email as an alert as well - that is up to you.

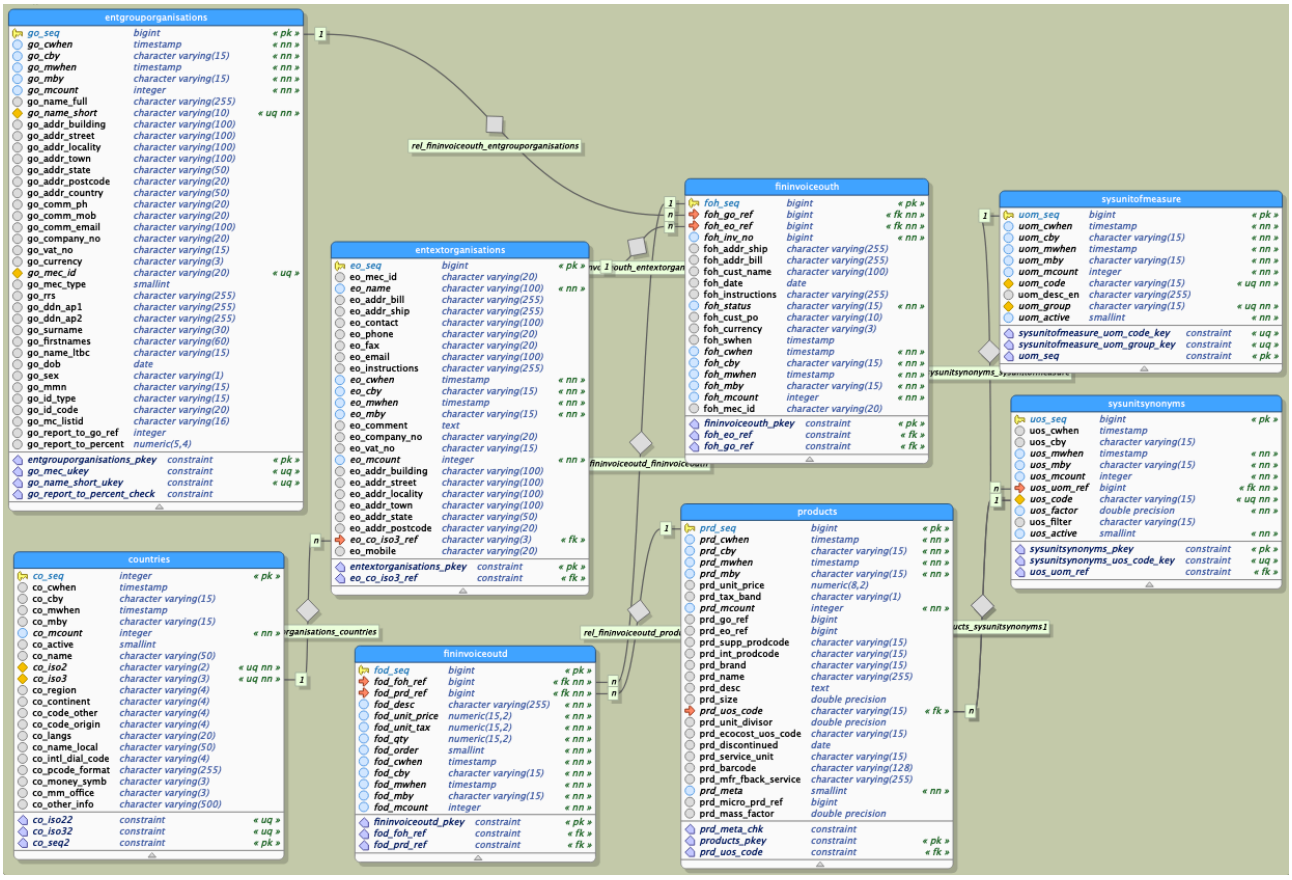
ACCOUNTING PERIODS

This was included in "infra" even though it was specifically implemented for our ecological accounting application because accounting periods apply to a number of different business application contexts. With it you can associate a business operation with a specific accounting period e.g. sending an invoice, which is part of "infra".



