REAPING THE BENEFITS OF ORACLE8I AND ORACLE9IAS

EOUG2002: 13.05.2002

Michael Garfield Sørensen
Forlaget Thomson / CeDeT, Denmark

Summary

This paper presents the solution for an on-line system based on Oracle8i and Oracle9iAS reaping the benefits of combining some of the newest technologies with good old-fashioned relational database lookups. Technologies utilized include:

- PL/SQL Web Toolkit for fast and flexible database access
- Meta data replication to give *current-awareness*, and automated document transfers for daily updates
- Relational database tables to hold the structure of XML documents combined with CLOBs to allow fast *on-the-fly XML-chunking and -caching*
- Function-based indexes to create *on-the-fly TOC's of XML documents*
- JavaServlet with XSLT to transform XML to HTML
- PL/SQL Gateway Cache to speedup re-occurring (big) requests
- interMedia Text to give fast free-text searches with "AND" and "NEAR" operators in combination with wildcards

Introduction

Two years ago Forlaget Thomson decided to invest in an on-line system with the two biggest brands on the Danish legal information market :

- Karnov: Primary law (statues) with commentaries
- UfR: Case reports

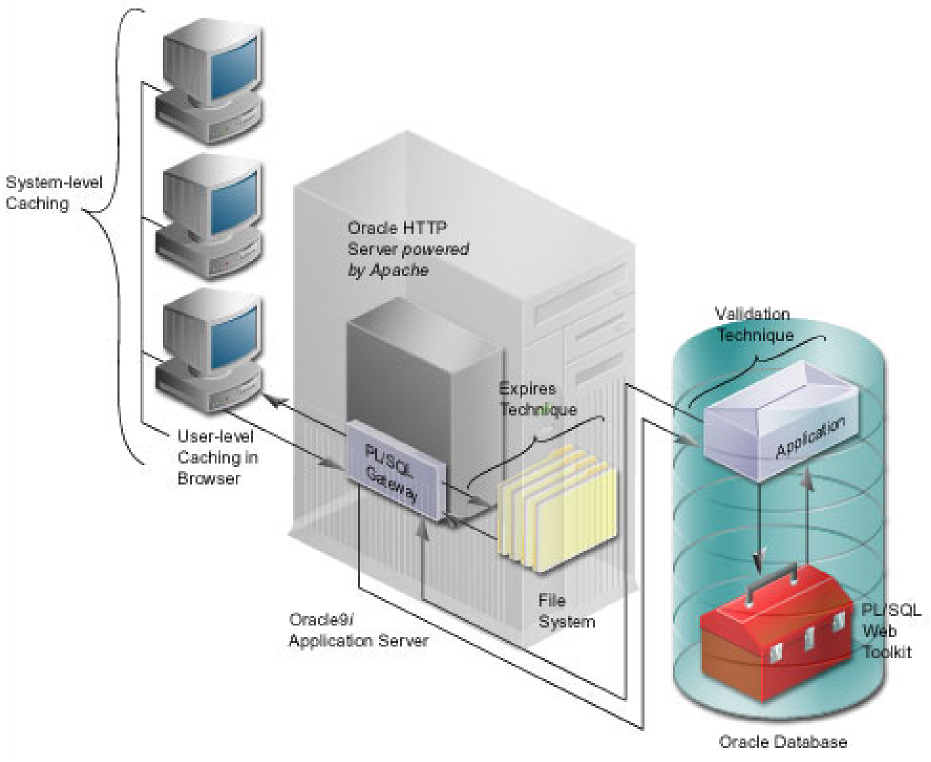
- Why an on-line system?
    Well, we all know that it's "e-business or no business".
- Why now?
    To stay ahead of (or maybe even trash) the competition!
- Why choose Oracle as the platform?
    Forlaget Thomson's production system is already based on Oracle, so it is a platform they are familiar with in terms of daily production and IT development. A prototype based on the existing production database, structure and content, was easily established - giving hope to the assumption that it wouldn't necessarily require a lot of additional internal resources to feed an on-line system.

The on-line system is a web application/interface on top of (an almost[1] exact replica of) the production database. The system has daily updates, more content, better (or more) functionality and is less troublesome to use than the existing print and CD-ROM products.

It's not 24x7 - and it's not open to the world (you have to pay to use it).

[1] The reason for it being only an **almost** exact replica will be explained later!

Environment



From **"Oracle 9i Application Server Using the PL/SQL Gateway"** (OTN, 2000-2001)

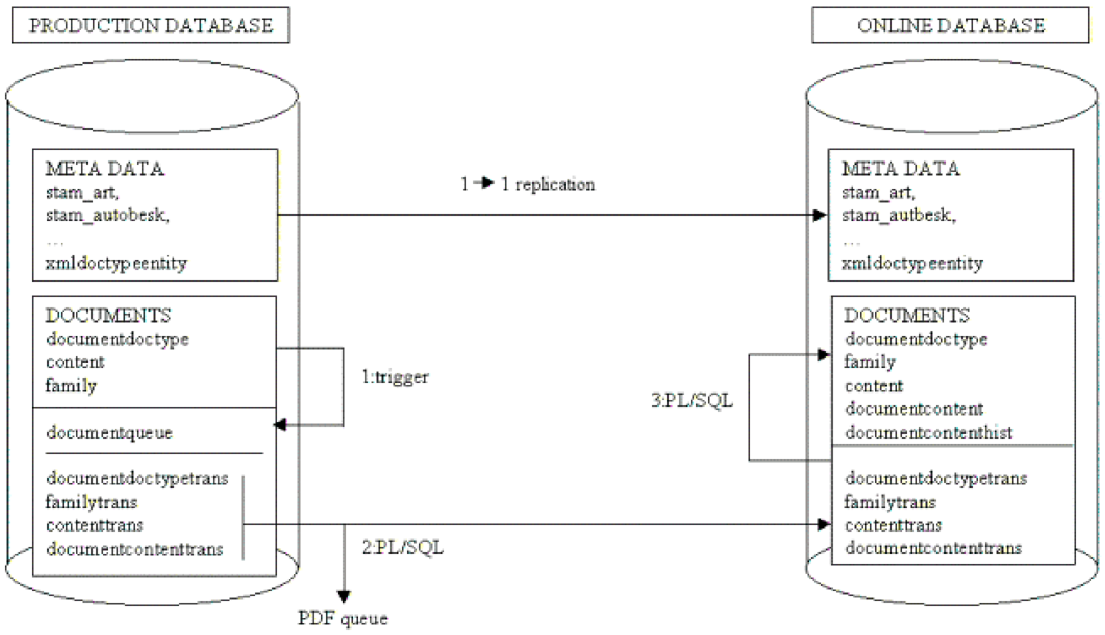
The on-line platform is a Sun Solaris with Oracle8i (8.1.7.2.0) and Oracle9iAS (1.0.2.0, mod_plsql 3.0.0.8.3). The majority of the code is written in PL/SQL as stored procedures utilizing the PL/SQL Web Toolkit and interMedia Text. A single JavaServlet utilizing the Oracle XML Development Kit is responsible for transforming XML to HTML with CSS using XSLT.

