XML Persistence

John Davies
Co-founder and CTO

Incept5

John.Davies@Incept5.com

Overview

- **A bit of background**
- **The world of the database**
- **Today's complex messages**
- **The Tools of the trade**
  - GigaSpaces, Tangosol (Oracle), Terracotta
- **The Enterprise without a database?**
- **Conclusions**

Incept5

Background

- **John Davies - “Über Geek”**
  - Founded 3 successful companies in the 80s and 90s
  - Chief global architect at 2 large investment banks
  - Co-founder and CTO of “C24” (Iona --> Progress)
  - Revolution Money - Chief Architect
  - Chief Architect at two London-based firms
  - Incept5 - Co-founder and CTO

- **Incept5?**
  - Founded in May 2008
  - $ six figure turnover within 6 months
  - Open-source hierarchical persistence API (to be released through Spring)
  - Building a derivatives matching and reconciliation engine

Incept5

Scalability

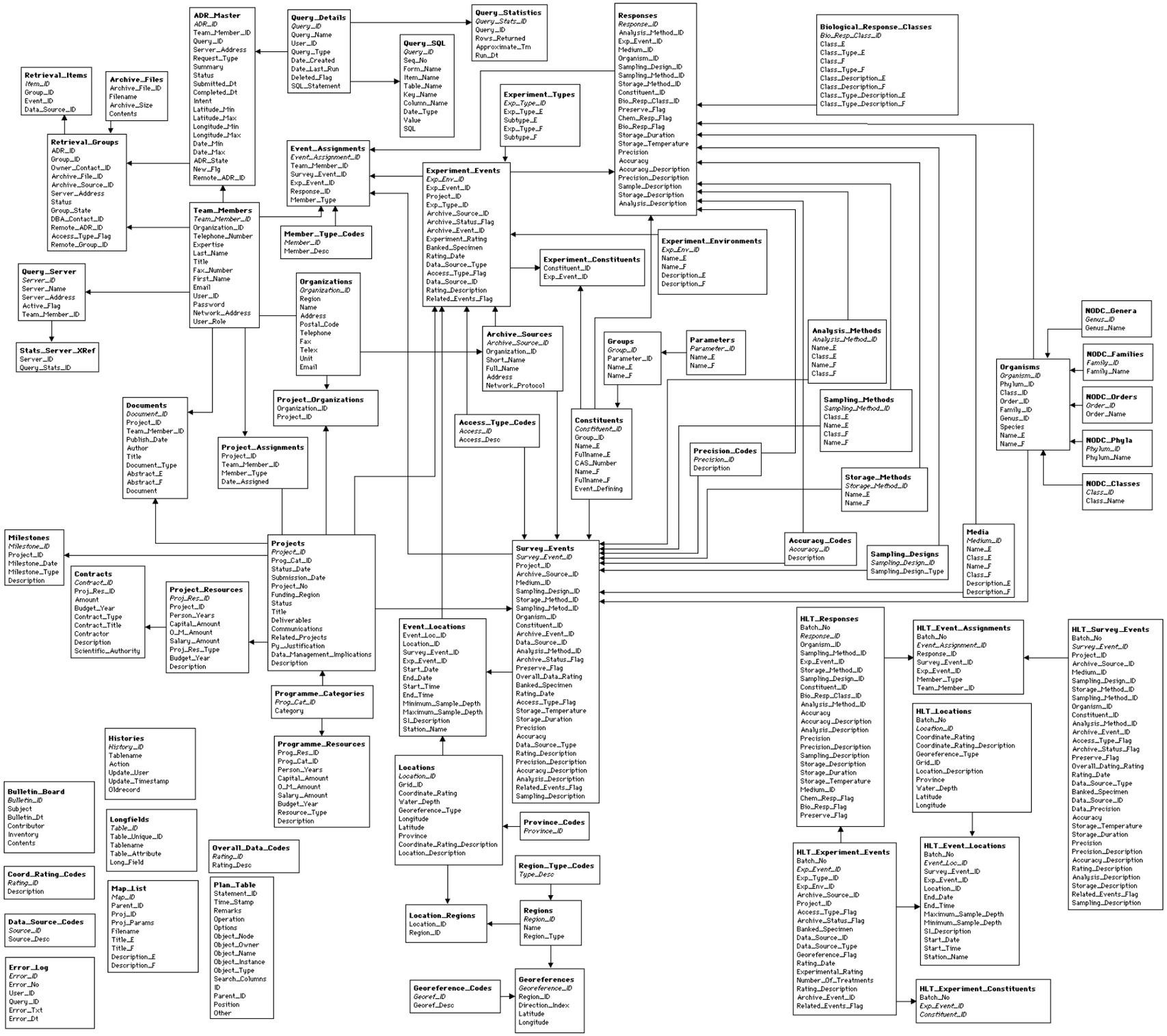
- **A customer...**
- **Previously written in .NET on MS and Oracle**
  - A big Oracle shop
- **Needed to scale from 25,000 to 50 million customers**
  - That's 2,000 times!
  - In 6 months!

