

# **A Description of Xpiori XMS vs. Traditional Database Management Systems**

**By Chris Brandin**

**Release 1.1**

**Xpiori, LLC  
2864 S. Circle Dr.  
Ste. 1200  
Colorado Springs, CO 80906  
(719) 527-1315  
[www.xpiori.com](http://www.xpiori.com)**

© 2007 by Xpiori, LLC. All rights reserved.

**Version 1.1**

**Copyright Xpiori, LLC All Rights Reserved**

**Xpiori technology is protected by the following patents:**

**US Patent #5,742,611 (21 Apr 98)**

**US Patent #5,942,002 (8 Aug 99)**

**US Patent #6,157,617 (5 Dec 00)**

**US Patent #6,167,400 (26 Dec 00)**

**US Patent #6,324,636 (27 Nov 01)**

**US Patent #6,493,813 (10 Dec 02)**

**US Patent #6,792,428 (14 Sept 04)**

**Other U.S. and international patents pending.**

**The information in this white paper has been provided by Xpiori, LLC. To the best knowledge of Xpiori, it contains information concerning the current state of information processing technology. Xpiori, LLC disclaims any and all liabilities for and makes no warranties, expressed or implied, with respect to products described in this paper, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. No specific reliance should be made on the material provided herein without thorough investigation of the technology and its proposed application to specific circumstances. Product and technology information is subject to change without notice.**

## A Description of Xpiori® XMS versus Traditional Database Systems

### Introduction

We are often asked to characterize the performance of Xpiori® XMS versus conventional database systems, such as Oracle® or SQL Server. Xpiori® XMS is a very different product targeted at different markets, so direct comparisons are difficult, if not impossible, to provide. In this white paper, we endeavor to provide performance information in a context that those familiar with traditional database technology will be able to relate to. Along with those comparisons that can be made, we provide high level technical discussions about the techniques employed to accomplish common data management tasks. In addition, we provide information about XMS's unique capabilities that are not a part of what conventional databases are designed to do.

### Common Database Functions and Capabilities

Before too much is said about common database capabilities, it should be pointed out that Xpiori® XMS was not designed to be a "better mousetrap"; that is, an incrementally better relational or object database solution. Those capabilities are very well served by existing solutions, and we have little interest in attempting any encroachment into those legacy market spaces. Nevertheless, there are certain expectations for performance and functionality common to most information persistence and management systems. How well does Xpiori® XMS perform in ordinary data processing environments, ones where traditional database systems currently dominate the market?

- **Direct query performance.** XMS generally outperforms traditional databases when servicing direct queries and it scales very well – in fact scaling performance is flatter than typical.
- **Set Intersection performance (joins, etc.).** XMS's set intersection performance is much better. Later in this paper is a discussion that explains why. Also, there is generally less need for joins with Xpiori® XMS.
- **Indexed sequential query performance.** This involves queries that return multiple records in some sort of order, like alphabetical lists, for example. Xpiori® XMS outperforms traditional databases servicing these kinds of queries. XMS contains an optional feature to enable ordered indices. Performance for indexed sequential queries significantly surpasses the capabilities of traditional databases.
- **Storage performance.** Considering that Xpiori® XMS always indexes everything, storage performance is extraordinary. Even without considering that, performance is competitive.
- **Transactional performance.** Xpiori® XMS is fully ACID compliant, providing the same multi user transactional integrity as traditional databases. XMS also supports "optimistic locking".
- **Data center readiness.** Xpiori® XMS supports hot-backup and auto-growth, allowing for 24x7 datacenter operation. In the event of catastrophic system failure, the recovery time is very short.

- **Footprint.** Xpiori® XMS generally requires considerably less space to manage information than traditional databases – in many cases, *much* less.
- **Access control.** Xpiori® XMS has a much more flexible and powerful access control mechanism than traditional database systems.
- **Flexibility.** No comparison - Xpiori® XMS wins hands down. At this point, one would naturally ask why one would want to migrate to Xpiori® XMS, even if performance is often better. If you are looking to simply replace a database in an existing system, you might not want to bother. If you are building a new system the answer is simple – you get all this, and much more, without having to do any work. Xpiori® XMS is entirely self-constructing, requiring no separate database design effort or indexing instruction whatsoever – the savings in time and cost can be tremendous.