Simple example PL/SQL code utilizing PL/SQL Web Toolkit:

```
PROCEDURE produktfunktioner(vid stam_productversion.versid%TYPE:=NULL) IS
  reg VARCHAR2(20):='';
BEGIN
  IF NVL(vid,'*')=ilseutil.vidKarnov THEN
    reg:='?reg=Syst.reg.';
  ELSIF NVL(vid,'*')=ilseutil.vidUfR THEN
    reg:='?reg=Kron.reg.';
  END IF;
  ilsehtml.helsidetop('Funktioner');
  htp.p('Der kan v&aelig;lges mellem f&oslash;lgende funktioner:<br><br>');
  htp.p('<a href="ilseprod.prodframe?frame1url=ilsehtml.topline'||reg||
    '&frame2url=ilselog.faq" target="thomsondk_main"><b>FAQ</b></a><br>');
  htp.p('Ofte spurgte sp&oslash;rgsm&aring;l og svar!<br><br>');
  ...
  ilsehtml.helsidebund;
END produktfunktioner;
```

⇒ ilsehtml is a package containing common procedures for producing often used HTML!

Replication and automated transfers



Overview

To give *current-awareness* document meta data is replicated on an hourly basis using simple, standard single Master to read-only Slave replication. We're using (almost) the same database structure for production and on-line. Most constraints in the on-line database have been changed to default `deferred` and a few to `on delete cascade` in order to make replication work and to clean-up unwanted data. Before the documents are transferred to on-line (temporary storage) they undergo a minor transformation:

- Unicode/character entities are changed to something that the XML parser and the XSLT processor can handle gracefully (and is immediately usable in HTML):

  before:`&#10003;`
  after: `<img src="check.gif" alt="" border="0" />`

- For performance reasons (hyper-)link validation is done beforehand:

  before:`<LR IDREF="LBKG2000123.§11" />`
  after: `<LR IDREF="LBKG2000123.§11" LINKSTAT="INAKTIV" />`

- Content is put together in one CLOB for efficiency (see **XML-chunking and -caching**).

Warning: Use temporary CLOBs with caution in Oracle 8.1.5 and 8.1.6 (***bug 122619***)

In order for this transformation not to interfere with ungoing document processing, documents are only transferred once a day (currently approx. 20-30 new or modified per day). All replication and transfers are automated using Oracle's job queue (`DBMS_JOB`). A database table trigger is responsible for queuing modified or new documents for on-line transfer. Indexing (keeping the interMedia Text indexes up-to-date) is a different story, see **interMedia Text**.

XML-chunking and -caching

Many of the documents in the database are of a considerable size. To transform them on the server-side to HTML on-the-fly using XSLT may not be an issue. But sending HTML documents which are more than a couple of 100KB to the user's browser is an issue. So we came up with a fairly generic (in our context) algorithm to do XML-chunking; i.e. splitting an XML document in reasonable (not only in terms of size, but also in terms of presentation) parts of a size not much larger than some fixed preferred (maximum) size. Even if "the chunking" is very fast, it's still faster to reuse previously created chunks, thus introducing "caching of chunks". The ability to do it on-the-fly as part of the on-line application avoids the need to have one or more additional processes to prepare documents for on-line.

The database structure for documents consists of five tables, `DOCUMENTDOCTYPE` containing header information, `FAMILY` containing a representation of the structure of the XML, `CONTENT` containing the content of the XML document divided into multiple rows, `DOCUMENTCONTENT` containing the whole XML document as one `CLOB`, and `DOCUMENTCHUNK` containing the chunks[2]. The `DOCUMENTDOCTYPE`, `FAMILY` and `CONTENT` tables have been maintained (but not put to full use; until now) by the existing document-storage system at Forlaget Thomson for many years (pre-XML even, using it to store valid, normalized SGML). If this had not been the case, the XML-chunking solution would not have been possible! Definitions of these three tables are given below with sufficient explanation of the columns to understand the chunking algorithm.

```
DOCUMENTDOCTYPE
  NORMID NOT NULL VARCHAR2(255) -- primary key, unique document id
  TOPID  NOT NULL NUMBER(11)    -- unique id (used in FAMILY and CONTENT)
  ...

FAMILY
  TOPID      NOT NULL NUMBER(11) -- (topid,id) is the primary key, uniquely
  ID         NOT NULL NUMBER(11)    identifying a node (tag,text) in the XML
  MOTHER              NUMBER(11) -- id of parent element (within same topid)
  FIRSTCHILD          NUMBER(11) -- id of first child element (-"-)
  NEXTSISTER          NUMBER(11) -- id of parents next child (-"-)
  BYTESTART           NUMBER(11) -- byte offsets in the document of the first
  BYTEEND             NUMBER(11)    and last characters of the node itself
  ...

CONTENT
 TOPID   NOT NULL NUMBER(11) -- (topid,id,contno) is the primary key, each
 ID      NOT NULL NUMBER(11)    node in FAMILY has one or more rows here
 CONTNO  NOT NULL NUMBER(3)     with the actual content
 CONTENT VARCHAR2(2000)      -- content split at maximum 2000 characters
 ...
```

Let's start looking at the XML-chunking and -caching backwards. When we have the XML-chunk that holds the data we want to deliver on-line, we store it in a table with a unique id and call (via a HTTP-redirect) the JavaServlet responsible for transforming it to HTML (see **XML to HTML transformation using XSLT**):

[2] The tables `DOCUMENTCONTENT` and `DOCUMENTCHUNK` are only in the on-line database.

```
DefaultMaxSize CONSTANT NUMBER:=43008; -- 42K
JavaXMLServlet CONSTANT VARCHAR2(16):='/servlet/IlseXML';

PROCEDURE docxml(cid documentchunk.chunkid%TYPE) IS
BEGIN
  owa_util.redirect_url(
    'http://'||owa_util.get_cgi_env('HTTP_HOST')||JavaXMLServlet||
    '?cid='||TO_CHAR(cid),TRUE);
END;

...
/* at this point c_lob contains the XML-chunk we want */
SELECT seq_chunkid.NEXTVAL INTO cid FROM DUAL; -- get unique chunk id
BEGIN -- insert chunk into table
  INSERT INTO documentchunk(
   chunkid,normid,dato,chunkno,chunk,maxsiz,bytestart,byteend)
  VALUES(
   cid,dt_rec.normid,SYSDATE,chunk_count,c_lob,msz,byte_start,byte_end);
  COMMIT;
EXCEPTION
  WHEN DUP_VAL_ON_INDEX THEN
    ...
END;
docxml(cid); -- deliver
```

