Sentences

from Lazy Software

An extract from the Bloor Research
Product Evaluation Series

Bloor Research

Fast facts

Sentences is an implementation (the first) of the Associative Model of Data. This model is an attempt to resolve a problem that has gone largely unrecognised but is now coming to the fore.

The problem is this: using a relational database, each application is dependent on the tables in the database that it addresses. Why is that an issue?

First, it means that each time you build a new application you have to construct or modify a new set of tables, which is time consuming and complex. Secondly, it means that you cannot easily record unique data. If, for example, you wanted to record information that was relevant only to a particular customer, then you would either have to do that as text, or allow null values in the relevant table column. Neither of which is really satisfactory.

The Associative Model of Data was designed as an alternative to the relational model, that would overcome these, and other, shortcomings of that approach. Sentences is the first database (it comes with supporting tools) to implement the Associative Model.

Key findings

In the opinion of Bloor Research the following represent the key facts of which prospective users should be aware:

- Applications written using Sentences should be much easier to maintain and tailor to customer requirements that applications written against conventional databases.

- The database itself is much easier to design than conventional models. For example, an eleven line Sentences schema (see below) requires 51 lines of SQL. Moreover, the metadata in a Sentences schema is much more intuitive and can be read directly.

- While there will be some additional training requirements involved in adopting Sentences, this will be much less than would otherwise be the case, because of its intuitive and readable nature.

- Currently in its first release, the product has limited scalability and, at present, it is primarily aimed at "Internet-based, departmental information systems" even though the product has been engineered with the features that one would expect in a high-end database designed to support transaction processing.

- The interface for the query part of the product is a bit clunky and not as intuitive as the rest of the product. Lazy Software plans to substantially upgrade this part of the product in the next release.

- Lazy Software recognises that it is not going to replace existing technology at this stage and is therefore focusing on co-existence, notably through supporting ODBC and a Java API. The company is also putting a significant investment into XML for release 2.0 of Sentences, which is due later in 2001.

The bottom line

We like the Associative Model of Data. While Object database vendors used to claim that objects were more intuitive than relational tables, the associative approach is arguably more intuitive than either. Moreover, it is not just the idea of associations that falls into this category, but its implementation in Sentences. This is one area where object databases fell down: the idea may have been simple but the implementation wasn't. What Simon Williams and his colleagues at Lazy Software have managed to produce is both a concept and an implementation that is intuitive and easy to understand.

Of course, being elegant is one thing while resolving business problems is something else again. But Sentences does precisely that. It should result in applications that are an order of magnitude easier to maintain and customise than those based on conventional database technology. Within the constraints of its current levels of scalability we would urge any company in the market for a new database to seriously consider the merits of Sentences.

Vendor Information

Background information

Lazy Software was formed in 1998 by Simon Williams, the former founder and Chief Executive of Synon, together with two of his erstwhile colleagues, Simon Haigh and Melinda Horton.

Despite the fact that the product has only been officially available for less than six months, and that it represents brand new technology, the company has already acquired some dozen customers. The most notable of these is Johnson Matthey, which has used Sentences to build an asset management database, and is currently exploring other uses of the database within its organisation. Lazy Software already has one overseas client, in Australia.

The company describes the typical user system as an Internet-based, departmental information system, usually aimed at improving group productivity. In particular, the rapid development associated with Sentences means that it is particularly suitable for systems that otherwise would not get built, either because of the problems with a relational implementation or because of opportunity costs.

At present the company is recruiting an internal sales force for direct sales and hopes to sign partnership agreements with VARs, to market into the general SME arena. While the flexibility of Sentences could be ideal for the Application Service Provider (ASP) market, the product does not yet scale (see below) to the sort of capacities required by a typical ASP. Thus this market is in abeyance, although Lazy Software hopes to address it in due course.

Lazy Software web address: www.lazysoft.com

Product availability & support commitment

The first general release of the product was in October 2000 with version 1.1. Sentences consists of the database engine itself, which runs under Sun Solaris, Windows NT or Linux, and ancillary tools that provide the user interface, schema specification facilities and query support. These can make use of Microsoft, Apache, WebSphere and Tomcat web servers, with Windows clients running either Microsoft or Netscape browsers.

In the current release the database is suitable for use where the storage requirements are measured in hundreds of megabytes, the number of heavy users in the tens, and hundreds or thousands of casual or web-based users. Pricing is banded based on the number of database requests being handled.

