

The Golden Age of Online Commerce

Jamie Hogle

<http://www.geniegate.com>

ABSTRACT

The era was 1996-1998, the web was still a relatively new medium. Dial up access was the norm and web hosting packages rarely included database support. The technology boom was about to begin.

This is the story of **LeCart**, an e-commerce solution born in an era of limited resources, destined to see the rise and fall of the economic boom of the 90's.

1. The Project Feature Set

The product was a basic cookie based shopping cart with pretty much all the features one might expect. A ticket/tracking system, checkout system, administrative control panel, PGP encryption, currency conversion, cross-selling, etc..

Coupons

The product did offer one unusual feature (even by today's standards) known as *coupons*, this enabled two business partners to form an association amongst themselves and cross-sell for each other.

LeCart was utilized by several small companies and even a few larger ones, everything from internet pet supplies, bunco games, books, even the web developers from the *Kenny Rogers* website had used it at one point.

2. Working in Tight Spaces

Budget hosting packages of this era offered extremely limited services, sometimes involving a *chroot* environment.

These limitations often meant that you didn't have access to even the standard "core modules" that were supposed to be part of every perl installation.

In extreme cases, a host might only provide the perl binary without *any* of its modules. Needless to say, access to an SQL database was completely out of the question.

2.1. Minimal dependencies

Functioning in a barren environment such as described meant that external modules were highly undesirable. To that end, LeCart opted to handle all essential operations internally. This meant, for example, the now ubiquitous CGI .pm could not be utilized.

2.2. Using a 486 for Speed

By developing the package on an older Intel 486 level computer, LeCart was made to run incredibly fast. Using a slower computer (slow even by the standards of that era) one could quickly spot resource hungry subroutines, making adjustments as needed.

One advantage of a stateless web application is that only a portion of the application is ever needed at any particular time.

Most of LeCart's speed could be attributed to on demand loading of code. While this approach is not as efficient for *mod_perl* or *FastCGI* applications, it did leverage the available technology of the common hosting provider.

3. Everything is a Database

One curious aspect of LeCart was its treatment of everything as a database, despite the fact that it didn't actually use an external database.

3.1. Internal Flat File Database

Inexpensive hosting providers in that era rarely offered access to *mysql* or any other database package. To address this problem, LeCart implemented a custom flat file database system.

Even by today's standards, this is often faster than accessing a remote database. Functionality normally available with SQL would need to be implemented in other ways.

Of particular concern was the product table. Some merchants may have thousands of products, making sequential access impractical. Furthermore, LeCart could not depend on core modules such as Berkeley DBM.

To address these concerns, an optional binary index[†] was implemented and made available for merchants who might stock thousands of products.

Several of the tables were in *paragraph format*, this enabled fields to be added *dynamically*.

```
A flat-file paragraph database  
  
id:100  
title:Sample Product Title Here  
description:D100  
  
id:200  
title:Another sample product  
description:D200
```

In a paragraph database as shown above, one can add fields *at will*, these fields could then be used to store auxiliary information such as template IDs. (we'll address these later)

One advantage of this system was that regular unix *shell tools* could be utilized for maintenance or other bulk loading of product data. Converting from a tab format datafile was extremely easy to do.

3.2. Is it a Template or a Record?

Perhaps the most unique aspect of LeCart was its handling of templates. The intent of the template system was to mirror the web itself on a microcosm level.

This involved designing a "network database" of sorts. The system proved to be quite effective in separating the model from the content as well as keeping much of the *display logic* separate from the actual HTML code.

Communicating the intent was a difficult task, people *wanted to see* a template language where there was none. For example, the template system had no looping constructs and only the most primitive *if/then/else* capability. This was really more of a *data description* language of sorts, enabling a template record to reference other information (such as product prices or titles).

People tended to have strong feelings about this approach, they either really loved it or really hated it.

[†] A binary search index resembles a phone book, it has to be rebuilt each time the information changes. This enables rapid searching at a cost of increased complexity.

A template "record"

```
&INC{I200};  
<h1>Sample Template Record</h1>  
Hello &FORM{NAME};  
&INC{I300};
```

3.2.1. Choosing the Escape Sequence

The selection of the "&" character was not an arbitrary choice, the *template functions* resembled HTML attributes. For example, the `&FORM{NAME};` would expand into the form variable `NAME`.

A common misperception was that this could interfere with standard HTML attributes. This was never the case, the system was able to discern them due to the curly braces.

3.2.2. Confusing Template Names

Perhaps the most controversial decision can be seen in the `&INC{ };` function. As you may have guessed, it simply includes the *template record* `I200`.

Due to a variety of technical reasons, the template records (or just *templates*) had awfully confusing names.

The short names enabled them to be referenced in other tables, just as one might reference keys in other database systems.

This worked out well for displaying extended product information or selecting alternate views for specific products. The product database was paragraph format, one could create an "out of stock" template record and use it for out of stock items by adding the template ID to the product table, the cart would then notice this extra field for the product and present an alternate "add to cart" screen.