Let's know assume we request a chunk (requesting a document itself results in requesting the first chunk) of a document. First thing we do is check if the chunk has already been created (by a previous request):

```
...
BEGIN -- check IF chunk already exists
  SELECT dc.chunkid
   INTO  cid
   FROM  documentchunk dc
   WHERE dc.normid=nid   -- unique document identifier
   AND   dc.maxsiz=msz   -- preferred chunk (maximum) size
   AND   dc.chunkno=cno; -- requested chunk number
  docxml(cid); -- the chunk already exists so just deliver it
EXCEPTION
  WHEN NO_DATA_FOUND THEN -- calculate AND create chunk
    ...
```

Now let's look at the main steps of creating a chunk. The new chunk is build (in PL/SQL) as a temporary `CLOB` (referred to as `tmplob` in the following). `tmplob` is initialized with a `DOCUMENT` root tag with attributes containing information needed in the XSLT transformation. We now call a procedure, `docsplit` (described in detail below), that figures out the actual splitting by returning information about where a chunk ends if the chunking was started at a certain offset in the document. `docsplit` is called successively (in a loop) until the requested chunk is reached; always starting with the first chunk from the start of the document. If the chunk requested is not the first (or last) chunk of the document then additional `CHUNKREF` tags containing hyperlinks to next (or previous) chunk is added to `tmplob`. This is what - in the end - makes it possible for the user to navigate between chunks.

If the chunk is not the first in the document, then just adding the chunk to `tmplob` would (most likely) result in XML that is *not* well-formed. The chunk may contain end-tags for tags that appear in the document before the position where the chunk is started. To solve this issue we loop through parent elements (`mother` column in the `FAMILY` table) of the first element in the chunk - bottom-up. And in reversed order, top-down - appends the corresponding tags.

An empty `CHUNKSTART` tag is added to `tmplob` before the actual content from the document is appended (using `DBMS_LOB.COPY` and the values returned by `docsplit`) to tmplob. And an empty `CHUNKSLUT` tag is added after. Doing this with `CLOB`s gives very good performance.

If the chunk is not the last in the document, then just like the before more is needed. This time the end-tags that appear outside the chunk, but actually belongs to start-tags within the chunk needs to be appended (with XML it has got to be well-formed). Again this is achieved by looping through parents, but this time appending the corresponding end-tags (rather than the tags for the parent elements themselves). The `tmplob` is finished of with a `DOCUMENT` end-tag, stored in the database and delivered as HTML by the JavaServlet. The overall structure of a chunk is:

```
<DOCUMENT NORMID="..."><!-- root element -->
 <CHUNKREFS>
  <CHUNKREF HREF="...?{chunkno-1}" PN="P"/><!--hyperlink to previous chunk-->
  <CHUNKREF HREF="...?{chunkno+1}" PN="N"/><!--hyperlink to next chunk-->
 </CHUNKREFS>
 <!-- appended start-tags to ensure well-formedness -->
 <LOV>
  <N1>
   <CHUNKSTART/>
   <!-- here starts the data from the document -->
   <N2>
    <P>
     ...
    </P>
   <!-- here ends the data from the document -->
   <CHUNKSLUT/>
   <!-- appended end-tags to ensure well-formedness -->
   </N2>
  </N1>
 </LOV>
</DOCUMENT>
```

The figures in the appendix shows a real example. Now let's turn to the actual splitting algorithm - `docsplit`. Given an entry point; an element in `FAMILY` identified by `tid`, `start_id`, calculate the size of the element with contents (if any) by subtracting it's byte offset (`bytestart`) from the byte offset of it's parent's next child (`nextsister` column in `FAMILY`):

```
SELECT fns.bytestart-f.bytestart
 INTO  size_of_element_with_content
 FROM  family fns,family f
 WHERE f.topid=tid
 AND   f.id=startid
 AND   fns.topid=f.topid
 AND   fns.id=f.nextsister;
```

If the actual element is the last child (has no `nextsister`) then we can check it's parent's (`mother`'s) `nextsister`. If at some point there are no more possible `nextsisters` to examine we have reached the end of the (sub-)tree!

If the element we're looking at is not suitable for chunking then we are done (even if the chunk at this point is a bit too large). Elements not suited for chunking include tables, graphics, lists, headings, and paragraphs because it would split the documents in awkward places not fit for presentation. Also if the element has no children (being text or an empty tag) it is obviously not suited for further chunking.

If the current element is suited for chunking and has a size that is greater than the preferred default (maximum) chunk size, then calculate the size you would get if you looked at it's first child (`firstchild`). This is done by calling `docsplit` recursively.

Actually that's the main logic of the splitting algorithm - calculating sizes of elements and sub-elements recursively. It sounds simple but the actual code doesn't look that simple. There's quite a bit of house-keeping of how far have we gotten (how many levels have we transcended into sub-structures of the document) and what to do if the tags following the splitting point are all end-tags - no good starting of a new (the next) chunk with end-tags.

All selects to do the calculations are using the primary keys on `FAMILY` and `CONTENT` and the algorithm is therefore very fast - based on the strengths of relational database lookups.

Function-based indexing

To serve table of content for documents on-the-fly we created a function-based index on the `CONTENT` table. The function returns a number for the `N1`, `N2`, ... `N7` start-tags and `NULL` for all others. `N1`, `N2`, ..., `N7` are the elements used for making logical sections and sub-sections in the documents. Thus the index only contains entries pointing at the start of new sections. Which make it very fast to receive exactly those to be used in a TOC:

```
CREATE OR REPLACE FUNCTION pr_niveaunr(
  sgmlstr VARCHAR2)
  RETURN NUMBER DETERMINISTIC
IS
BEGIN
  IF NVL(LENGTH(sgmlstr),0)<4 THEN
    RETURN NULL;
  ELSE
    IF SUBSTR(sgmlstr,1,3) IN ('<N1','<N2','<N3','<N4','<N5','<N6','<N7')
    AND SUBSTR(sgmlstr,4,1) IN ('>',' ') THEN
      RETURN TO_NUMBER(SUBSTR(sgmlstr,3,1));
    ELSE
      RETURN NULL;
    END IF;
  END IF;
END;
/

-- topid uniquely defines the document
CREATE INDEX content_inx_niveaunr ON content(topid,pr_niveaunr(content));

-- remember to analyze or the index will not work
ANALYZE TABLE content COMPUTE STATISTICS;

-- remember special privileges and initSID.ora settings
-- example of using the index
SELECT c.* FROM  content c WHERE c.topid=&&tid AND pr_niveaunr(c.content)=1;
```

XML to HTML conversion using XSLT

We started out wanting to use Oracle's XML Development Kit (XDK) for PL/SQL preferring to keeping all the code in PL/SQL. We soon discovered that it is not performing very well! So we had to do it outside the database with a JavaServlet which in our case performs very well (up-to 16 times faster than with PL/SQL). The main part of the Java code looks like this:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {
    Clob   c_lob = null;
    String selst;

    try {
        conn.setAutoCommit (false);

