Association for Information Systems

AIS Electronic Library (AISeL)

ICEB 2003 Proceedings

International Conference on Electronic Business (ICEB)

Winter 12-9-2003

Tracking a Web Site's Historical Links with Temporal URLs

David Chao

Tracking a Web Site's Historical Links with Temporal URLs

David Chao
College of Business
San Francisco State University
San Francisco, CA 94132
E-mail: dchao@sfsu.edu

Abstract

The historical links of a web site include the URLs invalidated due to web site reorganization, document removal, renaming or relocation, or links to document snapshots, which are defined as the document's contents as of a specific point in time. Tracking historical links will allow users to use out-of-date URLs, retrieve removed documents and document snapshots. This paper presents a logging and archiving scheme to track a document's history of changes, and a Temporal URL scheme for users to submit a URL with temporal requirements. With the proposed schemes, a web site is able to track its historical links and provide better searching and information for its users.

1. Introduction

A web site is a system of integrated web documents embedded with links to reference related documents. At any given time, a Uniform Resource Locator (URL) uniquely identifies a document on the Internet. All URLs have the same format: <protocol>://<host>/path-to-document. The <protocol> specifies the method to be used by the browser to communicate with the web server. Common protocols include http, ftp, gopher, etc. The host name is the web server that stores the document. The path to the document is a sequence of directory names and the document name. URLs are also known as the addresses, or paths, of documents on the Internet.

URLs are temporal and exist only for a limited period of time. There are many causes that will render a URL invalid: the host may cease to exist; the host may reorganize by changing its directory structure, hence changing the path to a document; a document may be renamed, deleted, or moved to a different directory. The document may no longer exist or have a new URL.

URLs are published information that may be out-of-date. Internet users gather the URLs of their interests from many sources such as references in books and documents, web page hyperlinks, results of searching with search engines, etc. Very often those published URLs may already be invalid. Requesting a document with an invalid URL may cause a file-not-found error (HTTP 404). Some websites do not handle this error and simply let the browser display the error code. Some web sites redirect users to a default error-handling page typically equipped with search tools. Searching takes time and may not be successful. Searching web sites storing a large amount of documents is particularly tedious and unfruitful. It may produce a large number of relevant documents but may not pinpoint the document requested by users.

The file-not-found error may be triggered by many causes such as: 1. The document may have never existed in the web site. 2. The document may be deleted. 3. The document may be renamed or relocated to another directory. Rather than simply displaying a file-not-found message, a web site will improve its services by handling each cause separately. For the first case, returning a message such as "file never exists" is more informative. For the second case, returning a message such as "file no longer supported" is appropriate. For the third case, the web site can redirect users to the document's new URL.

Even valid URLs don't necessarily retrieve the user's desired document. A URL may point to a document that is different from the one it pointed to earlier. This may happen because of the renaming of documents and directories. For instance, a document A may be renamed to C and another document B may be renamed to A; hence the URL originally pointing to A is now point to a different document. It may also happen when a web site removes a URL but later reinstates the URL for another document. Users submitting such a URL may get an unwanted document.

A document may have gone through several changes. A URL typically retrieves the current version of a document and its contents may be different from the last download. There are times when users would like to see the previous versions of a document. These previous versions are snapshots of a document. A document snapshot is defined as the document's contents as of a specific point in time. Studying a document's snapshots may help users identify trends and patterns of changes in a document.

Together, the URLs invalidated by a web site due to reorganization, document removal, renaming, or relocation, plus the links to document snapshots are a web site's historical links. Tracking historical links basically means recording the time and the type of changes that occurred to a document and its URL. It enables a web site to retrieve the document associated with an out-of-date URL submitted by users, deleted documents, and document's snapshots. This will improve

Web Document

Web site

	Added	**No change**	**Relocated/ Renamed**	**Modified**	**Deleted**
No change	New URL	URL current	Old URL invalid/ New URL	URL current/ Archive snapshot	Old URL invalid/ Archive
Reorganized	Old URL invalid/ New URL	Old URL invalid/ New URL	Old URL invalid/ New URL	Old URL invalid/ New URL/ Archive snapshot	

Figure 1: Effects of web site reorganization and web document changes.

a web site's searching ability and provide better information for users.