To address the problem of confusing template IDs, an alternate index mapped cryptic IDs such as `AC954` to more human friendly symbolic name such as `"cart-checkout-error"`. Due to performance overhead and the fact that no one seemed to know about this feature, these symbolic names were seldom used in practice.

3.2.3. Looping without an Iterator

The template database did not support iterators and one might ask how list oriented data, such as `<select>...</select>` boxes, could possibly be created without a means of iterating over the `<option>` tags.

Like everything else in this system, it was accomplished in a very *database-centric* manner. Each select box had four template records; one for the top, each unselected option, each *selected* option and finally a fourth template for the closing select tag. The software would then use these four templates to build the form element and make it available for display.

As you might imagine, this resulted in a very large number of template records for even the most basic display. While this was not an easy thing to maintain, it did prove itself useful later on in the project when control panels were designed to enable people to tailor the cart appearance on a *widget* basis.

3.3. Internal Functions as Records

The database oriented nature of LeCart was not limited to templates. The actions themselves were rows within another table. Adding a function was a fairly simple matter of adding a record to the appropriate table.†

† of course the function itself had to be *implemented* first

4. No Catalog Generation

With the exception of a rather simplistic product search, LeCart offered no real catalog generator.

While it may seem at first to be a limitation, it was actually one of its biggest assets.

Links to products could literally be placed anywhere a webmaster or merchant desired. Including links sent in email messages.

This "free form" approach enabled it to work with any web page creation tools a merchant may have been using. Webmasters loved this aspect of the package.

5. Keeping it Portable

We could (and did) make the claim LeCart would run on every UNIX platform that supported perl5.

I've seen it run on IRIX, Solaris, *BSD, Linux and countless other UNIX platforms, some of which could not even be identified. During the project's life span, I had *never* encountered a UNIX platform that wouldn't run it.

Perl, and all the hard work spent by countless open source developers combined with minimal outside dependencies were perhaps the leading reasons for LeCart's success in the area of portability.

6. The Sale of LeCart

The economic boom of the 90's resulted in a frenzy of online commerce activity and it was about this time that LeCart was sold to *XML Global Technologies Inc.*

6.1. Former Clients

Perhaps the most distressing aspect of the sale was my relationship with former clients. I was no longer the owner of LeCart and as an employee/founder of *XML Global* the boundaries of what I was and **was not** permitted to do became an issue.

This led to awkward situations at times. Now I was in a position where I could not offer assistance without a conflict of interest. I felt the corporate attitude toward these former clients, many of whom I regarded as personal friends, to be nothing short of appalling. I felt bad about the situation, but there was really nothing I could do.

6.2. The Conversion

The new owners of LeCart had different ideas for the package. Unfortunately these alternate plans were rather like coaxing a moped into becoming a bulldozer. The predictable result was a clumsy bike with heavy steel tracks.

Surprisingly, even under these extreme conditions, LeCart continued to function for several years.

6.3. Remote Hosted Solution

A primary goal the new owners had was the creation of an online "shopping mall". Small businesses would simply *rent* the software as opposed to installing it.

Due to the "everything is a database" model, this turned out to be a fairly simple affair. By changing the template database, we were able to implement a "copy on write" system, effectively shielding clients from each other's changes. Furthermore, due to the implementation of the actual *code* being heavily data driven, we were able to implement customizations on an *individual client* basis.

6.4. A Platform for Technology Demonstrations

The new owners also wanted to use LeCart as a showcase for their other technologies. As you can probably surmise, XML Global was an *XML* based company and while it was possible to convert LeCart to the usage of XML (everything is a database) it was not an optimal marriage.

The procedural, minimal dependency approach that served it so well in the world of low-end hosting had become a serious liability for the heavily object oriented code required in XML demonstrations.

7. Where it Stands Today

When I left XML Global, LeCart was still in use. At last communication, the package had been sold to another e-commerce company.

LeCart.com

Unfortunately, when XML Global went out of business, they didn't contact me regarding the sale of the domain name. Today it's parked and serves advertisements to anyone who happens to land there.

7.1. Meeting or Exceeding Original Design Goals

Although the programming style left much to be desired, I regard the project as a success. It met and exceeded its original design goals.

The product enabled many small businesses to gain a web presence and establish a foothold in the electronic marketplace. It is difficult to say which of these businesses are still around and of those, which of them are still using the application. Unfortunately, the loss of the *lecart.com* domain makes it difficult for any of them to locate me.

7.2. Closing Thoughts

While I can't say I completely regret selling LeCart, I do miss it. Perhaps one day, you'll see another version of LeCart. There is very little I would change, except perhaps the use modern, object oriented programming practices and a database server such as *postgresql* or *mysql*.

I believe that picking a **simple** design model *and sticking to it* can go a long way toward ensuring your software development project is a success.

Often there is a temptation to change the design structure mid-stream, this should be avoided. Adding additional features that do not "fit" with a software products intended purpose is another trap to avoid.

Software should strive to do what it's supposed to do and *only* what it's supposed to do.