        String cid = req.getParameter("cid");

        // Create a Statement and Read XML into Clob
        Statement stmt = conn.createStatement ();
        selst = "select chunk from documentchunk where chunkid='" + cid + "'";
        OracleResultSet rset = (OracleResultSet)stmt.executeQuery(selst);
        while (rset.next ()) {
          c_lob = ((OracleResultSet)rset).getClob (1);
        }
        rset.close();
        stmt.close();

        // Read and Parse XSLT (from filesys)
        DOMParser parser;
        URL xslURL;
        parser = new DOMParser();
        parser.setBaseURL(null);
        parser.setPreserveWhitespace (true);
        parser.setValidationMode(false);
        parser.setBaseURL(null);
        xslURL = createURL("http:///onlxsl/thomson.xsl");
        parser.parse(xslURL);
        XMLDocument xslDoc = parser.getDocument();

        // Read and Parse XML (in Clob)
        parser.setPreserveWhitespace (true);
        parser.setValidationMode(false);
        parser.parse(c_lob.getCharacterStream() );
        XMLDocument xmlDoc = parser.getDocument();

        XSLProcessor processor = new XSLProcessor();
        processor.showWarnings(true);
        processor.setErrorStream(System.err);
        XSLStylesheet stylesheet = new XSLStylesheet( xslDoc, xslURL );
        res.setContentType("text/html");
        PrintWriter out = new PrintWriter (res.getOutputStream());
        processor.processXSL(stylesheet, xmlDoc, out);
        out.close();

    } catch ( SQLException e ) {
       System.err.println("A database error occurred." + e.getMessage());
    }

  } // doGet
```

A good starting point when using XSLT to transform XML into HTML with Cascading Style Sheets (CSS) is to map element names in the XML to CSS classes of the same name. This is demonstrated in the sample XSLT and CSS given below for the `TITEL` element (and `.titel` class). The sample XSLT also demonstrate how the result of link validation (the `LINKSTAT` attribute on reference elements) is used to create different types of (or no) HTML hyperlinks:

Sample XSLT:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="LOV">
       <A NAME="{@ID}"></A>
       <TABLE BORDER="0" CELLSPACING="0" CELLPADDING="2">
              <xsl:apply-templates />
       </TABLE>
</xsl:template>

<!-- title elements -->
<xsl:template match="TITEL">
       <TR>
              <TD COLSPAN="3">
                     <SPAN CLASS="titel">
                            <xsl:apply-templates/>
                            <BR/><BR/>
                     </SPAN>
              </TD>
       </TR>
</xsl:template>

<xsl:template match="N0">
              <BR/>
              <xsl:apply-templates />
</xsl:template>
...

<!-- **** References  **** -->
<!-- references to other "documents" with explicit inline link text -->
<xsl:template match="LR|UR|URB">
   <xsl:choose>
       <xsl:when test="@LINKSTAT = 'INTERN'">
              <A HREF="#{@IDREF}">
                     <SPAN CLASS="dokref"><xsl:apply-templates/></SPAN>
              </A>
       </xsl:when>
       <xsl:when test="@LINKSTAT = 'EKSTERN'">
              <A TARGET="thomsondk_main" HREF="{$docbyref_prc}?idr={@IDREF}">
                     <SPAN CLASS="dokref"><xsl:apply-templates/></SPAN>
              </A>
       </xsl:when>
       <xsl:when test="@LINKSTAT = 'INAKTIV'">
              <xsl:apply-templates/>
       </xsl:when>
   </xsl:choose>
</xsl:template>
...
```

⇒ XSLT is also used to do XML to PDF transformation via XSL:FO(P). With XML you can easily have multiple output formats (HTML, PDF, Folio Flat-File)

Sample CSS:

```
BODY
{
    COLOR: #333333;
    FONT-FAMILY: Verdana, Georgia, 'Trebuchet MS', 'Times New Roman';
    BACKGROUND-COLOR: #eeeeee
}
A
{
    COLOR: #003399;
    FONT-FAMILY: Verdana, Georgia, 'Trebuchet MS', 'Times New Roman';
    TEXT-DECORATION: underline
}
A:hover
{
    COLOR: #003399
}
.titel
{
    FONT-WEIGHT: bold;
    FONT-SIZE: 15pt;
    COLOR: #333333;
    LINE-HEIGHT: 15pt;
    FONT-FAMILY: Verdana, Georgia, 'Trebuchet MS', 'Times New Roman'
}
...
```

PL/SQL Gateway Cache

Starting with Oracle9iAS there is a caching mechanism available as part of the PL/SQL Web Toolkit through the `owa_cache` package. It allows you to cache created HTML pages in three ways: Using an expiration technique or using a user- or system-level validation technique. See figure on page 2. As explained in the Oracle9iAS Documentation the expiration technique is the fastest (since it fetches from the cache without invoking a stored procedure in the database) but is more difficult to control than the validation technique. We have used the validation technique on system-level for (amongst other things) caching the result of interMedia searches - see example below. We would also have liked to use user-level caching, but after having set `KeepAlive=off`[3] in the Apache configuration, user-level caching seems to have no effect (which makes sense but the documentation doesn't mention any scenarios that would cause user-level caching not to work).

Sample PL/SQL for system-level caching using the validation technique:

```
cacheLevel CONSTANT VARCHAR2(6):='SYSTEM';

PROCEDURE <procedure_name>(<parameter...>)
IS
  <variable...>
  etag VARCHAR2(255);
  elvl VARCHAR2(10);
BEGIN
<code that needs to be performed every time (logging for example)>
```

[3] The reason for this being that with `KeepAlive=On` the server occasionally took up to as much 20 minutes before it decided to respond to a request. We've experienced the same thing on the AIX platform but surprisingly not on the NT platform! And we still haven't figured out why!?

```
  etag:=owa_cache.get_etag;
  elvl:=owa_cache.get_level;
  IF etag IS NOT NULL AND elvl IS NOT NULL THEN -- been here before
    IF etag=TO_CHAR(SYSDATE,'YYYYMMDD') AND elvl=ilseprod.cacheLevel THEN
      owa_cache.set_not_modified; -- today nothing has changed
      RETURN; -- so we won't go get it again
    END IF;
  END IF;
  owa_cache.set_cache(
    p_etag=>TO_CHAR(SYSDATE,'YYYYMMDD'),
    p_level=>cacheLevel); -- cache new page (produced below)

  <code that produces the page>
END;
```

The above code works well for interMedia free-text searches, since we're only indexing new/modified documents once per day.

interMedia Text

You face quite a few challenges when you get involved in using interMedia Text for indexing large amounts of XML data. Here are the five major challenges that we've run into:

1. Setting up the correct indexes for the fastest possible free-text searches with wildcards. Deciding (after having understood) whether to do zone- or field-indexing (or combinations) and which configuration options are available, where to set them, and what effect they have. The DBA did most of this for us!