Tracking changes has been the research topic in many areas such as managing database snapshots [1], materialized views [4], and document versions [6]. Those researches may focus on tracking the changes (deltas) [5] or tracking document's snapshots. Using a log to record changes [2] [3] and archiving the many versions of a document are typical methods in tracking changes. This research proposes a logging and archiving scheme for tracking historical links and proposes a Temporal URL scheme to retrieve document snapshots. The objectives of this scheme are: 1. For an invalid URL the scheme will be able to distinguish if the document is non-existent, or no longer supported, or it has a new URL so that appropriate action can be taken. 2. For a valid URL the scheme will allow users to retrieve the document snapshots. 3. For a valid URL that retrieves an incorrect document, the scheme will allow users to resubmit the URL and inform the web site to process the URL as a historical link.

An analysis of the problem and the designs of solutions are presented in Section 2. The temporal URL scheme and algorithms for processing historical links are presented in Section 3. Section 4 is a summary.

2. Problem Analysis and Solution Designs

2.1 Problem analysis

Figure 1 summarizes the effects of web site reorganization and change to a document on its URL. When a web site reorganizes its directory structure, all affected documents will have a new URL. Similarly, when a web document is renamed or relocated to other directory, it will have a new URL as well. When a document is modified, its URL will not change and the old version before the modification becomes the document's snapshot between the previous and the current modification. In order to retrieve the snapshot, either the difference between the old and the current version needs to be recorded so that the snapshot can be reconstructed, or the old version is archived. Tracking the difference between two unstructured documents such as web documents is not easy. Here, archiving is assumed. A deleted document is archived as well for later retrieval.

Figure 2 illustrates the progression of a web site. T_0 is the time when the web site is initiated, and T_i denotes a point in time when its contents change. Period i is the interval of time between T_i and T_{i+1}. Assuming the path of a valid URL is unique within the web site, a valid path leads to a document currently published. In the sequel, paths and URLs are used interchangeably. There are two sets of paths in period i: VP_i , a set of all valid paths, and $HP_{i,}$ a set of paths becoming historical at $T_{i.}$ Note that HP_0 is null. In Figure 2, paths that produce two dotted arrow lines such as P1 in period 0, and P4 and P5 in period 3 represent documents that are renamed or relocated. In this case, only the document's path has changed but the document has not changed. Therefore, there is no need to archive any documents, but it is necessary to chain the old path to the new path in order to track the change. Paths that produce two solid arrow lines such as P3 in period 1, and P3 and P5 in period 2 represent documents that are modified. In this case, the path is still valid but the document before the change needs to be archived as the document's snapshot. Paths that produce one solid arrow line, such as P2 in period 1 and P3 in period 3, represent documents that have been deleted. Those documents must be archived.

A path in HP_i may be repeated in HP_j, just as P3 is repeated in HP_2 and HP_3. To make a historical link unique, the time the path was published is added to the path. Hence P3 in HP_2 is identified as P3 + T_1 and P3 in HP_3 is identified as P3 + T_2. Similarly, a current path may duplicate a path in the historical links, as is the case of P1. By adding its published time, P1 + T_3 is distinguishable from the historical link P1 + T_0. The historical links of a web site before T_i are the union of sets HP_0 to HP_{i-1}, (HP_0 U HP_1 U … U HP_{i-1}). Assuming T_i is the last time the web site changed then the web site's historical links are the unions of sets HP_0 to HP_i, and the current links are VP_i. The union of historical links and the current links represents all the paths that are currently in use or have been used in the past. Every link in the union is uniquely identified by the value (path + path published time). This composite value is an example of a

