# Are you ready for Yottabytes?

## *StorHouse – Federated and Object/Relational Solution*

**Felipe Cariño Jr.** **Art Kaufmann** **Pekka Kostamaa**

**FileTek, Inc**

**360 N. Sepulveda Blvd. Suite 1080**

**El Segundo, California 90245**

**fcarino@filetek.com** **akaufmann@filetek.com** **pkostamaa@filetek.com**

**Abstract**

This paper describes how federated and object/relational database systems can exploit cost-effective ***active storage hierarchies*.** By active storage hierarchy we mean a database system that uses all storage media (i.e. optical, tape, and disk) to store and retrieve data and not just disk. A detailed discussion of the Atomic Data Store data warehouse concept can be found in [CB 99]. These also describe a commercial relational database product, ***StorHouse/Relational Manager (RM),*** that executes SQL queries directly against data stored in a complete storage hierarchy. This paper focuses on applications that can use, and may even require the use of, emerging federated and object/relational database technologies. Our analysis is based on two products now in development. We will refer to these as *StorHouse/Fed* (a federated database system that includes StorHouse/RM) and *StorHouse/ORM* (an Object-Relational database system). We conclude by describing candidate applications (with an emphasis on the federal sector) that can exploit the combination of cost-effective active storage hierarchy with federated and/or object/relational database technology.

### 1.0 Introduction

Commercial Database Management Systems (DBMS) have heretofore been designed to utilize disk storage. There is an active debate as to whether disk technology will evolve to the point where it is cost-effective to store truly large amounts of data (i.e. yottabytes.) It is our position that this will not happen. This topic has been discussed in other academic forums [CBOS 99]. Papers that describe and analyze storage systems capabilities and trends [MLLSKG 99] as well as "active disks" are [HS 96] and [RGF 98]. Storage device measurements are documented in [JM 98] and [CBa 99].

Database systems and applications primarily use disk media as their storage. Some support for optical media may be provided as part of a vendor's Hierarchical Storage Management (HSM) option. With current database implementations, HSM solutions require extensive database administrator support, or custom programs to be written to support optical media.

In the majority of implementations, tape is used to backup and restore whole or portions of database systems. In this sense, tape is a "passive" medium; it is not used for direct data storage and querying.

We use the term Active Storage Hierarchy when (say SQL) queries execute against data stored on diverse media. This should not be confused with the research term "active disks" [RDF 98].

The *current* economics of the storage media are that storing data on tape is approximately 7% the cost of storing the data on disk. The cost for storing data on optical media is about 42% of storing the data on disk. Figure 1 illustrates the cost-benefit tradeoffs for different storage media.