In addition to the product itself, the company offers three classes of service to its users including training, telephone-based technical support and subscription to its library services. This last includes both product updates and web-based technical support.

The next major version of the product will be release 2.0, which is scheduled for Autumn 2001. The major new feature of this release will be XML support.

Financial results

Initially Lazy Software was privately funded by the three founders but, at the end of 2000, the company acquired £3.5m worth of venture capital. At the time of writing the company is solely based in the UK, though it has just signed up its first distributor, in Switzerland. The company plans to open its first office in the United States in May 2001. This, of course, is always a fraught move but Simon Williams and his team have the experience of achieving success in this endeavour with Synon (which eventually established its head office there), so this bodes well for the future. The company currently has 20 staff.

Product description

Introduction

Sentences is an implementation of the Associative Model of Data, of which Simon Williams is the author. So, to understand Sentences we first need to understand the Associative Model.

The Associative Model is based on two things: entities and associations, where entities are things that exist in their own right and associations are things that only exist by virtue of their relationship to something else. This can be illustrated diagrammatically as follows, where both the Entity and Association types are subtypes of Type, and where Association types are both source and target types of Type:

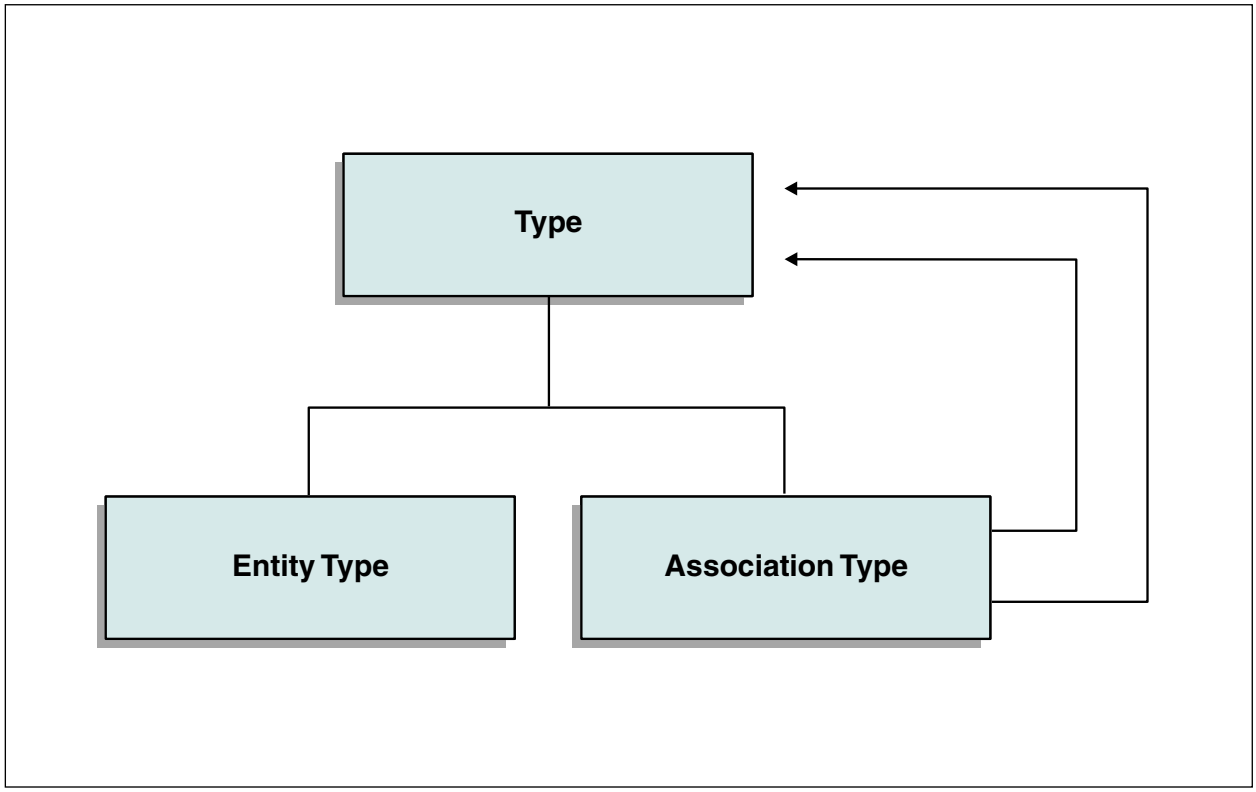

Fig. 1: Entities and Association Types

There are various points to note about these definitions:

1. An association may be used to define the role that an entity plays. For example, a table is an entity. But a dining room table, a kitchen table and a coffee table define particular functions that a table is capable of taking. Similarly, a person may be an employee, a customer, an accountant, a householder and so on. Here, these associations define the roles that a person may play.