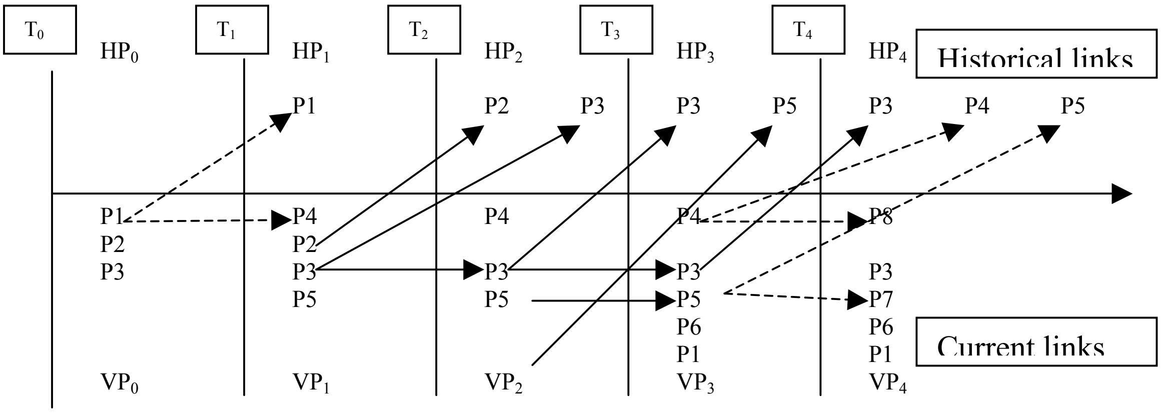

Figure 2: An example of the progression of a web site with changes since its creation.

Temporal URL (discussed in Section 3). For any path Px published at time T, the temporal URL (Px + T) uniquely identifies a document that is currently or was once associated with Px at time T.

2.2 Solution Designs

This section explains the scheme to track the historical links. The scheme has two major components: logging the changes to web documents and archiving deleted documents including document snapshots. The log, named TemporalURLLog, is designed to keep the history of changes to web documents. It has four fields: URL, PublishDate, ExpireDate, and NewURL. The URL field records a document's path; the PublishDate records the time the document is published; the ExpireDate field records the time this URL becomes invalid; should the change create a new URL for the document, the NewURL field records the document's new URL and serves as a link to chain all log entries related to the same document together. This log has a composite key of URL + PublishDate. The value of URL + PublishDate uniquely identifies a document in a web site's history. The Archive is a directory that stores deleted files and document snapshots. Those archived files are saved in the Archive using URL + PublishDate as file name. The log is maintained according to the log maintenance algorithm described below:

TemporalURLLog maintenance algorithm Changes to web documents are recorded in the log by the following rules:

New document: When a new document is added to the web site, a new entry is entered with its path and the time the document is published. The ExpireDate and the NewURL are set to null. Hence, log entries with a null ExpireDate are current document entries. Initially, all current documents have an entry in the log with null ExpireDate and null NewURL.

Deleted document: The log entry associated with the document is the entry of which the URL equals the document's URL with a null ExpireDate. First, it locates the entry and changes its ExpireDate to the time the document is deleted. Then, it saves the deleted document in the Archive with URL + PublishDate as file name.

Modified document: When a document is modified, its old version becomes a snapshot and its new version is treated as a new document. The log entry associated with the document is the entry of which the URL equals the document's URL with a null ExpireDate. First, it locates the entry and changes its ExpireDate to the time the document is modified. Then, it saves the old version in the Archive with URL + PublishDate as file name. Then, it adds a new entry with the same URL and the PublishDate is set to the time the document is modified. The ExpireDate and NewURL are set to null.

Consequently log entries with a non-null ExpireDate and a null NewURL may be related to snapshots or deleted documents. For such an entry, if there exists another entry with the same URL and its PublishDate equals to this entry's ExpireDate then this entry is a snapshot entry and the snapshot it associates with can be retrieved from the Archive using URL + PublishDate as its file name. The snapshot is valid from the PublishDate to the ExpireDate. Otherwise, the entry is associated with a deleted document which can be retrieved from the Archive using URL + PublishDate as its file name.

Relocated or renamed page: When a document is relocated or renamed, its old URL is expired and a new URL is created. The log entry associated with the document is an entry with the URL equal to the document's URL and with a null ExpireDate. First, it locates the entry and changes its ExpireDate to the time the document is relocated or renamed and changes its NewURL field to the document's new URL. Then, it adds a new entry with the new URL and the PublishDate is set to the time the document is relocated or renamed.

Hence, log entries with non-null NewURL field help chaining a document's log entries.

Using this log maintenance algorithm for the changes described in figure 2, the contents of the TemporalURLLog are shown in Figure 3, and there are five files in the Archive: $P2T_0$, $P3T_0$, $P3T_2$, $P5T_1$, and $P3T_3$.

URL	PublishDate	ExpireDate	NewURL
P1	T_0	T_1	P4
P2	T_0	T_2	Null
P3	T_0	T_2	Null
P4	T_1	T_4	P8
P5	T_1	T_3	Null
P3	T_2	T_3	Null
P3	T_3	T_4	Null
P5	T_3	T_4	P7
P6	T_3	Null	Null
P1	T_3	Null	Null
P8	T_4	Null	Null
P7	T_4	Null	Null

Figure 3: The contents of TemporalURLLog based on changes described in Figure 2.

