Content in allen Sprachen - bei Mehrsprachigkeit und Versionierung den Überblick behalten

Martin Friemel
Enterprise Web AG – www.webag.com

Schlüsselworte

Intranet, PL/SQL Softwareentwicklung, UTF8, Multilinguale Datenhaltung, Versionierung, Historisierung, Content Management, WebAG Automat, www.webag.com

Zusammenfassung

Dieser Vortrag beschreibt die Erstellung und Verwaltung mehrsprachiger, versionierter Inhalte. Der Vortrag beantwortet folgende Fragen:

- Wie muss eine Web-Benutzeroberfläche so gestaltet werden, dass Autoren den Überblick über die verschiedensprachigen Varianten des gleichen Inhalts behalten können?
- Wie werden mehrsprachige Inhalte in der Oracle-Datenbank gespeichert?
- Wie werden solche Dokumente versioniert, damit die Autoren auf vergangene Bearbeitungsstände zurückgreifen können?
- Wie können Oracle-Standardunktionen bei der Entwicklung solcher System helfen?

Die Umsetzung dieser Aufgaben wird an kurzen Beispielen online am Beispiel des Web-CMS WebAG Automat veranschaulicht. Gleichzeitig werden Beispiele des physischen Datenmodells und exemplarischer PL/SQL-Sourcecode besprochen.

Inhalt

Einleitung

Dieser Vortrag beschreibt die Konzepte, das Vorgehen und die eingesetzte Technik bei der Erweiterung einer vorhandenen Contentmanagement-Software für die Speicherung mehrsprachiger Inhalte und Versionierung Dokumente. Die Randbedingungen sind:

- Es sollen fremde Zeichensätze wie japanisch oder russisch unterstützt werden
- Alle Änderungen an Dokumenten sollen zurückverfolgt werden können
- Die Autoren des WebCMS sollen jederzeit den Überblick über die Versionen ihrer Inhalte und auch über mehrsprachige Varianten behalten können.
- Die Websites, die mit diesem System betrieben werden, sollen eine komfortable Sprachumschaltung erhalten.
- Das System wird weiterhin komplett in der Sprache PL/SQL entwickelt.
- Die bestehende Software und das Datenmodell sollen trotz der neuen die Versionierung und der Mehrsprachigkeit weitgehend unverändert bleiben

Die Projekt-Analyse ergab, dass beide Features „Mehrsprachigkeit“ und „Versionierung“ am einfachsten voneinander getrennt in das CMS eingefügt werden sollten. Daraus ergibt sich grob folgende Struktur:

- Jedes Dokument im WebCMS erhält seine eigene Historienverwaltung, um alle Änderungen an diesem Dokuement zu protokolieren und nachzuverfolgen.
- Unterschiedliche Dokumente können zueinander als Sprachvarianten fungieren. Auf den Webseiten wird den Lesern von jeder Sprachvariante die veröffentlichte Version angeboten.

Mehrsprachigkeit

Fremde Zeichensätze – UTF8

Wenn über Inhalte in verschiedenen Sprachen nachgedacht wird, ergibt sich sofort die Anforderung, nicht nur westeuropäische Zeichen zu verarbeiten, sondern beliebige Zeichen wie z.B. japanische oder russische Zeichen.

Es gibt eine Zeichensatzfamilie, die diese Anforderung abdeckt, nämlich UTF8. Was also ist mit der bestehenden Anwendung zu tun, um UTF8 zu nutzen?

Datenbank

Die Datenbank wird neu erzeugt. Als Zeichnsatz wird AL32UTF8 angegeben. Das ist die Oracle-Empfehlung für UTF8-Datenbanken. Dieser Zeichensatz wird als primärer Zeichensatz angegeben, also nicht als alternativer Zeichensatz für NVARCHAR2-Spalten. Grund: Wir wollen das Datenmodell nicht ändern. In jeder bestehenden VARCHAR2-Spalte sollen fortan fremde Zeichen gespeichert werden können. Wichtig ist dabei, dass die Spalten mit der Einheit „CHAR“ statt „BYTE“ angelegt sind, also z.B. `VARCHAR2(100 CHAR)`, denn mit UTF8 benötigen viele Zeichen mehr als ein Byte. Notfalls muss das per ALTER-

TABLE-Befehl nachgeholt werden. Danach werden die Altdaten in die neue Datenbank importiert.