2. Associations exist by virtue of their relationship to other things but "things" can be either entities or associations. Thus an association can be dependent on another association. For example, a sales order line depends on the sales order as a whole. Moreover, the sales order may depend on the customer, which is itself an association. This is a marked difference from relationships within a conventional entity-relationship model where relationships only exist between entities. In addition, associations are explicit within this model where relationships are implicit within the relational model.
3. From an implementational point of view, entities may be placed in context. That is, you can circumscribe your world-view for practical purposes. For example, an enterprise is an entity and its departments are associations. However, if looked at purely from a departmental point of view, then the department involved can be defined as an entity.

At first sight, the word "association" may not seem the most intuitive term that Simon Williams could have chosen. However, it should be remembered that association is not simply a synonym for relationship. It is also a noun in the sense of an association of persons, a document setting forth the common purpose of those persons and in the sense of an association of ideas.

The big difference between the Associative and the Relational models, is that the latter separates data from metadata whereas the former doesn't. That is, when building a relational database you have to separately and specifically define the database schema. And whenever you add a new table or change the definition of a table, you have to modify that schema. This is not the case with the Associative model, in which data and metadata are stored side-by-side. For example, the following is how you would define the metadata related to a bookseller, where the items in bold type are entity types:

Legal entity sells **Book**
… worth **Points**
… in **Country**
… from **Date**
… at **Price**

Person lives in **Country**

Person customer of **Legal entity**
… has earned **Points**
… orders **Book**
… on **Date**
… at **Price**

It is worth noting that it would take 51 lines of SQL to write an equivalent definition for a relational schema. Perhaps more significantly, you can actually read the Sentences metadata, to the extent that it is not necessary for us to actually describe the scenario – you can deduce this directly from the metadata.

The effect of this different approach to metadata is significant. In particular, it means that you can write programs that read metadata directly as a part of the program. If an application can do this, then it can run against any schema and you can change the database's metadata without effecting the suitability of the application. By contrast, with an application built against a relational data-

base, you have to change the program whenever the database changes. The Associative model potentially offers huge advantages when it comes to both application reuse and application maintenance.

Architecture

Sentences is called that, precisely because that is how data is described within the database. This is important because it directly mirrors the real world, which is a part of the reason that, conceptually at least, Sentences is so easy to use. In practice, a Sentences database consists of two types of data structure:

- Items: each of which has a unique identifier, name and a type.
- Links: each of which has a unique identifier together with the unique identifiers of three further things, which consist of a source, verb and target. For obvious reasons it is this source-verb-target structure that leads to the product's name. However, it should be noted that each of these may be either an item or another link, while verbs may actually be prepositions.

If we continue with the Bookseller example used above, here is how Sentences would store the relevant data. First, you would define the various entities. For example:

Amazon is a **Legal entity**
Dr No is a **Book**
Simon Williams is a **Person**
Simon Williams lives in *Britain*
Britain is a **Country**

Each of these sentences contains a subject (Amazon, Dr No etc.), a verb (is a, lives in and so on) and a target which is either an item (legal entity, book and so forth) or another link (Britain).

In addition, you will need to store the price list (as well as customer details), which will be in the format:

Amazon sells *Dr No*
… worth *75 points*
… in *Britain*
… from *1-Jan-00*
… at *£10*
… in *America*
… from *1-Mar-00*
… at *$16*

Alternatively, Sentences allows you to define data using brackets to indicate nesting so that the British pricing could be written:

(((Amazon sells Dr No worth 75 points) in Britain) from 1-Jan-00) at £10.

This vertical structuring into sentences has profound consequences. One of these is that there are no rules that predicate that the same number of sentences must apply to each item. In the example above, for instance, the link "… worth 75 points" applies to both Britain and America. However, it would be quite feasible to have defined the schema in such a way that allocating points was not mandatory and we could have links about points in Britain and not in America. Thus, for example, you might want to store more information about some customers and less about others. In the relational model this is only possible by allowing null values within the relevant database tables. Using Sentences you just define the relevant characteristics of each customer, as appropriate. As a simple example, this means that you only need to have the exact number of address lines that each customer requires, rather than having to allow six or seven lines just because a few customers have very long addresses. This is both space saving and, when it comes to more advanced applications such as CRM, more flexible.