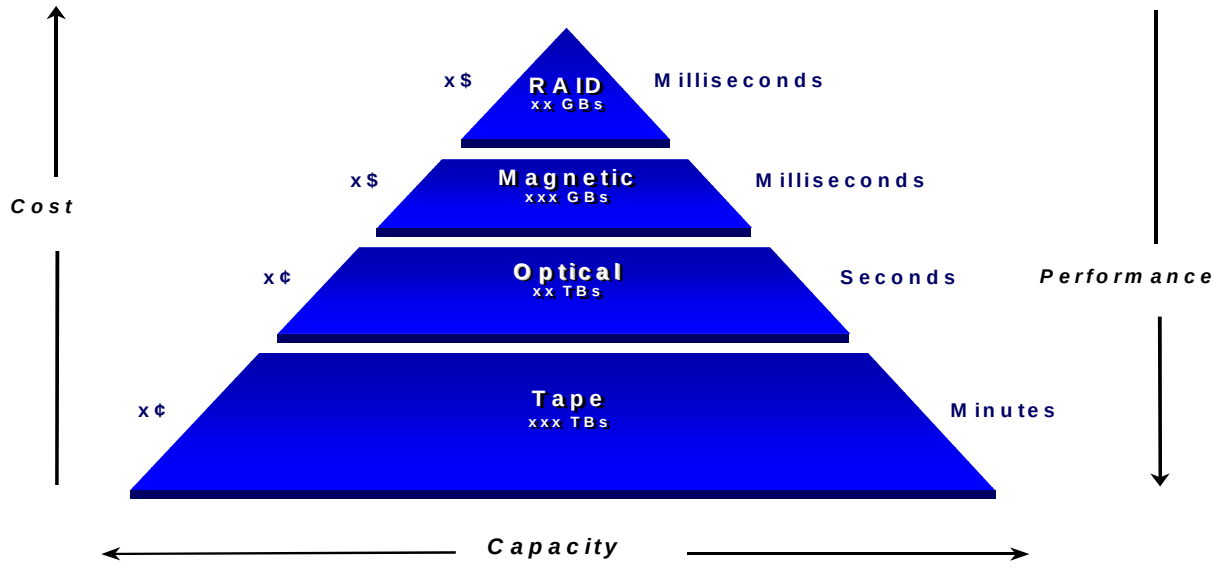



**Figure 1: Storage Cost-Benefit Triangle**

This paper is organized as follows: Section 2.0 briefly describes StorHouse/RM. Section 3.0 describes StorHouse/Fed and federated database technology and products. Section 4.0 provides an analysis of object/relational databases and StorHouse/ORM. We conclude in section 5.0 by describing candidate applications, categorizing them by their storage space needs. We then argue that many of these require the use of an active storage hierarchy in order to be cost-effective.

## 2.0 StorHouse/RM

StorHouse/RM is a database system that supports SQL-92 queries against the active storage hierarchy. This database system was designed and optimized to store atomic data on diverse media. StorHouse/RM handles the data placement issues as described in [CTZ 97].

StorHouse/RM (RM) works in conjunction with StorHouse/Storage Manager (SM) to administer the storage, access, and movement of relational data.

SQL access is available from diverse platforms through a variety of industry-standard protocols. SM and the RM engine run on the Sun Microsystems Ultra Enterprise family of computers.

Figure 2 shows StorHouse/RM architecture that is built on top of StorHouse/SM. A list of supported storage devices and media can be found at www.filetek.com this list is constantly being updated.

An RM database consists of the following:

- Table data that a user will store and access.
- Optional indexes—value, hash, and range—that locate table data.
- Metadata that describes the database components.

RM databases have both a logical and a physical structure. Logically, user tables and associated indexes reside in user tablespaces, and metadata resides in a system tablespace. Physically, user tables, indexes, and metadata are stored in SM files. RM user tables consist of one or more table segments. Each table segment is a separate SM file; the use of range indexing allows RM to restrict the number of segments processed, thus improving performance.