2. Writing the optimal SQL to utilize the indexes. We had a hard time getting the CBO to use the interMedia indexes if we used more than a couple of joins in the queries. So we ended up with doing simple selects only based on the interMedia indexes, and afterwards checking if other search criteria - requiring meta data lookups - was fulfilled:

```
  TYPE s_recType IS RECORD (
    srt INTEGER,
    rid ROWID,
    loc VARCHAR2(1),
    nid documentcontent.normid%TYPE);
  TYPE s_curType IS REF CURSOR RETURN s_recType;
  TYPE s_tabType IS TABLE OF s_recType INDEX BY BINARY_INTEGER;
  s_cur s_curType;
  s_rec s_recType;
  s_tab s_tabType;
  s_inx BINARY_INTEGER:=0;
BEGIN
  ...
  IF <LAWS and NOTES> THEN -- within LOV and NOTER elements
    OPEN s_cur FOR
      SELECT /*+ FIRST_ROWS */ SCORE(1),dc.ROWID,'L',NULL
       FROM  documentcontent dc
       WHERE CONTAINS(dc.content,'('||str||') WITHIN LOV',1)>0
      UNION ALL
      SELECT /*+ FIRST_ROWS */ SCORE(2),dc.ROWID,'N',NULL
       FROM  documentcontent dc
       WHERE CONTAINS(dc.content,'('||str||') WITHIN NOTER',2)>0
       ORDER BY 1 DESC;
  ELSIF <LAWS (and not NOTES)> THEN -- within LOV element
    OPEN s_cur FOR
      SELECT /*+ FIRST_ROWS */ SCORE(1),dc.ROWID,'L',NULL
       FROM  documentcontent dc
       WHERE CONTAINS(dc.content,'('||str||') WITHIN LOV',1)>0
```

```
     ELSIF <CASES (and not NOTES)> THEN
       OPEN s_cur FOR
         SELECT /*+ FIRST_ROWS */ SCORE(1),dc.ROWID,'D',NULL
          FROM  documentcontent dc
          WHERE CONTAINS(dc.content,'('||str||') WITHIN DOM',1)>0;
     ELSIF <CASES and NOTES> THEN -- within DOM element
       ...
     ELSIF <NOTES (and not CASES and not LAWS)> THEN
       ...
     END IF;
     ...
     LOOP
       FETCH s_cur INTO s_rec;
       EXIT WHEN s_cur%NOTFOUND;
       BEGIN
         IF s_rec.loc='D' THEN
           -- check other search criteria for cases - (D)OM element
         ELSIF s_rec.loc='L' THEN
           -- check other search criteria for laws - (L)OV element
         ELSIF s_rec.loc='N' THEN
           -- check other search criteria for notes - (N)OTER element
         END IF;
         -- other search criteria has been matched
         s_inx:=s_inx+1;
         s_tab(s_inx).srt:=s_rec.srt;
         s_tab(s_inx).rid:=s_rec.rid;
         s_tab(s_inx).loc:=s_rec.loc;
         s_tab(s_inx).nid:=s_rec.nid;
       EXCEPTION
         WHEN NO_DATA_FOUND THEN
           -- other search criteria not matched, ignore
           NULL;
       END;
       ...
       EXIT WHEN s_inx>maxNumberOfHits;
       ...
     END LOOP;
     CLOSE s_cur;
     ... -- loop through s_tab(1..s_inx) producing the HTML
   END;
```

3. We started out with using the `ctxsrv` server process which keeps the interMedia indexes up-to-date as fast as possible. But we soon saw the negative effect this process had on database and machine performance. So instead we decided to synchronize the index once a day during night-time using the `ctx_ddl.sync_index` procedure.

