Chapter 5

Production of Context Pages

Production of cases is discussed in Chapter 4. A case is a MPMSD, produced in the context of a set of relevant documents, where each document is again a MPMSD. The documents relevant to all the cases of a DPW, in general, constitute the DPW context for that DPW. A case may have a set of documents relevant specifically to itself. This set of specific documents and the DPW context of the corresponding DPW constitute the Case Context of the case. In this Chapter, production of both DPW Context as well as Case Context are discussed. Moreover, production of a state of the context, or of a part there of, based on a given time is a major issue in DPW.

5.1 Production of DPW Context

We recall that a DPW context is a collection of pages of category document, relevant to a DPW. Therefore, a page of category context has links to a set of pages of category document. A DPW has only one context page. Now, the issue is how to define the relevance of pages to a DPW. In conventional information retrieval systems, relevance is normally defined by a
set of keywords. There exists models, like Vector Space Model \[11, 16\], Latent Semantic Indexing \[11\] etc., for retrieval of relevant documents based on keywords. Keywords alone do not adequately describe an office document. Key words are only a part of the profile of an office document. Other parts are, record attributes, as discussed in the storage framework in Chapter 2, section 2.2.2. These record attributes are represented in the Page Cube as dimensions. Therefore, keyword based definitions of relevance in DPW in an office are not adequate. The relevance of pages to a DPW can be defined by the designer of a DPW using a set of requirement templates.

**Definition 5.1** A requirement template is a predicate defined by a set of attribute-value pairs. The attributes are dimensional attributes of the page cube. The attribute-value pairs are in conjunctive form, whereas, the list of values of an attribute are in disjunctive form. A range of values is an alternative form of expressing a list of values.

Requirement templates based on keywords can also be easily designed. Since well developed models on keyword-based document retrievals are available, we will exclude it from our discussion. But these models can easily co-exist with our Page Cube model.

The basis of construction of the context page of a DPW is a set of requirement templates. Using this set of requirement templates, a query is framed and the result set of the query is the context page of the DPW. For example, a template

\[ T_1 = \{\text{category}="\text{document}"\ , \text{class}="\text{rule}"\ , \text{topic}="s1, s2, s3"\ , \text{time}="t1-t2"\} \]

signifies that documents containing rules on topics s1 or on s2 or on s3, produced during the time period t1 to t2 are relevant to the DPW concerned. Sometime, a single document, relevant to the DPW, may have to be included
in the context of the DPW. For such a single document, the corresponding
template consists of only pageId, pageId = pid, where pid is the pageId of
the page.

5.1.1 An Algorithm for DPW Context Maintenance

The context of a DPW can be obtained by searching the entire Page Cube for
pages matching the given requirement templates. But this will be expensive
to do for every document that is to be created. Therefore, in this section an
algorithm for maintaining the DPW context based on a set of requirement
templates is discussed. The dynamic nature of the context associated with
a DPW is due to the following facts. When a new template is added to
the template list by a reviewer during case examination, the set of pages of
category document, satisfying the template are to be automatically plugged
to the context, if it is not already plugged. It is to be mentioned here, that
for the pages of category part corresponding pages of category document
need to be plugged only. When an existing template for a DPW is removed
from the set of templates, corresponding pages plugged to the context are to
be unplugged. Since the result-sets of templates are not necessarily disjoint,
some pages may belong to the result-sets of templates other than the template
to be removed. Such common pages are not to be unplugged. We recall,
that unplugging does not necessarily mean deletion of the edges from the
Category Graph. It simply means changes to the temporal label of the edges.
The history of plugging and unplugging is the basis of creation of the case
context of a precedent as discussed in the next section. When a new page
of category document is registered in the Page Cube, if it satisfies any of
the templates of a DPW, it is to be plugged to the context of the DPW
automatically. Moreover, when a page is burned, if the page is of category
document then if it satisfies any of the templates of a DPW then it is to be unplugged. Lastly, when an existing template is modified, then consequent changes are to be done to the context.