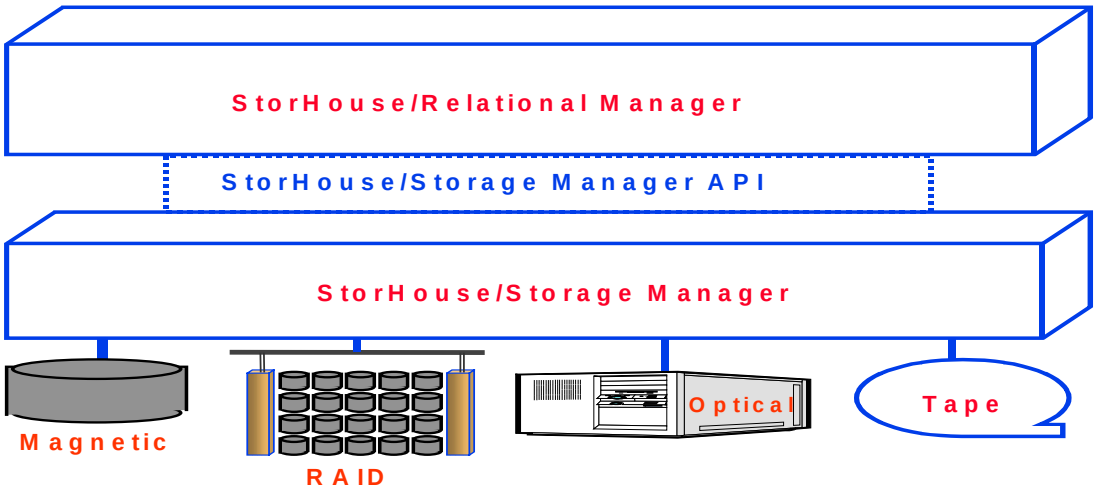



**Figure 2: StorHouse/Relational Manager**

## 3.0 Federated Databases

Federated Databases [Car 86] (or other middleware) can provide a "single system image" for diverse data sources; these products integrate the sources and provide uniform access to the data. These same (or complementary) products may also be used to migrate some or all of a user's (atomic) data to tape or optical storage from other data sources. Products in this area include IBM DataJoiner [DJ 99], Microsoft OLE/DB, Sybase OmniSQL and a new approach from Cohera.

Legacy [BS 95] applications are excellent candidates for a federated solution. It is important that such software be designed to accommodate legacy data. Either legacy data (flat files or database tables) are loaded into the federated system, or become, themselves, data sources within the system.

An active storage hierarchy may also be a data source in a federated system. From this we may see that data sources within a federated system may have widely varying capabilities and performance characteristics.

Although they may differ in details, most federators operate in the same way. A federator requires meta-data that describes the location of different data items, along with the capabilities of the various data sources. A federator receives an end-user query, which it then parses. The federator's meta-data is used to develop an access plan; elements of this plan become queries to the various data sources. The federator then combines and further refines the result sets returned by the data sources. Some federators may delegate this responsibility to one or more of the data sources themselves, depending on their capabilities.

Another federator task is that of data location. A federator may be given the job of initially assigning a data item to a particular data source, or may, during daily operation, migrate or replicate data from one source to another.

There are many ways for a federator to accomplish the job of data location. To varying degrees, these take into account both the capabilities of the data sources (especially cost-effective storage) and the nature and access patterns of the data itself. We will explain two of the simpler and more obvious scenarios:

(1) LHP - Logical Horizontal Partitioning and

(2) LVP - Logical Vertical Partitioning.

Horizontal partitioning (of rows) is used frequently, although it may not be recognized as such. Different rows from the same logical table are assigned to different data sources. One common example of this is when data rows are "aged out" from one source to another. Figure 3 shows an example of LHP.