Centre of the World

- **Up to 10 years ago everything revolved around the database**
  - Everything went into the database
- **The database was the source of all data**
- **Much of our business logic could be found in stored procedures**
- **Integration was in and out of the database**
  - The bus/network connected other databases
  - The Enterprise Service Bus didn't exist - Integration was "ETL"
- **The DBA was king**

Incept[5]

Such an easy model to work with…



Wisdom Prevailed

- **We started to move the business logic into the application layers**
- **Say hello to Java Enterprise Edition (JEE) and the Enterprise Java Beans (EJB)**
- **We tried to move the data into the application server**
- **Object-Relational-Mapping (ORM) provided the link from relational database to Java objects**

ORM

- **ORM tools became more and more powerful**
  - First TopLink and then Hibernate, iBatis, CocoBase and others
- **Business logic finally started to come out of the database towards the application layer**
- **Increasingly we were tempted to complete the logic in the “O” layer and avoid the “RM”**
- **Relational Mapping is expensive**
  - In theory it’s automatic but it needs a lot of tuning
  - Queries can be extremely complex
  - Models are either database friendly or object friendly but never both

There's nothing wrong with ORM

- **If your Objects (Java in our case) are relatively simple, ORM is a great way to persist them**
- **Similarly if the database schema is relatively simple, ORM is a great way to access the data**
- **To "talk" to any database via Java, ORM tools offer the simplest mechanism in most cases**
- **In fact if we were to abstract away the RDBMS all together an ORM tool would probably be the best starting point**

Incept[5]

Today's standards

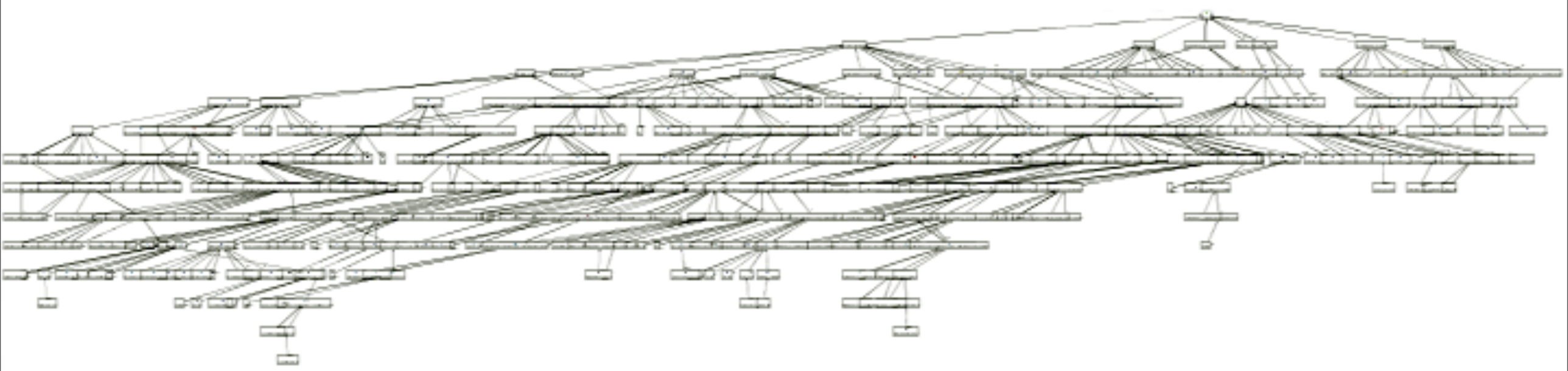
- **Standards are not published as database schema**
  - The used to be
- **ISDA's FpML and the ISO-20022 are good examples of today's standards**
  - Messages and meta-data
- **The latest releases contain thousands of elements with typically a dozen levels of hierarchy**
- **To map these to a traditional relational database can take man-months**
  - But bear in mind, the standards change every few months

Incept[5]

An FpML Swap

- **The fuzzy patch below is the complete model of and FpML Swap from the IRD (Interest Rate Derivative) schema**

- **It's one of several dozen financial models in FpML**



Incept[5]