There are two other important structures used within a Sentences database. These are profiles and chapters. Typically, a Sentences database consists of a number of Chapters and developers are free to place elements of the database schema in different Chapters as is appropriate. Profiles define the user views into the database and each may consist of one or more Chapters. This means that particular elements of the schema may be operative or inoperative depending on which Chapters are included in a particular Profile.

This organisation has a number of consequences beyond giving a personalised view of the database. For example, a user may change or delete an association within his profile, without affecting other profiles. This is because the deleted association is not physically removed, but acquires a special type of association (a stop link) which simply states that the association has been discontinued. Similarly, an entity name change is accomplished by acquiring a link to the new name.

Further, if you want to extend an application, for example to add a new business rule, then you can simply insert this into the schema and you will not have to change the data in any way. Moreover, through the definition of Profiles you can limit any such extensions to particular circumstances. This means that you can very simply tailor an application for individual users (including support for foreign language versions of the same application). While Sentences probably does not have the capacity to be suitable for Application Service Providers (ASPs) right now, this sort of facility is likely to prove attractive to these sorts of vendors in the future, as well as to VARs and others.

Some other consequences of the Associative Model include the fact that distributive capability is a natural feature of the model and that it will be much easier to merge two associative databases than it is with conventional database models. The ability to cross-report against entities and associations is also important. For example, the ability to list all the associations related to a particular entity effectively provides a where-used list that can be used for impact analysis. Similarly, you can click on an association type and see all instances of that association.

Product components

Sentences includes a number of components. These are:

- The Sentences Server – this is the database engine itself, which has been written in Java. It has been deliberately engineered from the start as a grown up database, with transaction

processing capability, concurrency support, ACID properties, rollback and recovery and so forth. In addition, there is a Java API that may be used for coding directly in Java. Most standard datatypes are supported though there is no support for LOBs. Having said that, since the product is written in Java, it should be relatively easy to build new datatypes when required, or Lazy Software can do this for you.

- Sentences Client – this allows web-based users to access the database. It runs as a set of Java applets that provides search and browse capabilities for both data and schemas, forms data entry capability and query execution. An example of a client screen, illustrating the book-seller problem described above, is shown below.