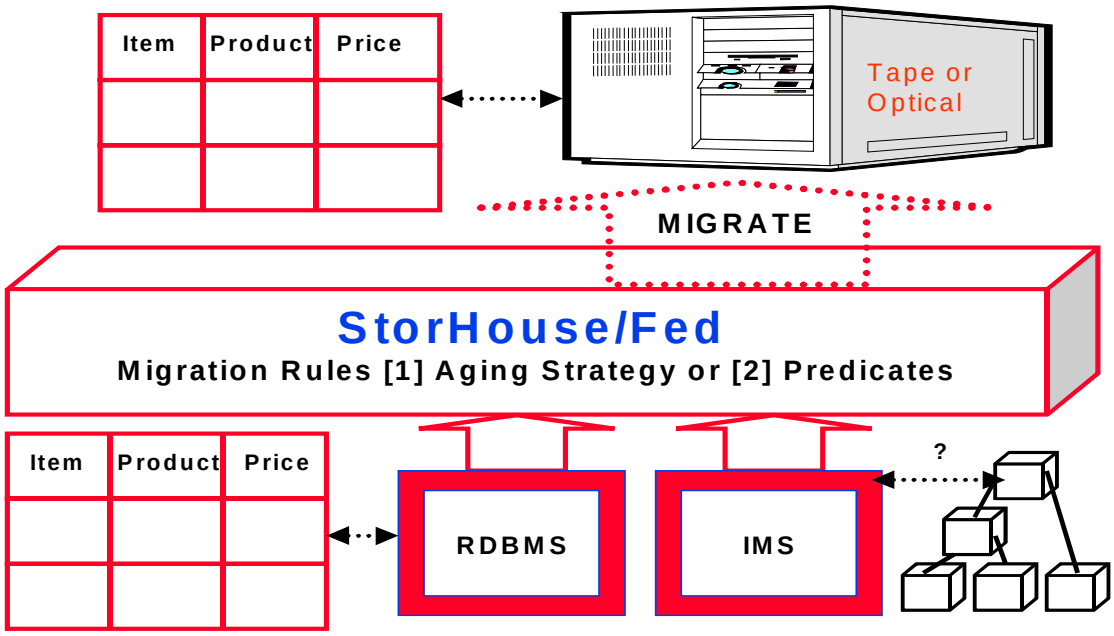



**Figure 3: Logical Horizontal Partitioning**

With LVP, different columns from the same (logical, federated) table may be assigned to different data sources (or storage media.) Figure 4 shows an LVP where text and video columns are stored on tape or optical and scalar values are stored on disk.

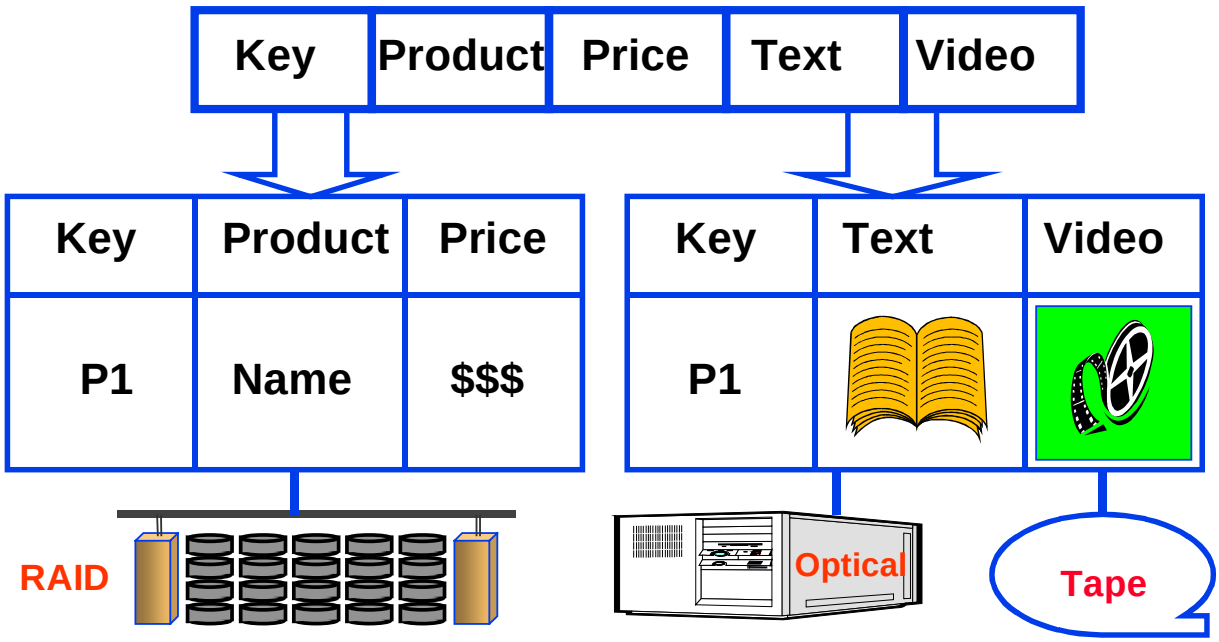



**Figure 4: Logical Vertical Partitioning**

## 4.0 Object/Relational Databases

Relational database and Object-Oriented databases are undergoing an evolution into a new database paradigm, called Object/Relational Databases [SM 98]. Relational database evolution is based on SQL-3 (to become SQL-99 in 1999) [SQL 99] whereby user-defined types (UDTs) and user-defined functions (UDFs) are added to relational database systems. Object-oriented databases, with Object Query Language (OQL) [OQL 99], are adding more relational-like navigational support.