3. Searching with TemporalURLLog

With the TemporalURLLog, a web server is capable of processing historical links involving deleted documents, out of date URLs, and document snapshots. For instance, using the log in Figure 3, it is possible to determine that:

. A URL P2 valid between T0 and T1 is deleted, and the document it pointed to is in the Archive with the name $P2T_0$.

. A URL P3 has been modified repeatedly and is eventually deleted. All documents associated with P3 can be found in the Archive.

. An old URL P5 is now renamed to P7. It has been modified on $T_{3,}$ and a copy of its snapshot can be found in the Archive with the name $P5T_{1.}$

. The log is able to determine that a historical line P1 is now renamed to P8.

. A URL P12 has never existed in the web site.

This section presents a Temporal URL scheme and algorithms for searching documents associated with historical links.

Requesting historical documents with temporal URL

A temporal URL is a URL submitted with temporal requirements of which the documents associated with the URL must meet. A typical way to submit additional information with a URL is through query strings. A query string is a set of name=value pairs appended to a URL. It is created by adding a question mark (?) immediately after a URL followed by name=value pairs separated by ampersands (&). Two types of temporal URLs are designed:

1. The URL is not current: When the URL is not current, a keyword IsHistorical can be appended to the URL to indicate users are searching for documents associated with a historical link. The query string is:

?IsHistorical

2. The URL is current: The document associated with the URL may have gone through modifications. There are situations where users may need to access a document's snapshots at different points of time. Special keywords can be created for users to specify temporal requirements for snapshots. Some typical keywords are:

?SnapshotAsOf=*date*
?SnapshotsBefore=*date*
?SnapshotsAfter=*date*
?SnapshotsBetween=*date1*&And=*date2*

The ?SnapshotAsOf=*date* query string will retrieve a document's snapshot at the specified date; others will retrieve all snapshots that meet the date criteria. A web site may choose to implement the types of temporal query strings to fit its needs.

Algorithms for processing temporal URL are presented below.

Forward search to locate documents associated with a historical link

This algorithm processes the ?IsHistorical temporal query string submitted with a URL Px and searches the TemporalURLLog to locate all the documents related to a historical link. In the TemporalURLLog, entries for a URL may have three patterns of changes: 1. If a URL has a log entry with a non-null PublishDate and null ExpireDate field then it is a current URL; such as P6 in figure 3. 2. If all entries of a URL have a non-null PublishDate, ExpireDate and null NewURL field, then this URL is deleted from the web site; such as P3 in Figure 3. 3. If a URL has a log entry with a non-null NewURL field, then it has been renamed, and the log entries for the new URL may again have these three patterns of changes. Because a URL may be used several times in the life of a web site, such as P1, this algorithm will locate all documents associated with the historical link.

In the algorithm, SetPx is a set of records where URL equals Px. If SetPx is null then Px never existed. Otherwise, a loop is used to study change pattern. It applies a forward search that will trace down all related entries until reaching the last entry. To move forward, an entry's successor can be found and processed by the following rules:

1. If the entry has a null NewURL then its successor must have the same URL and the successor's PublishDate

```
Algorithm Forward Search
Inputs: TemporalURLLog, historical link Px
Outputs: Links to all documents related to Px

SetPx = { Select * From TemporalURLLog Where URL = Px  And Not ExpireDate = Null
                                    Order By PublishDate }
If SetPx isNull Then
    Return Message: Document never exists
Else
    For Each Record R in SetPx
        If R.NextURL Is Null Then
            If Not Exist {Select * From TemporalURLLog Where
                   URL = R.URL And PublishDate = R.ExpireDate}  Then
                Return LinkArchive(R.URL + R.PublishDate)
            End if
        Else
            PxLink = R.NextURL
            Do
                SetPxLink = { Select * From TemporalURLLog Where URL = PxLink
                            Order By PublishDate }
                NextLoop = False
                For Each Record R in SetPxLink
                    If R.ExpireDate isNull Then
                        Return LinkDocument(R.URL)
                        Exit Do
                    Else
                        If R.NewURL IsNull Then
                          If Not Exist {Select * From TemporalURLLog Where
                           URL = R.URL And PublishDate = R.ExpireDate} Then
                              Return LinkArchive(R.URL + R.PublishDate)
                              Exit Do
                          End if
                        Else
                          PxLink = R.NewURL
                          NextLoop = True
                          Exit For
                        End if
                    End if
                Next
                If NextLoop = False Then
                    Exit Do
                End if
            Loop
        End if
    Next
End if
```

must equal the entry's ExpireDate. If no such successor is found, then this entry must have been generated due to a deletion and its archive name is created with this entry's URL and PublishDate. This algorithm returns a link referencing the archived document.