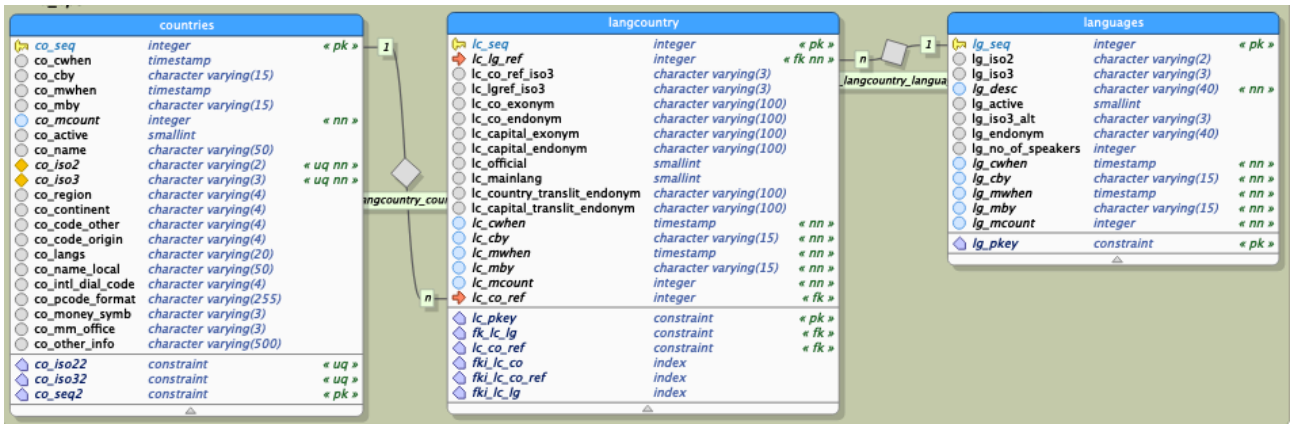
FINANCIAL INVOICES

The fundamental mechanism for recording an income event. Your context may involve a point of sale record which is similar. You will below that this structure includes referring to a unit of measure (from the Products table).



LANGUAGES AND COUNTRIES

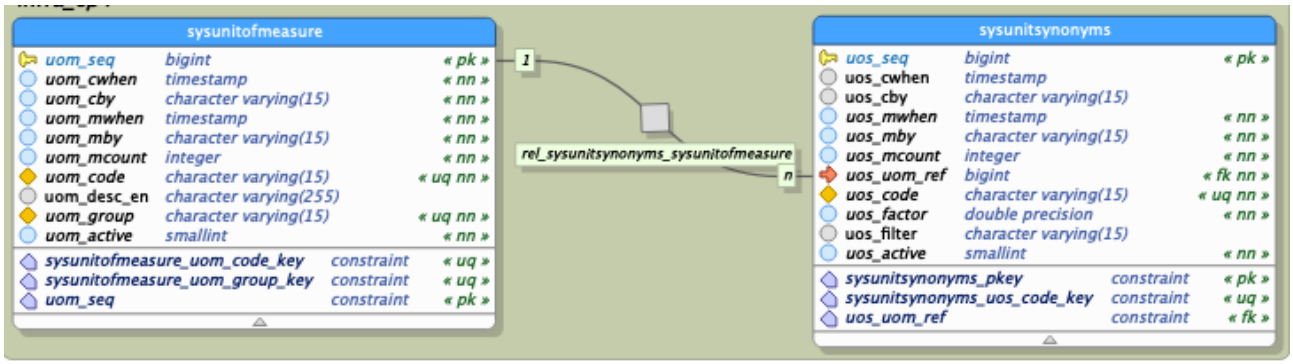
For international websites being language and country sensitive is relevant. The key feature of this structure is that you show the language and country in the language the international user can understand. e.g. there is no use showing “Japan” if the user can only read Kanji. Country name and the capital city of that country are available in endonym (local) and exonym (foreign – for us) format.



UNITS OF MEASURE

This is of particular importance to ecological accounting but its applicability could be generic. You may want to record a product with a unit of measure “pint” but when it comes to making comparisons to other companies whose products are in litres, you need to be able to convert to a common unit of measurement. This structure allows you to do this.

Always offer users your synonyms to choose from which includes a conversion factor to the scientific unit of measure.