FpML messages

- **This still needs to be stored...**

```
<?xml version="1.0" encoding="UTF-8"?><!--
  == Copyright (c) 2002-2007. All rights reserved.
  == Financial Products Markup Language is subject to the FpML public license.
  == A copy of this license is available at http://www.fpml.org/license/license.html
  -->
<FpML xmlns="http://www.fpml.org/2007/FpML-4-4" xmlns:fpml="http://www.fpml.org/2007/FpML-4-4" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" version="4-4" xsi:schemaLocation="http://www.fpml.org/2007/FpML-4-4 ../fpml-main-4-4.xsd http://www.w3.org/
2000/09/xmldsig# ../xmldsig-core-schema.xsd" xsi:type="DataDocument">
   <trade>
      <tradeHeader>
         <partyTradeIdentifier>
            <partyReference href="party1"/>
            <tradeId tradeIdScheme="http://www.chase.com/swaps/trade-id">TW9235</tradeId>
         </partyTradeIdentifier>
         <partyTradeIdentifier>
            <partyReference href="party2"/>
            <tradeId tradeIdScheme="http://www.barclays.com/swaps/trade-id">SW2000</tradeId>
         </partyTradeIdentifier>
      <tradeDate>1994-12-12</tradeDate>
   </tradeHeader>
   <swap><!-- Chase pays the floating rate every 6 months, based on 6M USD-LIBOR-BBA,
         on an ACT/360 basis -->
      <swapStream>
         <payerPartyReference href="party1"/>
         <receiverPartyReference href="party2"/>
         <calculationPeriodDates id="floatingCalcPeriodDates">
            <effectiveDate>
               <unadjustedDate>1994-12-14Z</unadjustedDate>
               <dateAdjustments>
                  <businessDayConvention>NONE</businessDayConvention>
               </dateAdjustments>
            </effectiveDate>
            <terminationDate>
               <unadjustedDate>1999-12-14Z</unadjustedDate>
               <dateAdjustments>
                  <businessDayConvention>MODFOLLOWING</businessDayConvention>
                  <businessCenters id="primaryBusinessCenters">
                     <businessCenter>GBLO</businessCenter>
                     <businessCenter>JPTO</businessCenter>
                     <businessCenter>USNY</businessCenter>
                  </businessCenters>
               </dateAdjustments>
            </terminationDate>
```

Incept[5]

FpML in the Database

- **Ever tried using ORM to create a relational model of FpML?**
  - 4000 elements, over a dozen levels of hierarchy
  - Most tools break with this level of complexity
  - Queries can be half a page in length with all the joins
  - Performance sucks

- **Just when you've get the basics working the new version of FpML comes out**
  - Releases are roughly every 2 months

- **Maintaining the changes is an increasingly difficult task**

FpML is just an example

- **No one uses “raw” FpML, we all specialise**
  - Murex, Swapswire, DTCC, internal canonical formats, each is slightly different
- **FpML via ORM is not a practical option**
  - Storing FpML as a BLOB is a more viable option
- **To persist FpML or similar complex messages another solution is needed**
- **Can we use the native XML support in modern databases?**

Incept[5]

XML Databases

- **Using the XML features of the database may appear to be a good choice**
  - Oracle, Sybase, Berkley DB, DB2 etc. all offer native XML persistence
- **There is no standard API or set or features across multiple vendors**
  - As a result you get vendor lock-in
- **Vendor tooling for the complexity of FpML is limited**
- **Performance is never optimised and is often several times slower than “home-brew” methods**

Oracle XML in 11g

- **Oracle's 11g seems to have added several new features for XML support**
  - Binary support is one feature offering impressive performance gains
  - Better XQuery support

- **BUT - It's Oracle's API, you can access most of the features through Java but you'll never be able to port to another database**

- **If you're going to get stuck with one database Oracle's a good choice but it will cost a lot of $$$ (€€€/£££ etc.)**

- **Perhaps Berkley DB - XML will provide a viable OS alternative?**

Oracle is proprietary

- **Oracle works but it's a lock-in**
- **No one likes to be locked in to a single solution**
- **There is no JDBC extension or API for handling XML**
  - Or Hibernate, Toplink or JPA
- **Using JPA means we can persist virtually any class in any database**
  - JMS means we can send virtually any message on almost any messaging system
- **But there's nothing around for XML Persistence**

Incept5

XML Persistence API

- **What is missing is a generic API for XML Persistence**

- **Imagine this...**

```
XMLStore xStore = new XMLStore();
xStore.write("<doc><header id="123">My Doc</header><body>Loads of
stuff<body></doc>");
```

- **Some time later...**

```
String myDoc = xStore.find( "/doc/header/@id='123'");
```

- **But it works on any database or better still - works without a database**

The Database Cache

- **The trend towards the data grid started with database caching**

- **Database caching provides a serious performance boost with complex models**

- **Several of today's data grid vendors started life as caching vendors**
  - These are typically the best options for moving the database into memory

When is a cache not a cache?

- **Cache comes from the French “to hide”**
  - But who cares what the French think?
  - This usually refers to hiding a database
- **If there’s no database then it’s not really a cache**
  - A distributed data grid without a database behind it is not a cache
- **Data is often short-lived, there is little point in writing it to a “classic” database**
  - Resilience is achieved through replication
  - This scenario can be termed a “distributed in-memory database”
  - A “classic” database (e.g. Oracle) can be used for archive

So, data grid or compute grid?