4. Getting the highlighting to work with our chunks.

5. And (of course) the wonderful joy of having to URL en-/decode search strings containing special interMedia operators.

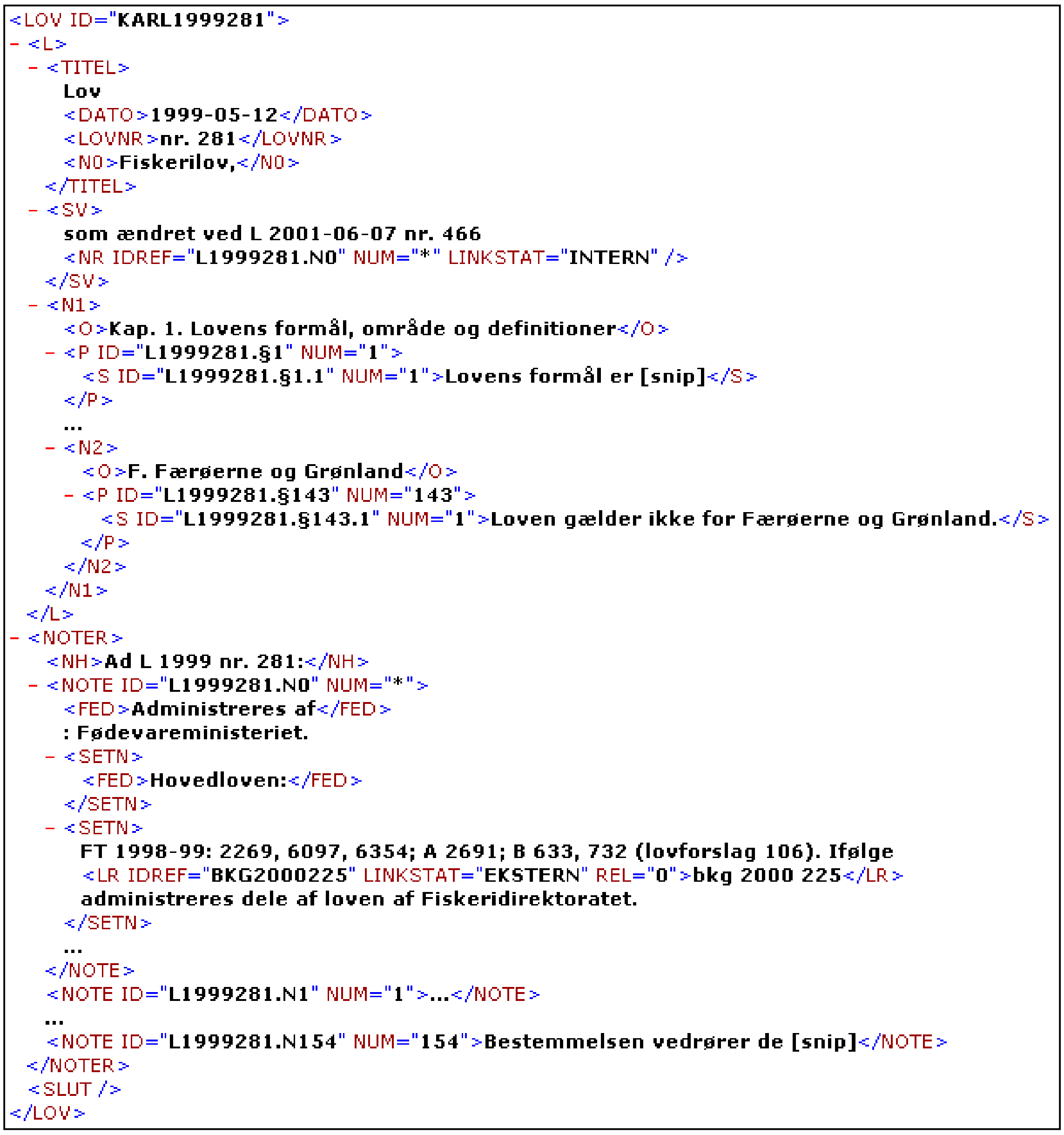
Conclusion

With Oracle8i and Oracle9iAS you *do* have a platform well-suited for developing on-line systems. There are some pitfalls to avoid, especially with the newest features. But by combining new features with traditional relational database features you *can* create powerful solutions:

Combine to reap the benefits of Oracle8i and Oracle9iAS!

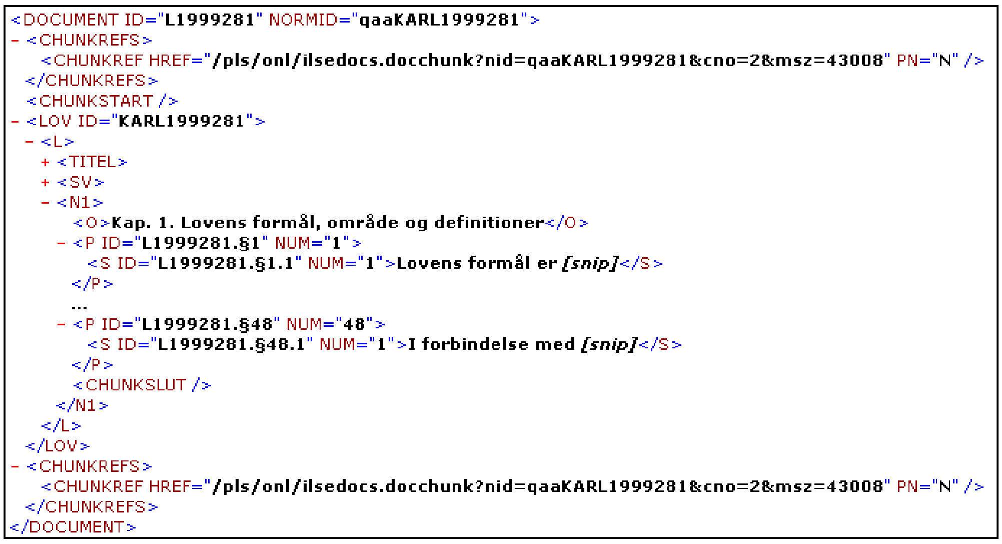
Appendix: Figures

Full document:

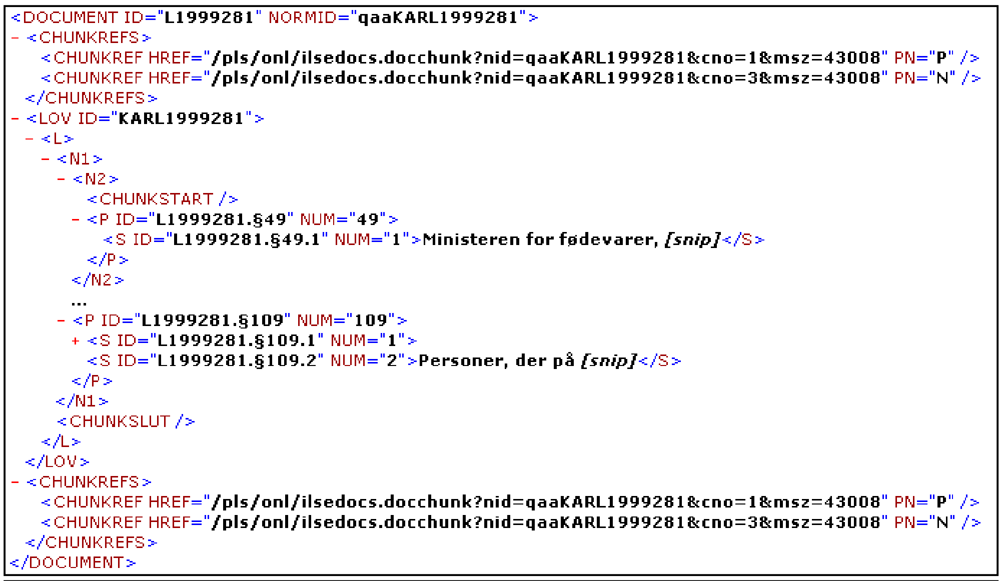


```
<LOV ID="KARL1999281">
- <L>
  - <TITEL>
      Lov
      <DATO>1999-05-12</DATO>
      <LOVNR>nr. 281</LOVNR>
      <NO>Fiskerilov,</NO>
    </TITEL>
  - <SV>
      som ændret ved L 2001-06-07 nr. 466
      <NR IDREF="L1999281.N0" NUM="*" LINKSTAT="INTERN" />
    </SV>
  - <N1>
      <O>Kap. 1. Lovens formål, område og definitioner</O>
    - <P ID="L1999281.§1" NUM="1">
        <S ID="L1999281.§1.1" NUM="1">Lovens formål er [snip]</S>
      </P>
      ...
    - <N2>
        <O>F. Færøerne og Grønland</O>
      - <P ID="L1999281.§143" NUM="143">
          <S ID="L1999281.§143.1" NUM="1">Loven gælder ikke for Færøerne og Grønland.</S>
        </P>
      </N2>
    </N1>
  </L>
- <NOTER>
    <NH>Ad L 1999 nr. 281:</NH>
  - <NOTE ID="L1999281.N0" NUM="*">
      <FED>Administreres af</FED>
      : Fødevareministeriet.
    - <SETN>
        <FED>Hovedloven:</FED>
      </SETN>
    - <SETN>
        FT 1998-99: 2269, 6097, 6354; A 2691; B 633, 732 (lovforslag 106). Ifølge
        <LR IDREF="BKG2000225" LINKSTAT="EKSTERN" REL="0">bkg 2000 225</LR>
        administreres dele af loven af Fiskeridirektoratet.
      </SETN>
      ...
    </NOTE>
    <NOTE ID="L1999281.N1" NUM="1">...</NOTE>
    ...
    <NOTE ID="L1999281.N154" NUM="154">Bestemmelsen vedrører de [snip]</NOTE>
  </NOTER>
  <SLUT />
</LOV>
```