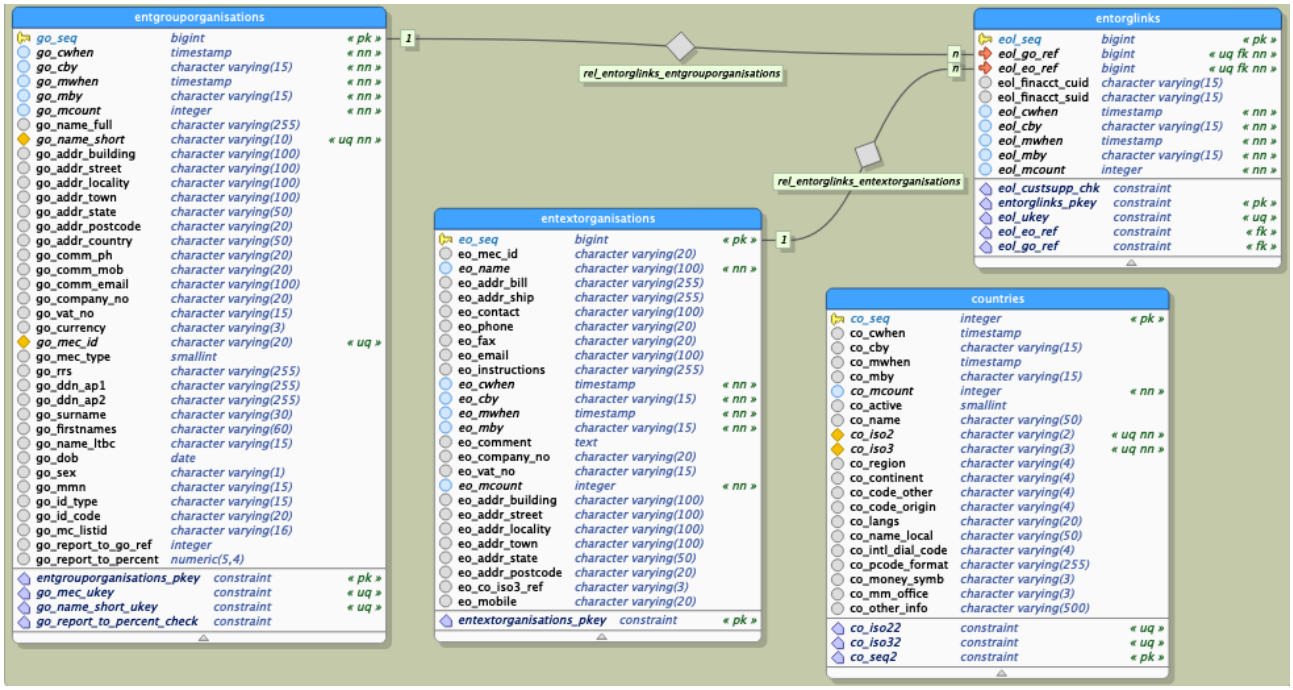


CUSTOMERS AND SUPPLIERS

Customers and suppliers are stored in the same table – entextorganisation – a table of external organisations (as opposed to group organisations). One motivation for this structure is that it is possible to have one company be both a supplier and a customer and we can have the same record for each. There is a link table between entgrouporganisation and entextorganisation, entorglinks, that determines whether this external organisation is a customer or supplier, with a

value in eol_finacct_cuid (customer id from financial accounting) or eol_finacct_suid (supplier id from financial accounting).

This peculiar arrangement has come about because we expect our applications never to be an accounting package (too many already exist) but we DO EXPECT to co-exist with an already established financial accounting system.



CONFIGURATION HIERARCHY

This is part of a visionary configuration facility where some functions will be extended by creating a new option (within a drop down list, say). There are four tables involved and only some of these tables are operational.

The vision is that our application will be used by millions of businesses around the world and we will be releasing new features from time to time. Some of those features will be released in code updates. Some features will be enabled by new entries in specific database tables and our plan for this is with the sysreferenceglobal table.

Global
sysreferenceglobal - using column prefix RFG_ - is a table that is currently accessible but in the future will not be. It is intended as a read only table at a client site. The idea is that this table will be updated from a central database. How this propagation will take has not been determined yet. It could be a poll from the client, it could be postgres replication. Suggestions for how to do this are welcome.

Local
If you have a group of companies at a single site, most of the config options will be handled in the sysreforganisation table - for company specific options. But you may have options that apply to all companies at a specific site. This is the local site - sysreferencelocal - using column prefix RFL_

Organisation
Configuration options applicable to a specific company within the group of companies are stored in sysreferenceorg - using RFO_ as the column prefix.