- **There's a thin line between a data grid (cache) and a compute grid**
  - What if we need lots of crunching on lots of data? – Not unusual
- **This is the problem the vendors have in positioning their products**
  - Some come from a compute background but can also function well as a data grid or cache (e.g. GigaSpaces)
  - Some come from the in-memory database or caching background and can also provide an excellent platform for number crunching (e.g. Gemstone and Tangosol)
- **Bear in mind where these technologies come from when considering which fits your needs**

Java Grid vendors

- **GigaSpaces**
  - Started as an implementation of Sun's JavaSpaces (part of Jini) in 2000
- **Tangosol (now Oracle)**
  - Started in "classic" caching in 2000
  - Acquired by Oracle in March 2007
- **Terracotta**
  - Founded in 2003, the youngest in this group
- **Others**
  - IBM, GemStrone, GridGain etc.

Tangosol/Oracle

- **Product is "Coherence"**
  - Started as a cache and has remained in this space
  - Early success perhaps due inefficiencies of EJBs
- **Extremely easy to use**
  - Essentially distributed Hashmaps
  - The API is already known to most Java programmers
- **Includes event mechanism for active queries**
- **Partnered and well integrated into many JEE vendors**
  - Hooks well into Spring, Hibernate and KODO etc.
- **Now playing strongly into the grid market place**
  - But acquisition by Oracle has made them less agile, more expensive and increases Oracle lock-in

GigaSpaces

- **GigaSpaces were one of the first implementations of Jini's JavaSpaces**
  - Jini was originally sold around the mobile phone networks, i.e. massively distributed
- **The JavaSpaces API is incredibly simple**
  - The basic API has just 4 methods
- **GigaSpaces coined the phrase "Space-Based Architecture" (SBA)**
  - The open source version is called OpenSpaces and is Spring-based
- **JavaSpaces is by default event driven and distributed**

Terracotta

- **Described as "Network Attached Memory"**
  - Quite simply Distributed Shared Objects (DSOs)
- **Unique in that it is open source**
- **A much lower-level than the other technologies**
  - It leaves most of the features to the use
  - However the advantage is in its flexibility and therefore power
- **Being open source it is already integrated into several other products**
- **Terracotta is probably the easiest technology to use in conjunction with others**

Can these replace a database?

- **GigaSpaces**
  - Probably the least obvious fit as an in-memory database however objects can be written and queried in a similar way to the other technologies
- **Tangosol**
  - Very similar to GemStone, Coherence presents a Map interface with advanced facilities for indexing
- **Terracotta**
  - Terracotta on its own is not a good in-memory database replacement however it's low-level API mean that distributed maps can be implemented to provide much of the functionality provided above
  - Their main niche is that the solution, although less "out of the box" is ultimately more flexible and cheaper

Working without a database

- **Rather than draw the classic database symbol on your architecture diagrams draw yourself an in-memory data grid**

- **The API is CRUD/Query**
  - The same as a relational database

- **Write the object (e.g. bound Java code for FpML) directly to the in-memory database**

- **Use getters and/or XPath to search/query objects**

- **Only write data you want to store long-term to a relational database - usually as a CLOB**

Ideally we'd abstract the implementation

- **This is the sort of interface we might work with...**

```
package com.incept5.xmlstore;

public interface XMLStore {
    Object add(Object name, String xml);

    boolean createIndex(String name, String xPath);

    String[] search(Object value, String indexName);

    String[] search(String xPath);

    String searchFirst(String xPath, String index);

    Object remove( Object guid );
}
```

Everyone's doing it

- **The in-memory database is replacing the classic relational database**
- **The relational database can be used to store blobs**
  - If only one index is required then the relational database is little more use than a file system
- **Sharing distributed memory can result in up to (and even over) 1 TB in memory**
- **Technology such as Solaris's ZFS provides an interesting mechanism to store "blobs"**

Incept[5]

Conclusion

- **The classic relational database no long meets our needs for complex data**
  - Even after squeezing the data in they are too slow to meet today's volumes
- **Memory is now so cheap it can be used to store almost all our daily needs**
  - Memory is much faster than disk-based searches
- **There's nothing wrong with storing completely de-normalised data when the data is structured**
- **XPath and XQuery offer a viable and in many cases better choice for searching than SQL**

Remember this!

- **If you walk away from one thing in this talk it should be this...**
- **Storing data is commodity these days, don't assume it has to be an RDBMS**
- **Design your persistence layer to persist your primary artefacts with minimal change**
- **XML has come of age, it maps better to Java than an RDBMS, use it to persist your objects**

Thank you

Tak!