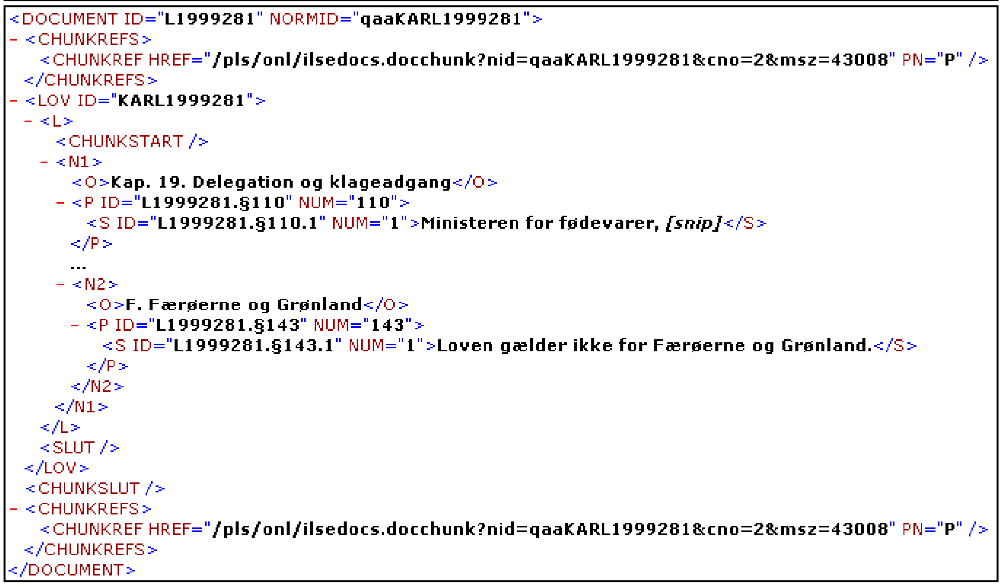
Chunks:



```
<DOCUMENT ID="L1999281" NORMID="qaaKARL1999281">
- <CHUNKREFS>
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=2&msz=43008" PN="N" />
  </CHUNKREFS>
  <CHUNKSTART />
- <LOV ID="KARL1999281">
  - <L>
    + <TITEL>
    + <SV>
    - <N1>
        <O>Kap. 1. Lovens formål, område og definitioner</O>
      - <P ID="L1999281.§1" NUM="1">
          <S ID="L1999281.§1.1" NUM="1">Lovens formål er [snip]</S>
        </P>
        ...
      - <P ID="L1999281.§48" NUM="48">
          <S ID="L1999281.§48.1" NUM="1">I forbindelse med [snip]</S>
        </P>
        <CHUNKSLUT />
      </N1>
    </L>
  </LOV>
- <CHUNKREFS>
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=2&msz=43008" PN="N" />
  </CHUNKREFS>
</DOCUMENT>
```

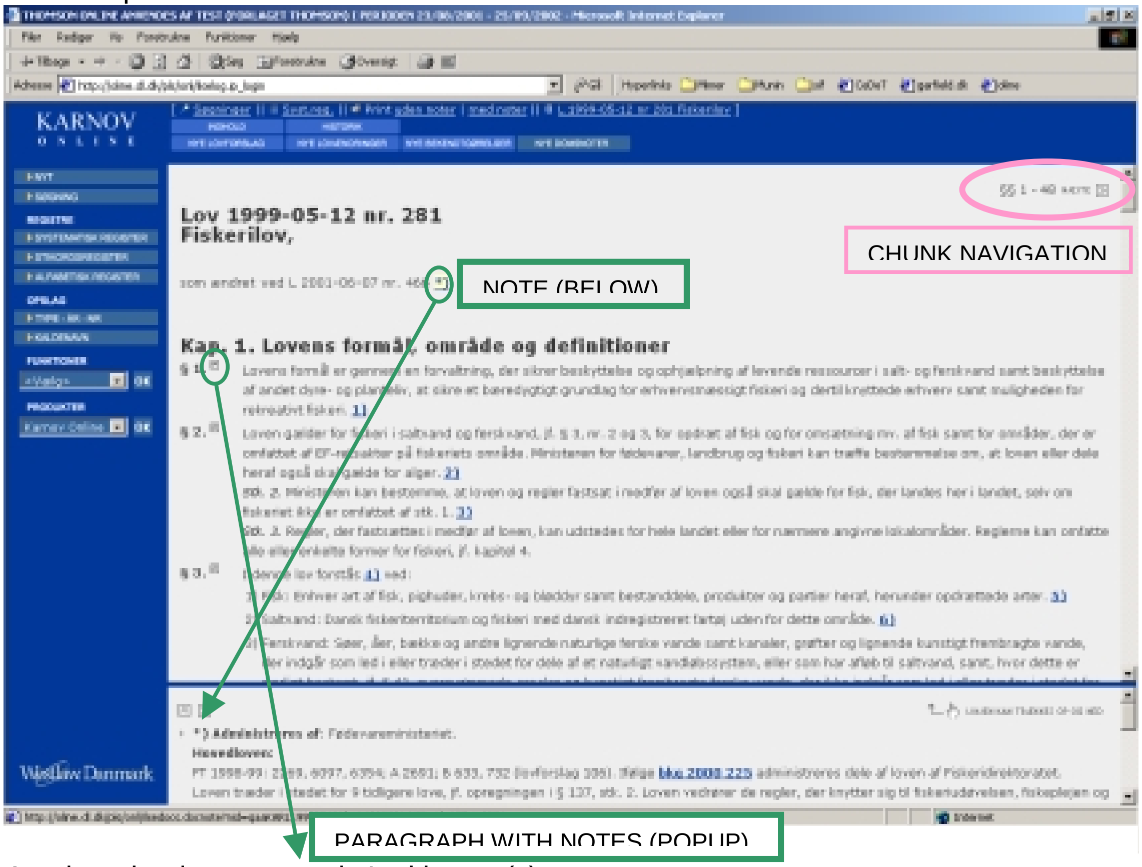


```
<DOCUMENT ID="L1999281" NORMID="qaaKARL1999281">
- <CHUNKREFS>
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=1&msz=43008" PN="P" />
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=3&msz=43008" PN="N" />
  </CHUNKREFS>
- <LOV ID="KARL1999281">
  - <L>
    - <N1>
      - <N2>
          <CHUNKSTART />
        - <P ID="L1999281.§49" NUM="49">
            <S ID="L1999281.§49.1" NUM="1">Ministeren for fødevarer, [snip]</S>
          </P>
        </N2>
        ...
      - <P ID="L1999281.§109" NUM="109">
        + <S ID="L1999281.§109.1" NUM="1">
          <S ID="L1999281.§109.2" NUM="2">Personer, der på [snip]</S>
        </P>
      </N1>
      <CHUNKSLUT />
    </L>
  </LOV>
- <CHUNKREFS>
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=1&msz=43008" PN="P" />
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=3&msz=43008" PN="N" />
  </CHUNKREFS>
</DOCUMENT>
```



```
<DOCUMENT ID="L1999281" NORMID="qaaKARL1999281">
- <CHUNKREFS>
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=2&msz=43008" PN="P" />
  </CHUNKREFS>
- <LOV ID="KARL1999281">
  - <L>
      <CHUNKSTART />
    - <N1>
        <O>Kap. 19. Delegation og klageadgang</O>
      - <P ID="L1999281.§110" NUM="110">
          <S ID="L1999281.§110.1" NUM="1">Ministeren for fødevarer, [snip]</S>
        </P>
        ...
      - <N2>
          <O>F. Færøerne og Grønland</O>
        - <P ID="L1999281.§143" NUM="143">
            <S ID="L1999281.§143.1" NUM="1">Loven gælder ikke for Færøerne og Grønland.</S>
          </P>
        </N2>
      </N1>
    </L>
    <SLUT />
  </LOV>
  <CHUNKSLUT />
- <CHUNKREFS>
    <CHUNKREF HREF="/pls/onl/ilsedocs.docchunk?nid=qaaKARL1999281&cno=2&msz=43008" PN="P" />
  </CHUNKREFS>
</DOCUMENT>
```