Oracle Webserver – modPLSQL

Die modPLSQL-Definition für den Zugriff auf die Datenbank (DAD – Database Access Descriptor) in der Datei `$ORACLE_HOME/Apache/modplsql/cfg/wdbsvr.app` muss angepasst werden, damit die Web-Formulare, die mit PL/SQL entwickelt wurden, beliebige Zeichen unterstützen. Dazu muss lediglich der Parameter `nls_lang` gesetzt werden, z.B.:

```
nls_lang = al32utf8.
```

PL/SQL-Anwendung

Hier ist wenig zu tun. Vorsicht ist lediglich geboten bei der Länge der deklarierten Variablen, denn auch hier kann es passieren, dass Zeichenketten mehr Bytes benötigen als die Anzahl der Zeichen im String.

Beispiel: Wenn Sie eine Routine programmiert haben, die eine Zeichenkette Zeichen für Zeichen abarbeitet und jedes Zeichen in eine Variable vom Typ `CHAR(1)` ablegt, scheitern Sie in einer UTF8-Datenbank bereits beim Speichern eines deutschen Umlauts, denn ein Umlaut belegt im UTF8-Zeichensatz zwei Bytes. Besser als `CHAR(1)` ist daher z.B. `VARCHAR2(1 CHAR)`.

Man kann es sich allerdings auch einfach machen und den `init.ora`-Parameter

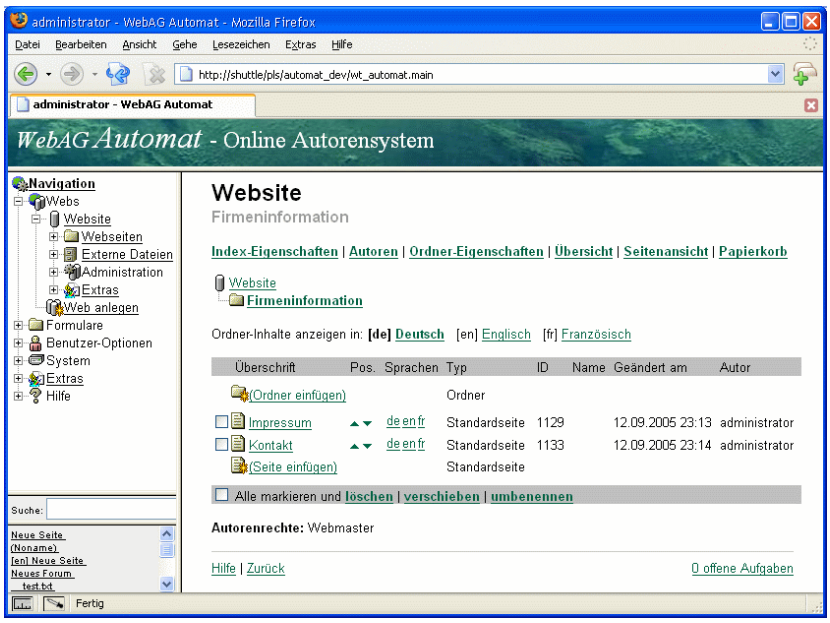

Abb. 1: Mehrsprachigkeit im Autorensystem

`nls_length_semantics` auf den Wert „`CHAR`“ setzen. Mit dieser Einstellung werden alle `VARCHAR2(n)`-Deklarationen automatisch als `VARCHAR2(n CHAR)`-Deklarationen betrachtet.

Was ist noch zu beachten?