How can we do this? Why hasn't anybody else done this?

### How Xpiori® XMS Works

Think back to the days before the Internet and World Wide Web. Computers had to be programmed to *display* specific information, just as they have to be programmed to *manage* information today. Then came HTML, a markup language that embedded metadata about how to display information in the data-streams themselves. HTML is what made the Web browser possible. Now we take the fact that we can display potentially all the information in the world with a single desktop application – without programming - for granted. This was inconceivable before HTML and Web browsers, but without Web browsers, the Internet wouldn't be very useful. Web browsers were not offshoots of prior technology; they were built from the ground up specifically to deal with HTML. XML, like HTML, embeds metadata, only it doesn't serve a specific purpose; rather the metadata in XML characterizes data, providing context. Well designed XML contains all the information necessary to know how to manage the data contained therein, and XMS uses this information to automatically and transparently manage and make available for query all that information – just like a Web browser can display *any* information without having to “know” anything about it and without having to be programmed.

The technology that makes it possible for Xpiori® XMS to do this is called “Digital Pattern Processing” (DPP). DPP was invented by, is patented by, and is proprietary to Xpiori. This is where XMS diverges completely from traditional database technology, and why XMS can do things others can not. Traditional databases basically work by establishing their own structure (tables, rows, and columns, for example), and then indexing that structure. Information is split into data and context; the context is statically predefined, and data can be managed dynamically. It's the static predefinition of context that requires all the work and eliminates the flexibility. Any approach that requires the pre-building of a structure in order to work is ill suited to manage the increasingly dynamic forms of information we are confronted with today. Simply put, old data management technologies won't suffice any more.

DPP solves this fundamental problem by taking a “patternistic” approach to managing information, rather than a structural one. There is no separate architectural data structure for the database, so there is simply nothing new that has to be built. XMS manages all aspects of information – data, context and structure – in the same dynamic way. This is why XMS can be self-constructing. This also means that context, for example, can be just as variable as data - opening up new possibilities for unified information management systems where disparate forms of information can be managed together.

In this realm, no comparisons with traditional databases can be made because traditional database systems simply cannot do any of this. They weren't designed to; objectives were different twenty years ago.

Who needs this new functionality? Why?

### **New Capabilities for Today's Needs**

Aside from the ever-present need to do things "better-cheaper-faster", there are new problems that simply can't be solved with old technology, no matter how much better, cheaper, or faster you make it.

**We need to be able to use information in ways we can't predict.** Xpiori® XMS does not require that you know how you will use information in advance because indexing is automatic and complete. There are no "primary keys"; everything is equally accessible. A perfect example of where this is very important is Homeland Security. An important component of our response to any new crises will be the information management infrastructure necessary to manage it. Using traditional approaches will likely result in development cycles that exceed the life span of the crisis itself – rendering the effort irrelevant. We cannot assume that we can anticipate what information we will need and how we will need to manage it – 9/11 taught us that. With XMS, fully functional new information management systems can be built not in months, weeks, or even days, but in hours.

**We need to consolidate information.** It is simply not practical to have to go to a database (or several), a document management system, a content management system, and a host of desktop applications to get the information we need – or to even know it exists. Xpiori® XMS can manage all information types in a unified environment and offers a large variety of standard interfaces to get information in and out of it.

**We need to explore discover, and analyze information our way.** Xpiori® XMS and its companion product – Insight – allow end users to do full multi-dimensional drilldown/ drill-around exploration operations, automatic correlations and analysis in real time with no programming whatsoever.

**We need to be able to get to and use our information without having to go through "gatekeepers".** Access control (which XMS handles with truly unique capabilities) is one thing; having to expend more time, money and effort to get to information than it is worth is another. Xpiori® XMS is self-constructing and requires no predefinition of use cases, which means the user doesn't have to wait for somebody to rebuild a database for them, or build a new index, or go find what they are looking for, or figure out how to connect office applications, etc... etc...