Chunk 1 presented to the user:



Lov 1999-05-12 nr. 281
Fiskerilov,

som ændret ved L 2001-06-07 nr. 466 *)

Kap. 1. Lovens formål, område og definitioner

§ 1. Lovens formål er gennem en forvaltning, der sikrer beskyttelse og ophjælpning af levende ressourcer i salt- og ferskvand samt beskyttelse af andet dyre- og planteliv, at sikre et bæredygtigt grundlag for erhvervsmæssigt fiskeri og dertil knyttede erhverv samt muligheden for rekreativt fiskeri. 1)

§ 2. Loven gælder for fiskeri i saltvand og ferskvand, jf. § 3, nr. 2 og 3, for opdræt af fisk og for omsætning mv. af fisk samt for områder, der er omfattet af EF-retsakter på fiskeriets område. Ministeren for fødevarer, landbrug og fiskeri kan træffe bestemmelse om, at loven eller dele heraf også skal gælde for alger. 2)

Stk. 2. Ministeren kan bestemme, at loven og regler fastsat i medfør af loven også skal gælde for fisk, der landes her i landet, selv om fiskeriet ikke er omfattet af stk. 1. 3)

Stk. 3. Regler, der fastsættes i medfør af loven, kan udstedes for hele landet eller for nærmere angivne lokalområder. Reglerne kan omfatte alle eller enkelte former for fiskeri, jf. kapitel 4.

§ 3. I denne lov forstås 4) ved:

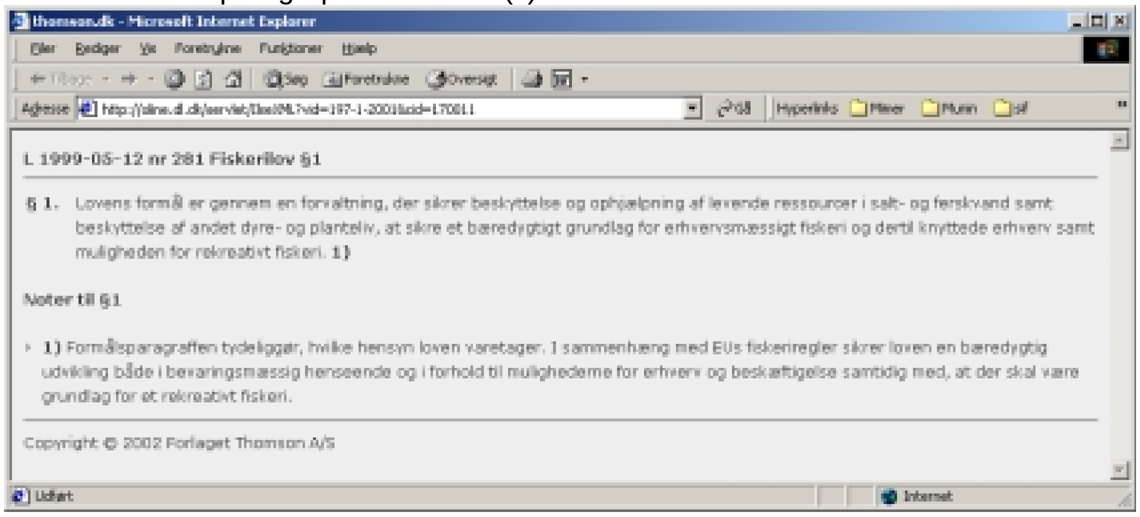
1) Fisk: Enhver art af fisk, pighuder, krebs- og bløddyr samt bestanddele, produkter og partier heraf, herunder opdrættede arter. 5)

2) Saltvand: Dansk fiskeriterritorium og fiskeri med dansk indregistreret fartøj uden for dette område. 6)

3) Ferskvand: Søer, åer, bække og andre lignende naturlige ferske vande samt kanaler, grøfter og lignende kunstigt frembragte vande, der indgår som led i eller træder i stedet for dele af et naturligt vandløbssystem, eller som har afløb til saltvand, samt, hvor dette er

*) **Administreres af:** Fødevareministeriet.
Hovedloven:
FT 1998-99: 2289, 6097, 6354; A 2681; B 633, 732 (lovforslag 106). Ifølge bkg.2000.225 administreres dele af loven af Fiskeridirektoratet.
Loven træder i stedet for 9 tidligere love, jf. opregningen i § 127, stk. 2. Loven vedrører de regler, der knytter sig til fiskeriudøvelsen, fiskeplejen og

Another chunk - paragraph 1 with note(s):



L 1999-05-12 nr 281 Fiskerilov §1

§ 1. Lovens formål er gennem en forvaltning, der sikrer beskyttelse og ophjælpning af levende ressourcer i salt- og ferskvand samt beskyttelse af andet dyre- og planteliv, at sikre et bæredygtigt grundlag for erhvervsmæssigt fiskeri og dertil knyttede erhverv samt muligheden for rekreativt fiskeri. 1)

Noter til §1

1) Formålsparagraffen tydeliggør, hvilke hensyn loven varetager. I sammenhæng med EUs fiskeriregler sikrer loven en bæredygtig udvikling både i bevaringsmæssig henseende og i forhold til mulighederne for erhverv og beskæftigelse samtidig med, at der skal være grundlag for et rekreativt fiskeri.

Copyright © 2002 Forlaget Thomson A/S