Die Tücke liegt (wie immer) im Detail. Auch wenn die Welt innerhalb der UTF8-Datenbank und der modPLSQL-Webanwendung in Ordnung ist, kann es vorkommen, dass die Anwendung mit anderen Systemen kommuniziert, die weniger gut mit UTF8-Zeichen zurechtkommen. Wenn Sie z.B. mit UTL_FILE aus den Datenbank-Inhalten in Textdateien schreiben, kann es sinnvoll sein, die einzelnen Zeilen zuvor mit dem `CONVERT`-Befehl in den Zeichensatz zu verwandeln, den das Betriebssystem bevorzugt.

Benutzeroberfläche im Autorensystem

Nach der Implementation dieser Änderungen für die Speicherung mehrsprachiger Inhalte wurde die Anwendung (das Web-Autorensystem WebAG Automat) für die Bearbeitung mehrsprachiger Varianten seiner Dokumente erweitert.

Wie sieht eine Autorensystem-Benutzeroberfläche aus, die den Autoren die Arbeit mit den Sprachvarianten ermöglicht, ohne sie bei der Veröffentlichung ihrer Inhalte zu belasten?

Als Belastung empfinden es Autoren, wenn sie gezwungen werden, für jedes ihrer Dokumente alle vorgesehenen Sprach-Übersetzungen liefern zu müssen. Außerdem ist es lästig, die Navigation zwischen den vorhandenen Sprachvarianten organisieren zu müssen.

In Abbildung 1 sieht man, wie die Autoren zu jedem Dokument die vorgesehenen Sprachvarianten aufrufen und bearbeiten können, wenn sie es möchten. Die Autoren müssen sich nicht darum kümmern, dass zwischenzeitlich die eine oder andere Sprachvariante zu einer Webseite fehlt, weil das WebCMS bei der Darstellung der Webseiten nur die vorhandenen Sprachvarianten anbietet.

Sprachumschaltung für die Dokumente im WebCMS

Für den Betrieb einer Website mit mehrsprachigen Inhalten ist es entscheidend, wie elegant die Leser zu den einzelnen Sprachvarianten wechseln können. Da das WebCMS die