Let $T_w = \{T_i | T_i \text{ is a requirement template for dpw} = ''w'' \}$ be a set of requirement templates for a DPW $w$, $R_i$ be the result-set (of pages) of a query formed by using $T_i$, and $P$ be a Page Cube. The result-sets of the templates in $T_w$ are not necessarily disjoint. A result-set may contain many pages and at the same time a page may belong to many result-sets of templates of $T_w$. To take care of the relation between pages and templates for each DPW a separate Page Table can be maintained, where rows represent pages and the columns represent templates. If a page $P_i$ belongs to the result set of a template $T_j$ of $T_w$ then the $(i, j)$ element of the page table will be equal to 1, 0 otherwise. The page table will normally be a sparse binary matrix. Therefore, we can choose an alternative representation of the page table, called Page List, which is a row-wise representation of the page table. With each page a list of corresponding templates in the row with only non-zero entries will be stored. For a DPW $w$, the structure of the page list is 

\begin{align*}
\text{PageList}_w(\text{Page, TList})
\end{align*}

For example, if a page $p$ belongs to the result sets of templates $T_1, T_5, T_7$ then the tuple in the PageList$_w$ will be $(p, TList_p)$, where $TList_p = \{T_1, T_5, T_7\}$. With this an algorithm to maintain a DPW context is designed as follows:

\textbf{ALGORITHM DpwContextMain($T_i$ or $p$, $P$, $w$, $T_w$, PageList$_w$)}

Algorithm to maintain a DPW Context using requirement templates

\textbf{INPUT}:

$T_i$ : a requirement template which is for either inclusion or exclusion from $T_w$

$p$ : a page which is either newly registered or burned
\[ P \]: a page cube
\[ T_w \]: a set of requirement templates for a DPW \( w \)
\( \text{PageList}_w \): list representation of page table for a DPW \( w \), as described above

**OUTPUT:**

\( status \): a flag, on successful completion the algorithm returns \( status = 1, 0 \) otherwise

**ASSUMPTIONS:**

\( status = 0 \) initially

**NOTATIONS:**

\( \text{include}(x, X) \): a function which includes \( x \) in the set \( X \) as a new member

\( \text{exclude}(x, X) \): a function which excludes \( x \) from \( X \)

\( R_\ast \): result-set of a query

\( \sigma_{T_i} P \): select pages from \( P \) satisfying the condition formed by template \( T_i \)

\( p_i \times p_j \): page \( p_i \) plugs page \( p_j \)

\( p_i \div p_j \): page \( p_i \) unplugs page \( p_j \)

\( p_{cw} \): DPW Context page for the DPW \( w \)

\( T_i(p) \): check whether page \( p \) satisfies a condition formed by template \( T_i \)

\( element.attribute = value \): the structure is a short form of writing the value of the attribute of an element

**BEGIN**

begin case

**step 1:** case 1: a new template \( T_i \) is submitted for inclusion in \( T_w \)

**step 1.1:** \( T_w \leftarrow \text{include}(T_i, T_w) \)

**step 1.2:** \( R_\ast \leftarrow \sigma_{T_i} P \)

**step 1.3:** for every \( p \in R_\ast \)

**step 1.4:** if \( \exists \) a tuple \( t \in \text{PageList}_w \), where \( t.page = p \) then

107
step 1.5: \[ T_{\text{List}} \leftarrow \text{include}(T_i, T_{\text{List}}) \]
    // update the template list of existing page
    
else

step 1.6: \[ T_{\text{List}} \leftarrow T_i \] // new entry in PageList
step 1.7: \[ \text{PageList} \leftarrow \text{include}(p, T_{\text{List}}), \text{PageList} \]
step 1.8: \[ p_{\text{cw}} \times p \] // plug the page to the context page