**We need to be able to take information management for granted – like we do the Internet.** Xpiori® XMS works like the Internet – transparently. We firmly believe that, in the near future, the idea that a database ever has to be "built" to accommodate a new application will seem as ridiculous as the idea that one would have to reprogram their Web browser to display a new Web page seems today.

### **Conclusion**

Although Xpiori® XMS compares favorably with conventional databases in conventional environments, and certainly represents a compellingly superior return on investment, the real

value Xpiori® XMS brings to the marketplace rests in its ability to do things conventional solutions cannot – things that are becoming increasingly important to users.

## Technical Appendix – (high level)

### How Xpiori® XMS Indexes Information

XMS relies on the semantic nature of XML in order to determine how to index it. XML is a hierarchical representation of information. Although much more intuitive than typical database models, it is still not entirely natural. The most direct model for expressing information is “information couplets”; that is data in a complete context. If I tell you my name, for example, I am likely to say something like, “My name is Chris Brandin”. “My name is” represents a complete context, and “Chris Brandin” is the data. A typical representation for this in XML might be:

```
<Name>
  <Last> Brandin </Last>
  <First> Chris </First>
</Name>
```

XMS breaks this down into two “information couplets” for indexing:

- <Name><Last> Brandin
- <Name> <First> Chris

The following index entries are created:

- <Name>
- <Name> <Last>
- <Name> <First>
- <Name> <Last> Brandin
- <Name> <First> Chris
- Brandin
- Chris

The index entries are converted into “Icons” – fixed field, property-preserving symbols that represent the text of the index entries. Icons can be processed much faster, and they occupy less space than text. In fact, they are always the same size, irrespective of the size of the original text strings. Icons make it possible to create so many index entries (in fact, *everything* is indexed – which is why nothing has to be chosen as a key) and still maintain a smaller footprint than traditional databases.

The index entries are placed in a patented virtual associative memory (an aspect of DPP technology). It is worth noting that DPP virtual associative memories are smaller than hash tables, much faster, and can be filled to capacity with no loss in performance. Each index entry points to a node in a hierarchal vector that represents the original XML. The technique is similar in effect to pointing to a node in an XML document. There is one major difference, however. Hierarchal vectors (also a patented component of DPP) can be used to accomplish “node specific convergence” of query terms without having to navigate trees; the technique is called vector correlation.

The result of all this is that when direct queries are serviced, they are completed faster than hash table lookups. And that performance is offered for every tag combination, every data element, and every combination of the two.

Often the ability to accommodate dynamic data types is emulated in relational databases with “name/value pairs”. This is not a suitable solution as it requires that full joins (set intersections) are executed to service what should be direct queries – resulting in performance that is hundreds, or even thousands of times slower. Also, with name/value pairs, all the structure of the hierarchy (implying groupings, among other things) is lost.

XMS’s Index entry architecture is oblivious to the structure of the original XML document. This means that query performance is the same whether the document is conveniently arranged, as in the example above, or the items to be indexed are data tagged in a much larger free text document; the ability to query the information is identical in both cases.

### **How Xpiori® XMS Accomplishes Set Intersections**

Set intersections (joins, etc.) are typically the second most common type of database access. For example, if we were to execute a query against a database of names modeled like the XML example in the above section, the database would intersect all “<Name> <First> Chris” entries with all “<Name> <Last> Brandin” entries and return the result.