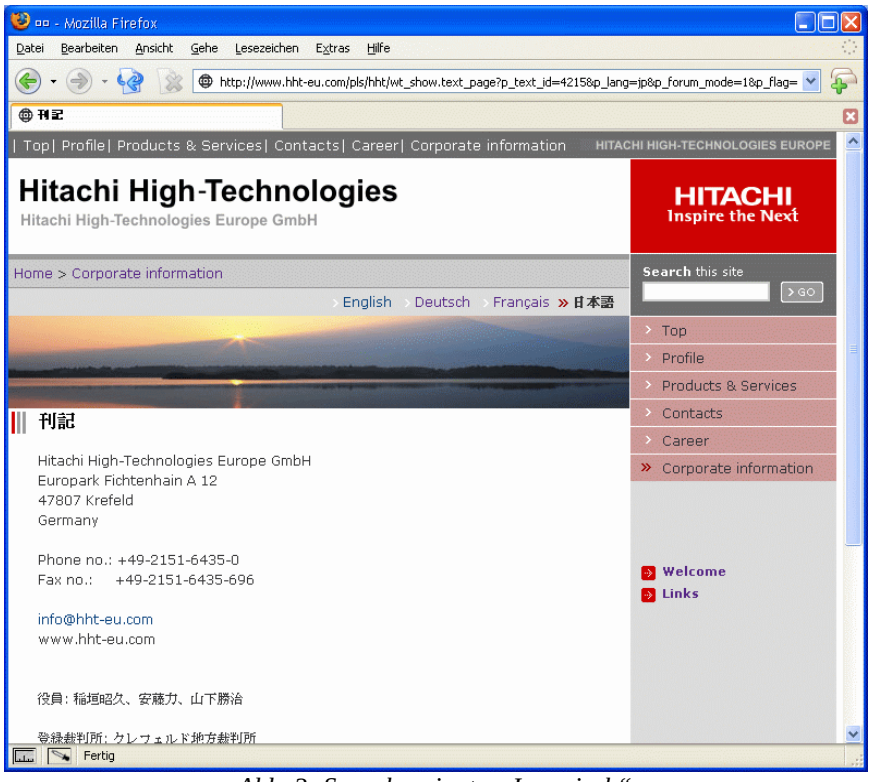

Abb. 3: Sprachvariante „Japanisch“

einzelnen Sprachvarianten zu einem Dokument verwaltet, kann es auf jeder Seite alle verfügbaren Varianten anbieten. Im nachfolgenden Screenshots sieht man, wie die Impressum-Seite einer Website, die mit dem WebCMS *WebAG Automat* betrieben wird, in vier Sprachen - darunter Japanisch – angeboten wird. Die Grafik zeigen die Impressum-Seite der Hitachi High Technologies Europe GmbH, Stand 12. September 2005.
Das HTML-Toolkit des WebCMS lässt den Webdesignern die freie Wahl bei der Gestaltung der Sprachumschaltungs-Navigationsleiste. Oftmals werden die Sprachvarianten nicht als Text-Links, sondern z.B. als Flaggen-Grafiken angeboten.

Freilich wird nicht jede Seite in allen vorgesehenen Sprachen übersetzt werden (z.B. weil der Inhalt u.U. nur für deutsche Leser interessant ist). In diesem Fall wird das WebCMS auch nur die Sprachen in der Sprachumschaltungs-Leiste anbieten, zu denen tatsächlich eine Übersetzung der Seite vorhanden ist.

Historisierung / Versionierung

Die historisierte Datenspeicherung und Versionierung werden mit Oracle-SQL-Mitteln transparent für die Anwendung realisiert. Die entscheidenden Stichworte des Konzeptes sind vorab:

- Tabellen werden in statischen Kopf- und versionierten Datenanteil aufgeteilt
- Die bisherigen Tabellen werden zum Lesen als Views bereitgestellt
- Diese Views erhalten INSTEAD-OF-Trigger zum Schreiben und für die transparente Historisierung
- Versionen werden in einer eigenen Tabelle verwaltet
- Ein PL/SQL-API für die Versionsverwaltung erleichtert die Programmierung der Masken, die für die Versionsverwaltung erstellt werden.

Beispiel:

Die Tabelle `WT_TEXT` soll zukünftig eine Änderungshistorie speichern und versioniert werden.

Historisierung – Änderungen verfolgen

Die Tabelle wird in einen HEAD-Bereich für die nicht versionierten Spalten aufgeteilt und in einen DATA-Bereich für die Spalten, zu denen die Änderungshistorie gespeichert werden soll.

WT_TEXT : 1			
TEXT_ID	DOM_ID	<pk>	not null
LANG_ID	DOM_ID	<fk5>	null
FOLDER_ID	DOM_ID	<fk1>	not null
STENCIL_ID	DOM_ID	<fk4>	null
HEADER	DOM_HEADER		null
SUB_HEADER	DOM_HEADER		null
MODIFIED_BY	NUMBER(10)		null
MODIFIED_AT	DATE		null
VALID_FROM	DATE		null
VALID_TO	DATE		null
KEYWORDS	VARCHAR2(2000)		null
BLOB_NAME	VARCHAR2(128)		null

Abb. 4: Ursprüngliche Tabelle

Pro Änderung wird in der DATA-Tabelle je ein Satz gespeichert. Die Historie der Änderungen wird in einer lückenlosen Folge von `VC_VERSION_FROM/-TO`-Werten gespeichert.

