

This paper explains the fundamental concepts underlying RelationshipGEM and why GEM delivers its remarkable intuitive behaviour. RelationshipGEM is based on Microsoft SQL Server and uses the services of Microsoft Outlook and Word. This means that users do not have to learn a whole bunch of new tricks to be able to use it. It also means GEM is fully scalable from individual home users to the largest enterprise. GEM will run on all versions of Windows, from Windows 2000 on, and works happily with Windows Vista. (Given suitable emulation software like 'VMware Fusion' from VMware.com, it will even run on a MAC.) GEM works with all versions of Microsoft Office, up to and including Office 2007. It is not the underlying technologies that give GEM its unique capabilities – but rather the way that they are used.

Entities and Relationships

GEM does not base its design on things, or Entities, like Customers, Suppliers, Staff, Shareholders and the like, as do conventional systems. This is because someone can be both a Customer and a Supplier, or Staff and a Shareholder. In our view this traditional approach muddles information about who you are dealing with, together with your relationship with them. It also forces you to duplicate their data, if you have more than one relationship with them - and we all know about the problems with duplicating data. It also prevents you from storing anyone else's relationships – and we now know how useful that can be.

So GEM stores information about Entities that actually exist in the real-world, like Organisation, People and the Projects or Jobs that they work on. Relationships are kept entirely separately. This means GEM can also store all kinds of Relationships between everyone and everything in the system as well.

Database Structure

The underlying database structure is implemented in such a way that:

- If something exists only one in the real-world, then its data exists only once in GEM - even phone nos.
- Further there is only one location, anywhere in the database, that stores Phone Numbers or Addresses, for example. (GEM distinguishes between Phone Numbers, Direct-Dial and Mobile Numbers so each is stored separately.)

This makes looking things up a lot easier, as there is only one place to look. In a conventional system, you store a customer's phone number and address in the Customers Table, supplier's phone numbers and addresses in the Suppliers Table and so on. So you need to search every table that you stored phone numbers in if you wanted to find out who a particular phone number belongs to. In GEM it's easy - there is only one place to look.

Bottom-Up V's Top-Down Design

The conventional approach to computer system design is 'Top Down'. You decide firstly what you want the system to do. You then design the data flows that will support these objectives, and then the underlying data tables that will hold the information. This is like designing a skyscraper. You start from the top, designing each floor on the way down, making each floor strong enough to support all the floors above. When you get to the bottom you design the foundations.

In computer terms, the data structure is the foundation of your system. The problem with computer systems however, is always that when they are finished, someone always wants to add to them. This is analogous to adding a couple of stories to the top of your skyscraper. You effectively have to reinforce all the floors below and the foundations as well! So it's often easier to start again. (Anyone remember Incis?)

In computer terms adding new requirements often means you need to re-design your data structure. However all the code that you have written, relies on the data being in a particular place. You therefore usually have to modify a lot of code that is already working, as well as adding the code for the new requirements. Often it is easier to scrap what you've done and start again – just like that skyscraper.

GEM is built effectively from the Bottom-Up. This means that we first design where the data belongs in our model of the 'Real-World'. We then design how we are going to extract that information to make use of it. This gives us a data structure that is almost impervious to change - for even if our requirements change, the 'Real-World' doesn't.

To pull this off effectively we have to be very particular about how we design our data model. It always results in an almost philosophical discussion about where each data element should be placed. It does not depend upon what we want the system to do – rather it depends upon what happens in the real world.

Take addresses for example most computer systems would have an address for a customer, or depending upon what you wanted the system to do, perhaps a delivery address and an invoice address. But when you are thinking about the real world how many addresses can someone have? Well the answer is 'unlimited'. People can obviously have multiple home and holiday addresses. Similarly business can have multiple addresses of all sorts – Business/Office, Delivery, Invoice, Factory... Further the same Address can be the Business Address of multiple organisations and the Home Address of multiple people. So following our design philosophy GEM has to cater for the full complexity of the real-world, and allow anyone to have an unlimited number of addresses that can be shared by anyone in the system. So that is what it does.

It must be realised that much of GEM's apparently intuitive behaviour results directly from this 'Bottom-Up' design philosophy. It did not result from us deciding the behaviours we wanted the system to have, and then designing the data structure to support it. Its behaviour results from the design philosophy, and because the data-structure resembles the real-world, the system behaves like the real-world.

It does mean that our data structure is more complex, but it means that we can always add new requirements to the system without having to re-work anything that we've already done. 😊

Information Components

Because GEM only ever stores one kind of information in one location in the database, it gives us the opportunity to develop a number of standard 'Information Components' that can be turned on for People and Organisations and off for Projects for example. These 'Information Components' cover; Addresses, Departments & Subsidiaries, Staff, Family, Categories, Dairy Notes, Notes, Photos and User Defined Details.

This makes it easier for us to introduce new 'Entity Types' as we only need to concentrate on the differences between what we have already done and the new 'Entity Type'. It also means we can put more sophisticated features into each component, as we know we are never going to have to re-invent the wheel again. Upgrading and maintenance are also simplified, as changing a component once,

changes it everywhere it is used. Life is also simplified for the User as behaviour is standardised throughout the system.

Bi-directional Relationships

In GEM all Relationships are two-way. So the system knows that for example:

- If I am your Supplier you must be my Customer,
- If you are my Nephew I must be your Uncle - or perhaps Aunt.

GEM automatically creates and maintains both sides of a relationship, so that you can always see all the relationships for anyone in the system. Relationship types are all data-driven, so the correct words are used to describe a relationship from both points of view:

Customer \leftrightarrow Supplier

Nephew \leftrightarrow Uncle

and so on.

Because of this new 'Relationship Types' can be added readily – even by the User.