XMS uses a unique method (also patented) to optimize set intersections. “Brandin” is an uncommon name, so the number of entries for it will be small. “Chris” is a common name, so the number of entries will be greater. Let’s say, for example that there are 20 “Brandin”s and 1,000,000 “Chris”s. XMS will immediately locate the first “<Name> <Last> Brandin” and then attempt to determine if an instance of “<Name> <First> Chris” intersects with it at the node “<Name>”. Traditionally, this has to be achieved by rippling through instances of “Chris” until either the common node instance “<Name>” is reached, or is passed by. XMS works a different way. The list of “<Name> <First> Chris” is structured (as are all index entry duplicate lists) in such a way that it can be navigated using binary reduction. Rather than having to look at the first instance, the next, and so on, the first entry sought will be entry 500,000. If that’s too high, XMS will try entry 250,000. If that’s too low, XMS will go to 375,000 – and so on. The effect is that it can never take more than 21 probes to either find the proper instance of “<Name> <First> Chris” or to discover that there isn’t one. This means that set intersections are the log<sub>2</sub> sum of the size of the sets being intersected, rather than being related to the linear size of the sets (which is typical for traditional databases). The upshot of this is that, in XMS, set intersections are very, very fast.

### **How Xpiori® XMS Minimizes Storage**

XML has the reputation of being a “bloated” means to represent information. True, it is, especially if it is stored as “BLOB”s or “CLOB”s, as is the case with most traditional databases. Either that, or XML is shredded into rows and columns, which defeats the advantages of using XML in the first place.

Xpiori® XMS converts XML documents into vectors of Icons (fixed field representations of strings), preserving the integrity of the original XML. Icons “stand in” for original data for the purpose of processing. One of the functions they serve is to act as pointers to dictionaries containing the string items contained in the original XML documents. The dictionaries contain no redundant strings because multiple Icons can reference the same text string. Therefore all the redundancy inherent to a series of XML “records” is eliminated, and tag and data “verbosity” has a negligible effect on storage footprint. Another reason storage is efficient is because there is no equivalent to “columns” in XMS, so data items only present in some records are only allocated to those records, and not globally (resulting in no empty fields taking up space). The total XMS footprint, including everything – the original XML, indices, vector tables, program items – typically ranges from 1x to 2.5x the original data, and even less with highly homogeneous data sets.

## How Xpiori® XMS Supports Access Control

Typically, database management systems provide access control by limiting access to columns in tables based on roles. Although somewhat inflexible, this works well for conventional, homogeneous applications. This model, however, tends to break down in many applications – especially if disparate forms of information are managed in one system. Xpiori® XMS takes a completely different approach to the problem; this approach is based on patterns that can include field types, data contents, structure, role – nearly anything that can be expressed as a pattern. Furthermore, rules can be arbitrary, not just limiting access to tables or columns.

The basic process behind enabling access control consists of the following steps:

- An administrator establishes a profile. A profile can be based on role, group, or individual.
- Rule triggering patterns are established. They can be based on virtually any available information – even the contents of unrelated fields in unrelated records or documents. For example, a pattern could specify that a rule should be applied to a particular field if a particular user is logged in and the contents of another field, in another record, is more than “x”.
- Rules to be triggered by the above pattern matches are specified. Rules can also be arbitrary, specifying whether the user can see the field, change it, delete it, or even know it exists. Rules can be applied to data fields, nodes (and their children), documents, and functions.

## How Xpiori® XMS Supports Information Exploration, Discovery and Analysis

Xpiori® XMS supports a number of functions that have no equivalent in traditional databases. Primarily, they are focused on ad hoc discovery, exploration and analysis. These tasks are traditionally very difficult and expensive to do, requiring separate data cubes (yet another database design and building task) and the like. Even with the additional work, flexibility is lacking. Xpiori® XMS exploration, discovery and analysis functions were designed with the following objectives:

- Real time operation. The user should be able to get what he needs as fast as he can think of it.
- Iterative operation. The user should be able to work one step at a time, changing his approach according to what is discovered along the way.
- Support of a “personal path of discovery”. Each person has his own way of going about solving problems. If he is forced into following a path defined by somebody else, he will be less efficient.
- No programming required. If anything has to be programmed, the exploration, discovery, and analysis process will not be real time, or personal, or efficient.
- Help along the way. When a user discovers something interesting, but doesn't know where to go next, the system should be capable of suggesting possibly interesting paths to pursue.