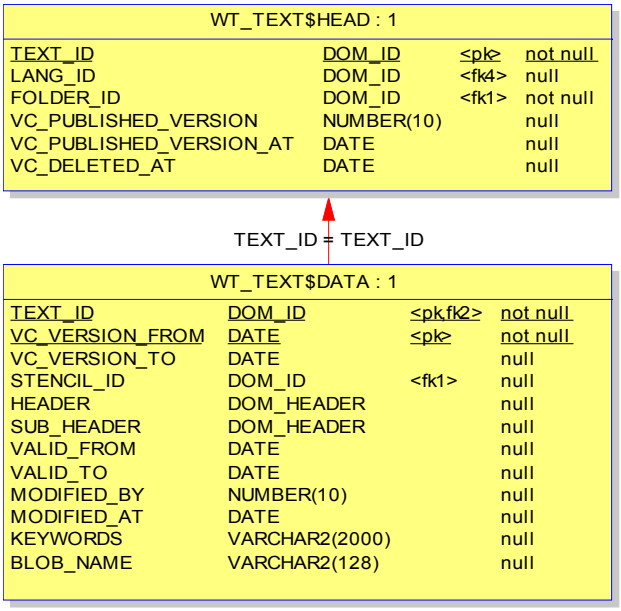

Abb. 5: Aufteilung in Kopf- und Datenanteil

Der Primary Key der ursprünglichen Tabelle „TEXT_ID“ kann weiterhin für andere Tabellen als FK-Join-Column dienen, weil er im HEAD-Bereich weiterhin eindeutig ist.

Die ursprüngliche Tabellenstruktur wird mit Hilfe einer View simuliert. Diese View liefert alle Spalten der ursprünglichen Tabelle durch einen Join der HEAD- mit der DATA-Tabelle.

Eine Frage ist bei der Verwendung einer solchen View noch zu beantworten: Welchen Satz aus dem DATA-Bereich soll sie zu einer TEXT_ID liefern? Wenn die View die ursprüngliche Tabelle WT_TEXT simulieren soll, muss es sich pro ID um einen einzigen Satz handeln. Welcher das ist, hängt nicht zuletzt von der Anwendung ab: Autoren, die an der neuen Version eines Dokuments arbeiten, müssen natürlich jeweils den letzten Bearbeitungsstand sehen. Leser der Webseite hingegen sehen normalerweise den *zur Veröffentlichung freigegebenen* Stand.

Durch eine temporäre Tabelle wird dieses Problem gelöst. Die Anwendung, die auf das Datenmodell zugreift, kann in die temporäre Tabelle eintragen, welche TEXT_ID welchen Stand sehen soll. Dazu wird in die Tabelle neben der TEXT_ID eine DATE-Spalte befüllt. Die View, welche die ursprüngliche Tabelle simuliert, beinhaltet einen geschickten Join zu dieser temporären Tabelle, um zu ermitteln, welchen Satz aus dem DATA-Bereich sie lesen soll. Wenn in der temprorären Tabelle kein Satz zur TEXT_ID gefunden wird, liefert die View immer den *veröffentlichten* Zustand.

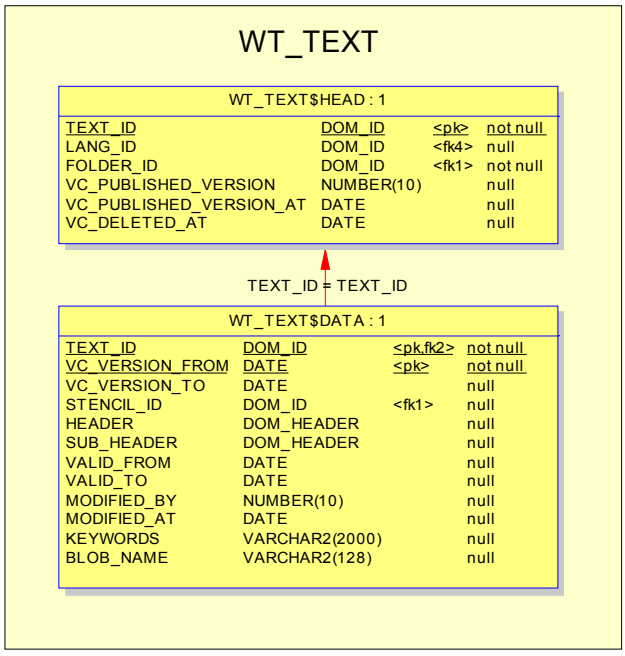

Abb. 6: Eine View simuliert die alte Tabellenstruktur

Damit wird gewährleistet, dass alte Anwendungen, die nicht für die historisierte Datenhaltung angepasst wurden, immer den freigegebenen Datenbestand lesen, während aktuelle Änderungen der Autoren bis zur Veröffentlichung verborgen bleiben. Das Autorensystem