User
Not in active use. You may want to offer users within the company to set their own preferences. The sysreferenceuser table is there for this purpose, using column prefix RFU_

Configuration Usage

For brevity all examples below will refer to the sysreferenceorg table columns (RFO_) but they apply to all tables with the relevant prefix (RFG, RFL, RFO, RFU).

The columns of each of these tables are almost identical. Identification of records is done by referring to the RFO_CLASS column, that can be further reduced to a single record with RFO_VALUE. To define the 5 options within a dropdown list, create 5 records with RFO_CLASS set to 'OPTIONS'. There is a unique constraints on RFO_CLASS and RFO_VALUE so each of the 5 records must also have a value in RFO_VALUE that together make them unique.

There is a RFO_DESC column which is the one you would typically display in a user interface. Any code that refers to specific options should refer to the RFO_VALUE. This way you have scope to add string table entires to convert the contents of RFO_DESC to a local language.

Each table has a number of columns to store specific values relevant to your configuration:

RFO_CHAR	a character column of 1500
RFO_INT	an integer
RFO_NUMBER	a real number, floating dp
RFO_DATE	a date
RFO_TIME	a short time
RFO_BIN	a binary column
RFO_JSON	to store a number of sub values within a json string.

Two additional columns exist to facilitate time sensitive records:

RFO_EFFECTIVE	a from date that this records relates to

RFO_EXPIRES a to date after which the values within the record are out of date

This table structure has been used for nearly 20 years, before JSON became commonplace. Some code is written with with individual values stored in different records while more recent code would store a number of values within a JSON string.

Access to records in your code is primarily done with the following methods:
Do toValues.$initGlobal('CLASS') Returns kList
Do toValues.$initLocal('CLASS') Returns kList
Do toValues.$initOrg('CLASS') Returns kList
Do toValues.$initUser('CLASS') Returns kList

The RFO_ORDER column is used to display records in a particular order. Each $initxxx method will do a $sort($ref.RFO_ORDER,0,$ref.RFO_VALUE,0)

To bring back a specific config record you use:
Do toValues.$initOrg('CLASS','VALUE') Returns kRow

Configuration "object oriented" record hierarchy - $initInherited()

There is a special method: Do toValues.$initInherited('CLASS') Returns kList

This method was originally written in Omnis code but has been replaced with a postgres function. It was written so that we could issue a new operational function/option facilitated with a new entry in the sysreferenceglobal table but the client may wish to suppress that function or option. Another scenario is that we offer a wide range of options but some companies only want to provide a subset to their users due to the nature of the business (e.g. we offer commercial and personal insurance options but the company only serves personal clients).

To facilitate this the option in sysreferenceglobal would remain active (because it is a read only table) and the company would duplicate the CLASS and VALUE record in sysreferencelocal (or sysreferenceorg) and make it inactive.

The $initInherited method reads records from the org, local and global tables (all referring to the same CLASS) with the "org" entries taking precedence over "local" entries, taking precedence over "global" entries. It is like an inherited class structure, hence $initInherited().

Because this method accumulates records from up to 3 different tables (RFG, RFL and RFO), the prefix used for the returning result set is RFO_.

There are two columns that influence $initInherited behaviour - RFO_ACTIVE and RFO_INHERIT.

Sometimes you want an inactive record to stop going up the inheritance tree and other times you want it to continue up the inheritance tree.

If ACTIVE is zero then INHERIT is used to determine whether searching continues up the tree. If INHERIT is 0 then no further searching is done. If INHERIT is 1 then the function will continue looking for superior records.

To deactivate a value that is active in GLOBAL duplicate the values in a lower table (say sysreferencelocal) and set ACTIVE to 0 and INHERIT to 0.

You may have a situation where you want to provide entries in the reference hierarchy for documentation reasons so you place all entries in sysreferenceorg and you want all these ignored because the real entries to refer to are in sysreferencelocal. You want the $initInherited method to ignore the lower records and look elsewhere further up the tree. In this case set ACTIVE to 0 and INHERIT to 1 for each entry in the lower table (in this example sysreferenceorg).

In all cases if ACTIVE is 1, INHERIT will be ignored and no searching higher up the tree is done.

The sysreferenceglobal table has no RFG_INHERIT column.

IMPORT TEMPLATES

We have a generic data import facility that used the Filemaker UI as an example. To make repeated import operations easier, when an import mapping for a particular table has been submitted, the user is offered to provide a name to save the template. The import mapping is stored in the “importtemplate” table.