This paper will concentrate on SQL-99 based Object/Relational Database Management Systems (OR/DBMS) since relational databases (i.e. DB2, Oracle, SQL Server, Sybase, Informix, Teradata and so on) represent over 90% of the database market. Section 5.0 describes candidate applications that can and need to use cost-effective storage hierarchy to store and retrieve data based on their content. Our arguments also apply to OQL or OODBMS systems, but we don't know of any implementation that is not disk-based, hence they are not cost-effective for very large (i.e. exabyte) database applications.

## SQL-99

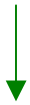
The following examples illustrate the new large object (LOB) [SM99], UDT and UDF capabilities. We provide an analysis of UDF library capabilities in the next section where we provide candidate example usage. First we show an SQL table that has both scalar and UDT columns.

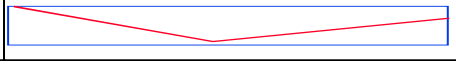
```
CREATE TABLE StockInfo (
        Stock       VARCHAR(32),
        Price       FLOAT,
        Earnings    FLOAT,
        Reports     TEXT,  -- TEXT UDT
        Film        VIDEO); -- VIDEO UDT
```

| Stock | Price | Earnings | Reports | Video |
|---|---|---|---|---|
| Felipe.com | 500 | 100 |  |  |
| Art.com | 75 | 1 |  |  |
| Pekka.com | 100 | -10 |  |  |

The next example shows an SQL-99 query where UDFs appear in both the projection list (PlotDailyValues) and in the selection predicate (MovingAverage and LastPrice). This example uses the *StockHistory* table, similar to the above *StockInfo* table, but it stores stock data in time sequence, such as a row per day for each stock.

```
SELECT Stock, LastPrice(PriceHistory), Film,     PlotDailyValues(PriceHistory,30)
FROM       StockHistory
WHERE      MovingAverage(PriceHistory,30) /
           LastPrice (PriceHistory) < .8;
```



| Stock | Price | Video | PlotDailyValues |
|---|---|---|---|
| Felipe.com | 500 |  |  |
| Art.com | 75 |  |  |
| Pekka.com | 100 |  |  |

### SQL-99 LOB and UDT Columns

SQL-99 allows LOB and UDT columns. The major difference is that LOB column values have no meaning (semantics) within the database, while UDT column values can be interpreted by the database server through UDFs. In general, LOBs can be retrieved only by querying associated row scalar values whereas UDT columns can be retrieved based on their content (using a UDF.)

A simple step would be for an application to add Image, Text and Document column data to an existing database and retrieve these via scalar predicates. The SQL-99 object/relational evolution provides functions (i.e. UDF) to retrieve data based on the content of the column.

Database vendors have realized the value in provided UDF libraries written by domain experts. User can develop their own UDF functions or libraries, but these packaged libraries provide a good head start for most applications.

The leading database and UDF libraries are shown below:

| Database System | UDF Library Name |
|---|---|
| IBM DB2 | Extenders |
| Oracle | Cartridges |
| Informix | DataBlades |

## 5.0 Candidate Applications

The fundamental point of this paper is that database systems should and, in fact, *must* directly support all forms of active storage in order to be cost-effective for large applications.

Our experience talking to customer prospects clearly shows that many applications are being written lacking critical functionality due to (1) Limitations in Technology; (2) System Cost; and (3) Business Issues.

We have discussed and referenced the technology and system cost issues. Business issues cover many complex non-technical issues (e.g. process re-engineering, funding cycles, enterprise politics, and so on) which are beyond the scope and purpose of this paper.

Table 1 classifies representative applications by database size. These numbers drive our premise that disk storage is not realistic (i.e. cost-effective) for all applications.

### Terabyte Applications - $10^{12}$ bytes

There are several terabyte-level database systems documented by [Winter 99]. The largest commercial installations are a national retailer point of sale database (~16TB) and a telecommunications call detail warehouse (~200TB.) These database systems continue to grow, but they are almost exclusively composed of scalar valued data.