2. If the entry has a non-null NewURL then it must have a successor with a URL equal to the NewURL and the PublishDate equals to the entry's ExpireDate. Renaming or relocation must have generated the successor. When such a successor is found, a new loop is started to trace down the changes in the new URL. Although Px is not current, this new URL can be current if there exists an entry with the new URL and a null ExpireDate; a link referencing the new document is returned. Otherwise, the new URL eventually is deleted and the algorithm returns a link referencing the archived document.

To illustrate, if P5?IsHistorical is submitted, initially SetPx contains two entries: (P5, $T_{1,}$ T_3, Null), (P5, $T_{3,}$ T_4, P7). The first entry's NewURL is null but the next entry's PublishDate equals the first entry's ExpireDate indicating that this is a modification. The second has a non-null NewURL indicating that it has been renamed. After processing the second entry, a new loop is started using the NewURL P7 to search. In the second loop, SetPx contains: (P7, T_4, Null, Null). Since its ExpireDate is null, this is a current URL and P7 is returned.

```
Algorithm Backward Search
Inputs: TemporalURLLog, valid URL Px, snapshot time T
Output: Px's snapshot at time T

CurrentEntry = {Select PublishDate From TemporalURLLog Where URL = Px and ExpireDate = Null}
If CurrentEntry.PublishDate <= T Then
    Return Document(URL)
Else
    Candidate.URL=CurrentEntry.URL
    Candidate.PublishDate=CurrentEntry.PublishDate
    Candidate.ExpireDate=Null
    PD=CurrentEntry.PublishDate
    Do
        PreviousEntry={Select * From TemporalURLLog Where URL=Px And ExpireDate=PD}
        If PreviousEntry Exists Then
            Candidate.URL=PreviousEntry.URL
            Candidate.PublishDate=PD
            Candidate.ExpireDate=PreviousEntry.ExpireDate
            PD=PreviousEntry.PublishDate
            If PD < =T Then
                Return Archive(Candidate.URL + Candidate.PublishDate)
                Exit Do
            End if
        Else
            PreviousEntry={Select * From TemporalURLLog Where NewURL=Px And
                                    ExpireDate=PD}
            If PreviousEntry Exists Then
              PD=PreviousEntry.PublishDate
              IF PD < =T Then
                If Candidate.ExpireDate=Null Then
                    Return Document(Candidate.URL)
                Else
                    Return Archive(Candidate.URL+ candidate.PublishDate)
                End if
                Exit Do
              Else
                Px = PreviousEntry.URL
              End if
            Else
              If PD > T Then
                Return Message: Snapshot Does not exist at T
              Else
                If Candidate.ExpireDate=Null Then
                    Return Document(Candidate.URL)
                Else
                    Return Archive(Candidate.URL+ candidate.PublishDate)
                End if
              End if
              Exit Do
            End if
        End if
    Loop
End if
```

If P1?IsHistorical is submitted, initially SetPx contains only one entry: (P1, T_0, T_1, P4). Since its NewURL is non-null, a new loop is started and SetPxLink is created using P4 to search; SetPxLink contains one entry: (P4, T_1, T_4, P8). Because this entry's NewURL is non-null, a new loop is started to create a new SetPxLink with one entry: (P8, T_4, Null, Null). This indicates P8 is current and a reference to it is returned.

Backward search for a document's snapshot at a specific date This algorithm processes the query string ?SnapshotAsOf=*T* submitted with a URL Px and searches for the snapshot based on the specified date T. It assumes Px is a current URL to simulate a scenario in which users who are viewing a current document would like to view the document's snapshot at a point in time. The algorithm is briefly explained below.