hingegegen trägt mit einem PL/SQL-Aufruf im Versionierungs-API jeweils ein, dass es den neuesten Stand sehen möchte, weil die Autoren an diesem Stand weiterarbeiten möchten.

Mit diesen Maßnahmen ist es gelungen, die bisherige Tabellenstruktur für SELECT's weiterhin zur Verfügung zu stellen, als hätte es keine Datenmodelländerung gegeben.

Auszug aus dem Versionierungs-API:

```
create or replace package wt_vc_api is
(...)
   PROCEDURE set_view_date (
      i_text_id    IN NUMBER,
      i_view_date  IN DATE DEFAULT
NULL
   );

   PROCEDURE set_view_latest (
      i_text_id    IN NUMBER DEFAULT
NULL
   );
(...)
end;
/
```

Abb. 7: Auszug aus dem Versionierungs-PL/SQL-API

Wenn der bisherige Code der Anwendung nicht geändert werden soll, muss zusätzlich eine ähnliche Transparenz für die schreibenden Befehle `INSERT/UPDATE/DELETE` hinzugefügt werden. Die Aufgaben übernehmen `INSTAD-OF`-Trigger. Sie verteilen die DML-Befehle auf die beiden unter der View liegenden Tabellen.

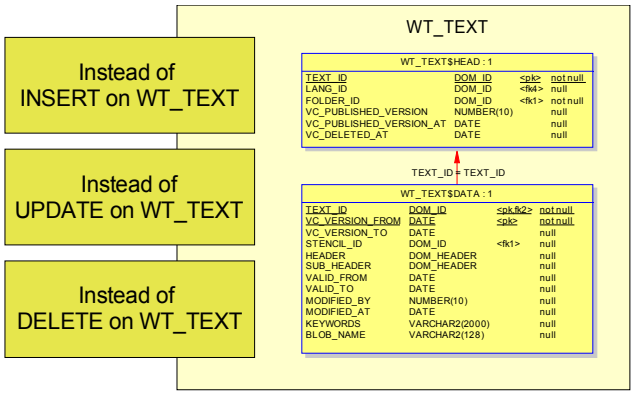

Abb. 8: Instead-Of-Trigger für DML-Befehle

Besonderheiten:

- Wenn lediglich Spalten aus dem HEAD-Teil geändert wurden, wird kein neuer DATA-Satz geschrieben.
- Bei mehreren Änderungen in der gleichen Sekunde wird der DATA-Satz überschrieben und keine neue Version angelegt.