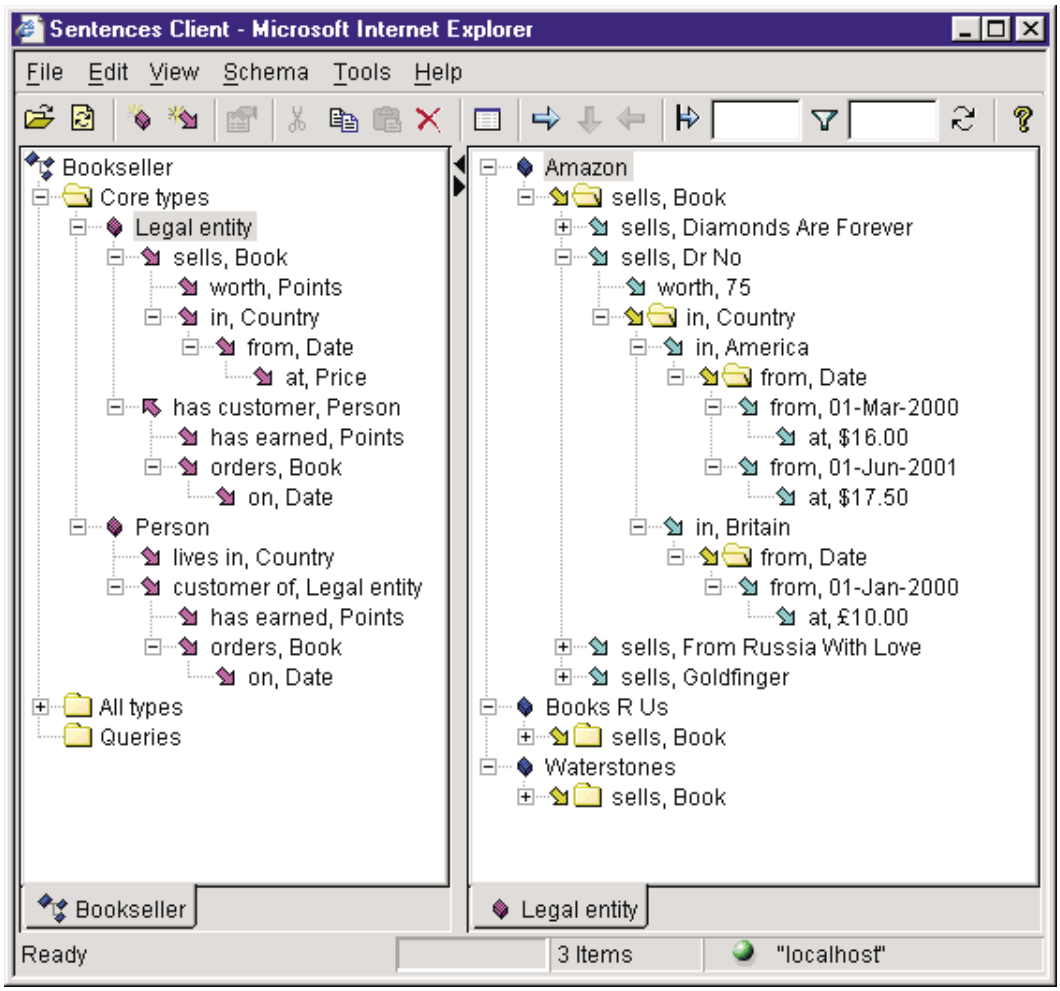

Fig. 2: An example client screen

The query facility uses drag and drop functions and looks superficially similar to SQL, though there are some significant differences which may take some time to learn. This query facility is not as intuitive as it might be and Lazy Software plans to implement some major enhancements in this area in version 2.0.

- Sentences DevTools – this extends the ability of the Client product by providing schema, business rule and query capability in a more advanced manner. This can be used instead of, or in addition to, the Java API for developing applications. As far as the schema facilities are concerned, drag and drop is again used but here the user is much less likely to be confused. Indeed, the lack of any need to reference tables or worry about the restrictions imposed by keys, means that this is incomparably easier than conventional relational modelling. It should also be noted that you can import scheme definitions via CSV (comma separated values) files should you wish to do so.

- Sentences Gateway – this provides interoperability with other environments, primarily through an ODBC interface in this release. A major feature of version 2.0 will be the support for XML. It is worth noting that the combination of Sentences and XML has a distinctly different emphasis from conventional approaches, notable that the view is one from the top down rather than bottom up. For example, using Sentences the approach would be that "a sales order has many order lines" rather than "each order line belongs to a sales order". In our view the top-down approach is more intuitive.

- Sentences QuickStart Application Pack – this includes pre-built applications for Human Resources, Reception Management, Technology Asset training, Library Management, Employee Share Options and Incident Tracking amongst others. And, because it is so easy to modify or adapt Sentences applications, that these can genuinely form the basis for user applications even when these may differ significantly from the basic functionality provided.

Summary

Sentences represents unique technology. Therein lies both its advantage and its disadvantage. On the one hand it offers significant advantages over traditional approaches to data management but, on the other, it will be seen as risky technology by most potential users.

At the present time, Sentences lacks the scalability to support large-scale applications. This is just as well, as it forces Lazy Software to address niche applications within corporates, where those enterprises are more likely to be prepared to try out new technology. This sort of softly-softly approach would seem to be a sensible strategy on the part of Lazy Software.

The bottom line, of course, is whether Lazy Software can maintain its marketing momentum. If it can get this right (and Simon Williams' history suggests that it can) then Sentences can, and should, establish itself as a database technology that can and should be investigated by those building new application databases. Of course, the ultimate accolade will be when another company wants to build a product based on the Associative Model of Data. We don't expect that to be long in coming.

Bloor Research is Europe's leading IT research and publishing organisation. Research is available on many subjects such as Data Warehousing, RAD, 4GLs, Development Tools, Client/Server, CASE, Software Testing, Databases, Object Databases, Parallel Databases, Networking, Computer Hardware, IT and IS Strategy, Desktop Strategy, Rightsizing and Object Orientation.

Bloor Research
Challenge House, Sherwood Drive, Bletchley,
Milton Keynes, MK3 6DP,
United Kingdom
Telephone: +44 (0) 1908 625100
Facsimile: +44 (0) 1908 625124
E-mail: mail@bloor-research.com

Web Site: www.bloor-research.com