Introduction to tMaster and key table class methods

Another feature of the "infra" library is the accrued techniques implemented in accessing the database. The primary class for this is tMaster. All table classes in the application library will be subclassed from infra.tMaster or some other subclass of infra.tMaster.

Originally written to access an Omnis datafile, it has been adapted to suit a postgres db. Nearly all the code is generic SQL. The use of sequences to generate primary keys is postgres specific but adaptable to other databases.

Both single database connection and session pools is supported. If you run with session pools instead of a dedicated trFormSession connection, that is fine. Each table class method that is about to call the database server calls the method:
Do $cinst.$ImmediateSession(loSession,loStatement) Returns lbImmediateSession

If tFormSession is used to determine a database connection at Startup it will be returned through loSession and lbImmediateSession will be kFalse. If tFormSession has no connection, a session from the session pool will be obtained, using loSession and loStatement with lbImmediateSession set to kTrue. If lbImmediateSession is kTrue the session is returned to the pool when it is no longer needed.

The foundation for this structure involves:
- instance variable icPrefix
- instance variable icPrimaryKey
- instance variable icForeignKey
  - multiple foreign keys are supported but only one dominant foreign key
- instance variable ibHasSequence
- a table class for each schema class
- the $construct in each table class
  - sets the above 3 instance vars and
  - executes a Do inherited
- standard record accounting columns
  - XXX_CWHEN creation timestamp
  - XXX_CBY who created record
  - XXX_MWHEN last modified timestamp
  - XXX_MBY who was last to modify record
  - XXX_MCOUNT a count of the number of times record has been modified

Key methods in tMaster

$fetch_pk	fetch row on primary key
$fetch_fk	fetch list on foreign key default foreign key is in icForeignKey, alternate f.keys can be specified
$insert	calls $cinst.$validation to check essential data exists in record calls $cinst.$InsertCols to provide default values (e.g. CWHEN, CBY etc) if primary key has already been provided leave as is else if ibHasSequence the icPrimaryKey is omitted from insert and returned
$update	calls $cinst.$validation to check essential data exists in record calls $cinst.$UpdateCols to provide default values (e.g. MWHEN, MBY) uses MCOUNT or MWHEN as part of the update criteria to ensure record has not been modified - optimistic locking
$delete_pk	deletes record with primary key
$delete_fk	deletes records from a child table if parent key provided

Common methods in individual table classes

$construct	setting icPrefix, icPrimaryKey, ibHasSequence and icForeignKey
$ForeignKeys	builds list of all foreign keys associated with this table
$Links	builds list of joins to other tables in the database
$validation	enables a "trigger level" check of acceptable record contents
$defaults	This may be called after a Do row.$clear() to set defaults values

$Links functionality

By specifying a list of all joins possible from a given table to other tables in the database means coding for SQL becomes much easier. e.g.

```
Do $DefineList(ilInvoices,'tfinInvoiceOutH')
Do ilInvoices.$addLinkedCols('EO_NAME')   ;; add customer name
Do ilInvoices.$fetchDateRange(idFrom,idTo)
```

This will fetch invoice header records belonging to trGroupOrg along with the Customer Name between the date range provided. The programmer does not have to provide the join criteria because it is automatically determined from information defined in the $Links list.

The $Links functionality supports outer joins, multiple routes from one table to another and aliases. It also works using the $fetch_pk and $fetch_fk methods.

A key benefit of using this facility is that the primary table the list or row is defined from, tfinInvoiceOutH in this case, can continue to use $insert, $update and $delete methods to facilitate changes to the record, even though columns from other table exist within the list or row.

International Support

All of the remote forms support internationalisation using string tables. The $construct of each form will Do method TranslateText
which in turn calls Do $cinst.$translateLabels()

This method gathers the name of each visual object in the form and uses the name to get string table entries and substitute values where necessary. The form may have string requirements beyond visual components such as messages, strings to use in superclass objects and he like. These need to be explicitly added to a list within the method.

There is a naming convention for string table entries - name_LBL for labels, name_TT for tooltip information.

We do not store string tables entries in an Omnis string table (.stb) file. We store them in the database using the STBEDITOR library (a standalone library that does not refer to "infra"). The $construct of Startup_Task in the application library should connect to the sub database, fetch all the string table entries from the database into a string table held in memory.

Refer to the STBEDITOR repository for more information.