Beziehungen zwischen historisierten Tabellen

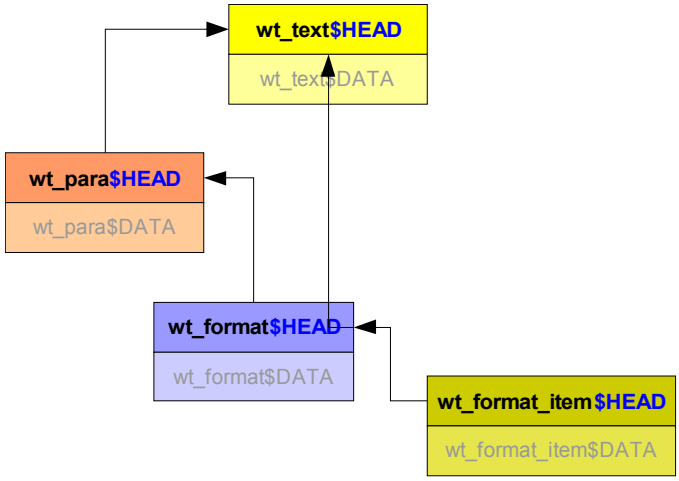

Abb. 9: Beziehungen zwischen Tabellen

Andere Tabellen, zu denen die Änderungshistorie gespeichert werden soll, verwenden die gleiche Technik zur Historisierung, sind also auch in HEAD- und DATA unterteilt und simulieren die ursprüngliche Tabelle mit einer View und `INSTEAD-OF`-Triggern. Solche Tabellen sind jeweils über die HEAD-Tabellen miteinander verbunden.

Wenn eine Anwendung aus den Views liest, die jeweils jede der Tabellen umschließt, wird sie ohne weitere Anpassungen im Sourcecode insgesamt aus allen Tabellen einen in sich konsistenten Stand erhalten. Die optionale Eintragung des gewünschten Zeitpunktes in die temporäre Tabelle wirkt sich auf alle Objekte aus.

Versionierung

Mit den o.a. Datenmodelländerungen wurde erreicht, dass alle Änderungen in den umgebauten Tabellen protokolliert werden. Darauf aufbauend wird die Versionierung hinzugefügt.

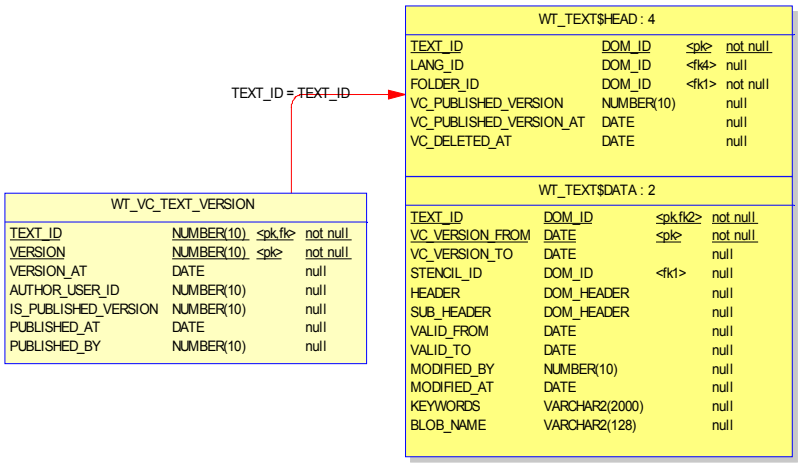

Abb. 10: Versionsnummern in einer eigenen Tabelle speichern

Das zugrundeliegende Konzept sieht vor, dass eine Version für ein Dokument im CMS den Zustand zu einem exakten Zeitpunkt innerhalb der Änderungshistorie repräsentieren soll. Genau eine der Versionen eines Dokuments ist die „*derzeit veröffentlichte*“ Version.

Versionsnummern werden in einer eigenen neuen Tabelle gespeichert. In dieser Tabelle werden die einzelnen Versionen eines CMS-Dokuments abgelegt, indem zu jeder `TEXT_ID` als Primary Key des Top-Objekts `WT_TEXT` eine Versionnummer und der sekundengenaue Zeitpunkt gespeichert wird, dessen Änderungszustand die Version repräsentieren soll.

Das oberste Objekt im Baum der versionierten Tabellen *(hier WT_TEXT als Top-Objekt eines Dokuments im WebCMS)* wird als Referenzobjekt für die Versions-nummern-Zuordnung verwendet. Alle historisierten Detail-Tabellen ergänzen das Gesamtobjekt „Dokument im CMS“ jeweils mit den Datensätzen, die zum definierten Zeitpunkt `VERSION_AT` der jeweiligen Version innerhalb der Änderungshistorie passt.

Kontaktadresse:

Enterprise Web AG
Tonhallenstraße 19
D-47051 Duisburg
Telefon: +49(0)203-2952550
Fax: +49(0)203-2952599
E-Mail info@webag.com
Internet: www.webag.com