### Petabyte Applications - $10^{15}$ bytes

Text (i.e. all forms of documents) and Image (I.e. all forms of pictures and images) are likely the most important data types to be incorporated into database applications.

In the product documentation example, the storage and retrieval of Text documentation (say for a large extranet application) will require the ability to search based on document content. Through UDFs and specialized index types, the database internally handles indexing and concept matching issues. A concept match would be to do a search for operating system and return documents that, for example, contain the keywords UNIX, LINUX, Solaris, MVS and NT.

For Image types, columns can be retrieved based on “image features”. A great deal of research has been done in image feature extraction and indexing, and the results of this research are beginning to appear in commercial products. Readers can learn what features are and how they may be supported by evaluating OR/DBMS UDF library capabilities. For example, the IBM DB2 Extenders for Text UDF library capabilities.

Document management can extend in the petabyte range in size. For example, to store the current printed records and documents in the Library of Congress is estimated at 10 Terabytes. For all the US academic research libraries, the current storage requirement is estimated at 3 petabytes.

**Exabyte Applications - $10^{18}$ bytes**

Historically, patient clinical and diagnostic records have been maintained on paper by different organizations with no incentive to integrate patient information. Thus, relatively few (if any) patients have a complete record of their family history, medical problems, diagnoses, treatments, medications, recent laboratory results or complications. This can cause unnecessary delays in treatment and duplicated tests. The consequences of this situation can be quite serious; it is not uncommon to have serious and even fatal reactions when incompatible medications are prescribed.

The healthcare industry, insurance companies and several federal agencies need access to diverse data (types). These include data such as: (a) text medical record, (b) doctor's progress reports, (c) written notes, (d) image data (e.g. X-rays, EKG, CAT-Scan, MRI), (e) audio data (e.g. verbal notes, progress notes, diagnosis dictation) and (f) video data (e.g. surgical procedures, patient education, nutrition counseling. The National Medical Knowledge Bank (NMKB) is a NIST_ATP program that is addressing the above issues [Ster 98]. Assuming four Gigabytes of storage for each citizen in the U.S adds up to a total of one Exabyte of storage.

**Zettabyte Applications - $10^{21}$ bytes**

Spatial databases of the solar system or even planet earth – The Digital Earth (www.digitalearth.gov) – store and retrieve data based on "spatial" (geometric) properties. Data types in this domain range from the simple (point, line, circle) to the complex (maps.) When the time dimension is added, say for weather information provided by satellite, the volume of data grows at a tremendous rate.

All of the candidate applications mentioned in this section have the potential to create very large databases. Video is in a special class by itself and has non-database issues that must be addressed as well. These include: (1) digitizing the video; (2) compressing and decompressing the video (quickly); (3) network streaming to avoid hiccups; and in the future (4) content-analysis and analysis-based retrieval.

Given this it is likely that applications will first store and retrieve video using BLOBs and only later retrieve them based on their content.

| *Database Size* | *Application* |
|---|---|
| ***Terabyte $10^{12}$ bytes*** | Telecommunication Call Detail Warehouse |
| | National Retail Point Sale Data |
| ***Petabyte $10^{15}$ bytes*** | Text and Images Product Descriptions |
| ***Exabyte $10^{18}$ bytes*** | National Medical Insurance Records |
| ***Zettabyte $10^{21}$ bytes*** | Spatial and Terrestrial Data |
| | Video and Audio Data Archives |
| ***Yottabyte $10^{24}$ bytes*** | Moore's law: Database size in 2050 |

**Table 1: Applications and Database Sizes**

There are many potential uses for video: (1) video messages to provide information over the web; (2) information (perhaps a commercial) about a product; and (3) video archives. Video may come from numerous sources, from existing television and movie archives to even video intercepted by an intelligence agency.