endif
end for

step 2: case 2: an existing template \( T_i \) is submitted for exclusion from \( T_w \)
step 2.1: \[ T_w \leftarrow \text{exclude}(T_i, T_w) \]
step 2.2: \[ R_i \leftarrow \sigma_{T_i} P \]
step 2.3: for every \( p \in R_i \)
step 2.4: select \( (p, T_{\text{List}}) \) from PageList
step 2.5: \[ T_{\text{List}} \leftarrow \text{exclude}(T_i, T_{\text{List}}) \]
step 2.6: if \( T_{\text{List}} = \text{null} \) then
step 2.7: \[ \text{PageList} \leftarrow \text{exclude}(p, T_{\text{List}}), \text{PageList} \]
step 2.8: \[ p_{\text{cw}} \div p \] // unplug the page from the context page

endif
end for

step 3: case 3: a new page \( p \) of category="document" is registered
step 3.1: if \( \exists T_i \in T_w \) and \( T_i(p)="true" \) then
step 3.2: \[ \text{PageList} \leftarrow \text{include}(p, T_{\text{List}}), \text{PageList} \]
step 3.3: \[ p_{\text{cw}} \times p \] // plug the page to the context page

endif

step 4: case 4: a document \( p \) of category="document" is burned from \( P \)
step 4.1: if \( \exists T_i \in T_w \) and \( T_i(p)="true" \) then
step 4.2: \[ \text{PageList} \leftarrow \text{exclude}(p, T_{\text{List}}), \text{PageList} \]

108
step 4.2: \( p_{cw} \div p \) //unplug the page from the context page

endif

step 5: case 5: a template \( T_i \) is modified to \( T'_i \)

step 5.1: do step 2 to exclude \( T_i \) from \( T_w \)

step 5.4: do step 1 to include \( T'_i \) to \( T_w \)

step 6: return(1)

dend case

END

This algorithm maintains the DPW context of a DPW \( w \). All the five cases, where the context may change, are taken care of in the algorithm.

In case 1, when a new template is added to the template list if the page \( p \) is already plugged, then only the \( TList_p \) of \( p \) is modified by including the template (step 1.5) otherwise a new page entry is made in the \( PageList_w \) (step 1.7), and the page is plugged to the context page (step 1.8). In case 2, when a template is excluded from the template list, the result-set of the template is regenerated and for every page belonging to the result-set the template is excluded from the corresponding \( TList \). After exclusion, if the \( TList \) of the corresponding page is not empty, it means that the page is also common to the result-sets of other templates in the list. Hence, the page is not unplugged. If the \( TList \) is empty then the page is not common to any other template and hence its entry in the PageList is deleted and it is unplugged from the context page. Case 3 and 4 are logically simple and straightforward. But the efficiency depends on how to find out whether the page satisfies any template of a DPW, and this depends on the data structures that are used. Similar situations are handled in Active Databases using rules [11]. Modification of a template is a two step process: exclusion of a template and inclusion of a new template.
From the perspective of security, we can put the following restriction on operations on a context. A reviewer cannot unplug any page from the context page of a DPW but can plug new pages by incorporating suitable requirement templates in the template set. Only the designer of the DPW, who may be a higher authority in an office, not necessarily a reviewer of the DPW, can exclude or modify an existing template from the template list of a DPW.

5.1.2 Retrieval of a DPW context

Once a DPW context is maintained using the algorithm $DpwContextMain()$ discussed in the previous section, the question arises as to how to retrieve the current state or the state at a particular time of a DPW Context. A $GetDpwContext()$ algorithm can be designed, similar to the $GetDocument()$ algorithm discussed in Chapter 3. A DPW context will have only $docPart$ kind of edges.