A good way to describe special capabilities XMS possesses to accommodate these goals is to follow an example analysis session and describe what goes on “under the hood”. In this example we will explore the National Transportation Safety Board (NTSB) database of automobile complaints:

The first step is to get the database. It is available via the Internet in a variety of formats – we will pick “comma separated values” (CSV). Once downloaded, the file needs to be converted into XML using an ETL (Extraction, Translation and Loading) tool. The process is simple, straightforward, and takes about half a day to complete. While the ETL tool is transforming the NTSB database, the database is being loaded into XMS; so once the transformation is complete, we are ready to go.

The next step is to bring up Insight – Xpiori’s exploration, discovery, and analysis tool. Insight discovers what is in XMS through reflection and presents a tree showing available query items (which, in fact, corresponds to everything in the database). This process takes less than a minute.

Now the analysis session actually begins. First, the user has to decide where to start. He may decide to see a breakdown of records by state, or manufacturer, or tire model, or any other field in the database. He chooses by simply double clicking on a field. XMS then determines how many unique data items there are for that field and presents a histogram of the counts for each item. XMS’s context/data index maintains a count for each duplicate chain, so the item counts for the histogram can be accessed directly, in one probe, without having to actually locate each individual data item and count it. The result is that the query is serviced in several seconds instead of minutes, hours, or days – and that’s true no matter which field is selected.

In this session we have chosen to get a breakdown by state. It appears that a vast majority of records come from four states – Texas, Arizona, Florida, and California. This seems interesting, but we don’t know where to go next. We select those four states as “Filters” and run a “Correlation” function in order to see if Insight comes up with any suggestions. The correlation function automatically goes through every other type of field in the entire database and determines whether there are any other factors following the same pattern as the four selected states.

Two types of correlations are available: absolute and relative. The absolute correlation measures the frequency of every data item in every other database field, filtered by the selected states, and presents how commonly they appear within those states. The relative correlation does all that, but also determines how commonly those factors occur in the states not selected and compares the two. For example, an absolute correlation might show that 82% of records involve a particular automobile brand for complaints in Texas, California, Arizona, and Florida. A relative correlation would show that, plus it might show that this brand of automobile only account for 75% of complaints in other states, and that, in the selected states, these automobiles have 7% more complaints than in the other states. Absolute correlations typically take seconds to complete, and relative correlations take less than a minute. The ability to directly access discreet data item instance counts is critically important to making this work; otherwise, the correlator would have to read the entire contents of the database to service each and every relative correlation request.

Further drilling down and around, plus correlations, might go something like the following:

- After the manufacturer is selected, another correlation is executed and it is discovered that one particular model of automobile accounts for 78% of the records pertaining to that manufacturer. That model is selected.
- Another correlation shows that tire blowouts account for 59% of the records. Blowout is selected.
- Another correlation shows that 35% of blowouts occur on rear driver side tires, which is more than for rear passenger side tires. Rear driver side tire is added to the filter.
- Another correlation shows that an unusually high percentage of records are dated 1996.

Does hot weather relate to blowouts? More so for a particular tire brand? Was there a design change in that particular automobile model in 1996? These are all good questions. And the time it took to go through the entire exploration, discovery, and analysis process was less than half an hour (after the database was loaded), which means we have gone from locating the NTSB database to having discovered all this in about half a day.

Furthermore, it wouldn't have made a difference what order we did the discovery and analysis in. We could just as easily have started with manufacturer, then tire model, then automobile model, then state, etc...

In order to achieve this, Xpiori® XMS makes use of several unique features (that is, features generally non-existent in traditional databases):

- Self-construction
- Automatic indexing of everything (context, data, and the combination of the two)
- Intrinsic "n-dimensionality" (where everything is equally keyed and indexed)
- No need to determine how data is to be used in advance (because everything is already indexed)
- Vector correlation (no tree navigation)
- Icon processing (small symbols instead of big data = much faster processing)
- Direct access to discreet data item counts

Without any one of the above features, this kind of discovery, exploration, and analysis would be incomparably more difficult, time consuming, and expensive to accomplish.

# # #