It processes log entries backward starting from a document's current entry to trace back its changes in order to locate the snapshot. If the current URL's PublishDate is less than T, then the current document itself is its snapshot at T. Otherwise the backward search starts. To trace back, an entry's predecessor has one of the following properties:

1. Its URL equals the current entry's URL and its ExpireDate equals the current entry's PublishDate. This indicates the predecessor is an old version of the document and it is the document's snapshot from the predecessor's PublishDate to its ExpireDate.

2. Its NewURL equals the current entry's URL and the ExpireDate equals the current entry's PublishDate. This indicates the predecessor has been renamed or relocated.

Initially, the current document is treated as a candidate for the snapshot. Before reaching past time T, whenever an old version is found it becomes the new candidate for the snapshot. Note that an old document's life may span beyond its PublishDate. If a document Y is originally derived from a document X through a series of renaming or relocation, then this document's life span is from document X's PublishDate to document Y's ExpireDate. The search reaches the last entry when its PublishDate is before T or it no longer has a predecessor. If the last entry's PublishDate is greater than T then snapshot does not exist at T.

To illustrate, if a request P7?SnapshotAsOf=T_2 is submitted, the backward search will pick up three entries: (P7, T_4, Null, Null), (P5, T_3, T_4, P7), and (P5, T_1, T_3, Null). They show that the document associated with P7 has been unchanged since T_3 and was originally associated with P5 and renamed to P7 at $T_{4.}$ The document has been modified at $T_{3.}$ Therefore, the algorithm returns Archive(P5 + $T_{1)}$ as snapshot. If P7?SnapshotAsOf=T_3 is submitted, the backward search will pick up two entries: (P7, T_4, Null, Null)and (P5, T_3, T_4, P7) and return Document(P7) as snapshot.

If a request P8?SnapshotAsOf=T_0 is submitted, the backward search will pick up three entries: (P8, T_4, Null, Null), (P4, T_1, T_4, P8) and (P1, T_0, T_1, P4). These show that the document associated with P8 has been renamed at T_1 and $T_{4.}$ It has not been modified since T_0. The algorithm will return Document(P8) as snapshot.

This algorithm can be modified to process ?SnapshotsBefore=*date*, ?SnapshotsAfter=*date*, and ?SnapshotsBetween=*date1*&And=*date2* temporal query strings by modifying the temporal conditions in the algorithm.

4. Summary

The historical links of a web site include the URLs invalidated due to web site reorganization, documents' removal, renaming or relocation, plus links to document snapshots. Tracking historical links will allow users to use out-of-date URL and retrieve removed documents and snapshots. This paper has two contributions: 1. It presents a logging and archiving scheme to track a document's history of changes. These changes include a document's creation, modifications, web site reorganizations, and deletion, which are recorded in a log. The log records a URL's publication date, expiration date, and the new URL it is assigned to, if applicable. Document snapshots and deleted documents are archived to support users who need historical documents. 2. It presents a temporal URL scheme for users to submit a URL with temporal requirements of which the documents associated with the URL must meet. A URL will retrieve a current document by default. With the temporal URL users may inform the web server to treat the URL as a historical link, or request the document's snapshots. Algorithms for processing the log and the temporal URL are also presented. With the proposed schemes, a web site is able to track its historical links and provide better searching and information for its users.

Reference:

[1] Adiba, M. & Lindsay, B. (1980). Database snapshots. Proceedings of the 6th International Conference on Very Large Data Bases, pp. 86-91.

[2] Chao, D. (2002). The Design of A Web Snapshot Management System for Decision Support Applications. Proceedings of the 2nd International Conference on Electronic Business, Taipei, Taiwan, 2002.

[3] Chao, D., Diehr, G., & Saharia, A. (1996) Maintaining Join-based Remote Snapshots Using Relevant Logging. Proceedings of the Workshop on Materialized Views, ACM SIGMOD, Montreal, Canada, 1996.

[4] Labrinidis, A. & Roussopoulos, N. (2000). Webview Materialization. ACM SIGMOD International Conference on Management of Data, May 14-19, 2000.

[5] Marian, A., Gregory Cobena, S., & Mignet, L (2001) Change-Centric management of Versions in an XML Warehouse. Proceedings of the 27th VLDB Conference, Roma,Italy, 2001.

[6] Chien, S., Tsotras, V., & Zaniolo, C. (2001) Efficient Management of Multiversion Documents by Object Referencing. Proceedings of 13th International Conference on Very large Data Bases, 2001.