ALGORITHM $GetDpwContext(w, t)$

Algorithm to retrieve the state of the DPW context page $p_w$, for the DPW $w$, at time $t$

INPUT:

$w$: dpwId of a DPW

$t$: time which defines the state of $p_w$

OUTPUT:

$p^t_w$: state of the page $p_w$ at time $t$

ASSUMPTIONS:

A DPW has only one DPW Context page

NOTATIONS:

$P$: a page cube
null: a constant with value 0

now: a timestamp, which signifies the present time

element.attribute = value: the structure is a short form of writing the value of the attribute of an element

BEGIN

step 1: \( p_w \leftarrow \sigma_{\text{category} = \text{"context"} \land \text{dpw} = \text{"w"}}(P) \) // selects the DPW context page

step 2: From \( p_w \), form a set of forward edges of kind \( \text{conDoc} \)

\[ L^+_t = \{ e^s | e^s.\text{source} = \text{pid} \land e^s.\text{kind} = \text{conDoc} \land e^s.\text{dir} = + \land e^s.t_+ \leq t \land (e^s.t_- > t \lor e^s.t_- = \text{null}) \} \]

step 2.2: if \( L^+_t \neq \text{null} \) then

sort edges in \( L^+_t \) in ascending order of \( e^s.t_+ \)
endif

step 3: return(\( p^t_w \))

END

The algorithm \( \text{GetDpwContext()} \) returns the DPW context page \( p_w \), with a set of of edges to the pages of category document that were plugged but not unplugged from the context page \( p_w \) before time \( t \). Unlike \( \text{GetPart()} \) or \( \text{GetDocument()} \) algorithms, discussed in Chapter 4, there is no attempt to concatenate the contents of all documents defining the state of \( p_w \) at time \( t \). It can be done logically, but normally, in practice, a such concatenated DPW context page is likely to be huge. Therefore, we assume that on selection of an edge from \( p^t_w \) thus returned, the implementation software will supply the pageId of the target document of the edge selected and \( t \) to the \( \text{GetDocument()} \) algorithm and thus the document can be retrieved.
5.2 Production of Case Context

We recall from the definition of Case Context (Chapter 2, definition 2.8), that the context of a case is the set of documents specific to the case, plugged at different times by different reviewers of the case under a DPW and also the DPW context page for the DPW. Since a case is a MPMSD, for every part of a case, the state of the context may be different. In the present section we discuss the retrieval of a state of the Case Context for a case at a particular time. Normally, at the time of creation of a particular part of the case, a case context is automatically produced by the arbiter and it is registered as a separate page with a pageId. Initially the case context is plugged to the DPW context of the DPW of the case with conCon kind of edges. The case context, in turn, is plugged to the corresponding case. Every case will have one and only one case context page, and every case context will have one and only one case. At the beginning of the review process of a case under a DPW, the case context will contain only one edge to the DPW context. As soon as a reviewer finds a relevant document specific to the case under examination, he/she will plug the document to the case context. Thus a case context page will have one forward edge of kind conCon to the DPW context page and zero or more forward edges of kind conDoc to relevant documents.

The algorithm to find the state of a case context, at a particular time, say GetCaseContext(), is similar to the GetDpwContext(), discussed in the previous section. The only difference is that, it will call the GetDpwContext() algorithm once inside GetCaseContext() to get the set of relevant documents available in the DPW Context at time t and then find the set of documents available in the Case Context at time t. Union of these two sets of documents defines the state of the context of a case at time t. Therefore the GetCaseContext() algorithm is not given explicitly.
5.3 Discussion

In this Chapter maintenance and productions of both DPW Context as well as Case Context have been discussed. The algorithms for production of these two types of contexts have been presented. In Chapter 3, we mentioned that the category graph is a mandatory dimension graph for the DPW problem. Without the category graphs it is not possible to handle the issues relating to DPW contexts and Case Contexts. Thus the Page Cube model and the algorithms discussed in this Chapter together provide solution of organizational memory hitherto not addressed in office information systems. Here, the page generated to represent a state of an actual page, stored in a page cube, at a particular time is not a persistent page. Such a page is generated on the fly during case examination and hence it need not be registered in the Page Cube.