Estimates in [MM 98] are that all the TV stations in the world generate about 100 petabytes of data per year. Assuming that in the near future all stations will start to broadcast in High definition format, this alone will increase the data requirement by a factor of 7. Add to this all the new sources of video feeds available on the net, and the total data available will be a few exabytes in a year. [MOR 98] also estimates the storage of all audio calls in the world add up to a few exabytes a year. Imagine adding to this the video and audio sources on the Internet over the next few years, and a zettabyte of data is possible.

### Yottabyte Applications - $10^{24}$ bytes

Yottabyte applications can be expected when the technology advances to make these levels of storage feasible. Assuming the famous Moore's law will hold, we can expect a yottabyte of storage to be economically cost effective in the middle of the next century. As every database administrator knows, users always find ways to fill up any available storage. A yottabyte of storage will be no different.

## Conclusion

In this paper we briefly restated the case [CB 99] for a database system that directly supports an "active storage hierarchy" (i.e. tape, optical and disk). We described two major database technologies: federated and object/relational database systems. Federation can be used to enhance or develop new applications. We described how federation can use logical horizontal or vertical partitioning of rows or columns to exploit the storage hierarchy (i.e. tape and optical). We also described object/relational databases that provide object, functions and libraries of functions. This is a major database (and application) paradigm shift that will facilitate adding new capabilities to government and enterprise-wide applications. We concluded by describing some applications that will require cost-effective active storage hierarchy managed directly by a database management system.

## References


[BS 95] Brodie, M. and Stonebraker, M., "Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach", Morgan Kaufmann, 1995.

[Car 86] Cariño, F. "HETERO – Heterogeneous DBMS Frontend", Proceedings Conference on Methods and Tools for Office Systems, Pisa, Italy, October 1986, pp. 159 – 172.

[CB 99] Cariño, F. and Burgess, J., "StorHouse/Relational Manager (RM) –*Active* Storage Hierarchy Database System and Applications"

[CBOS99] Cariño, F., Burgess, J., O'Connell, W. and Saltz, J. "Active Storage Hierarchy, Database Systems and Applications – Socratic Exegesis", Proceedings 25th VLDB Conference, September 1999 http://www.filetek.com/vldb.htm

[CTZ 97] Christodoulakis, S., Triantafillou, P. and Zioga, F. "Principles of Optimally Placing Data in Tertiary Storage Libraries", VLDB '97, pp. 236 – 245.

[DJ 99] www.software.ibm.com/data/
datajoiner

[HS 96] Hillyer, B. and Silberschatz, A., "Random I/O Scheduling in Online Tertiary Storage Systems", SIGMOD 96, pp. 195 – 204.

[JM 98] Johnson, T. and Miller, E., "Performance Measurements of Tertiary Storage Devices", VLDB 98, pp. 50 – 61.

[MLLSKG99] Moore, R., Lopez, J. Lofton, C. Schroeder, W., Kremenk, G. and Gleeicher, M., "Configuring and Tuning Archival Storage Systems", Proceedings of 16th IEEE Mass Storage Systems Symposium, March 1999, San Diego, California.

[MM 98] Morrison, P. and Morrison, P., "The Sum of Human Knowledge?", Scientific American, July 1998.

[OQL 99] Object Data Management Group, "Object Query Language (OQL)", www.odmg.org.

[RGF 98] Riedel, E., Gibson, G. and Faloutsos, C., "Active Storage For Large Scale Data Mining and Multimedia", VLDB '98, pp. 62 – 73.

[SM 99] Shapiro, M. and Miller, E., Managing Databases with Binary Large Objects", Proceedings of 16th IEEE Mass Storage Systems Symposium, March 1999, San Diego, California, pp. 185 – 193.

[SQL 99] ANSI "SQL3" www.ansi.org

[Ster98] Sterling, W. "The National Medical Knowledge Bank", Proceedings 24th VLDB Conference, New York, pp. 637 – 640.

[TP 97] Triantafillou, P. and Papadakis, T. "On-Demand Data Elevation in a Hierarchical Multimedia Storage Server", VLDB '97, pp. 226 - 235.

[Winter99] Winter Corporation VLDB Survey, www.wintercorp.com