Historisierung und Versionierung

für ein bestehendes Datenmodell

ohne Änderung der Anwendung

Martin Friemel, Martin Kubitza
Enterprise Web AG, Duisburg

Versionierung und Historisierung

Konzept / eingesetzte Datenbank-Technologie

- Tabellen in statischen Kopf- und versionierten Datenanteil aufteilen
- Bisherige Tabellen als Views bereitstellen zum Lesen
- INSTEAD-OF-Trigger für die Views zum Schreiben und für die transparente Historisierung
- Versionen in einer eigenen Tabelle verwalten
- PL/SQL-API für die Versionsverwaltung

Versionierung und Historisierung

Tabelle zerteilen – ursprüngliche Tabelle

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

Versionierung und Historisierung

Tabelle zerteilen

Vorher

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

Nachher

WT_TEXT$HEAD : 1			
TEXT_ID	DOM_ID	<pk>	not null
LANG_ID	DOM_ID	<fk4>	null
FOLDER_ID	DOM_ID	<fk1>	not null
VC_PUBLISHED_VERSION	NUMBER(10)		null
VC_PUBLISHED_VERSION_AT	DATE		null
VC_DELETED_AT	DATE		null

TEXT_ID = TEXT_ID

WT_TEXT$DATA : 1			
TEXT_ID	DOM_ID	<pk,fk2>	not null
VC_VERSION_FROM	DATE	<pk>	not null
VC_VERSION_TO	DATE		null
STENCIL_ID	DOM_ID	<fk1>	null
HEADER	DOM_HEADER		null
SUB_HEADER	DOM_HEADER		null
VALID_FROM	DATE		null
VALID_TO	DATE		null
MODIFIED_BY	NUMBER(10)		null
MODIFIED_AT	DATE		null
KEYWORDS	VARCHAR2(2000)		null
BLOB_NAME	VARCHAR2(128)		null

Versionierung und Historisierung

Tabelle zerteilen

HEAD-Tabelle enthält statische, nicht-historisierte Informationen.

Der PK TEXT_ID kann weiterhin für andere Tabellen als FK-Join-Column dienen.

DATA-Tabelle speichert versionierte Daten.

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

WT_TEXT$HEAD : 1

TEXT_ID	DOM_ID	<pk>	not null
LANG_ID	DOM_ID	<fk4>	null
FOLDER_ID	DOM_ID	<fk1>	not null
VC_PUBLISHED_VERSION	NUMBER(10)		null
VC_PUBLISHED_VERSION_AT	DATE		null
VC_DELETED_AT	DATE		null

TEXT_ID = TEXT_ID

WT_TEXT$DATA : 1

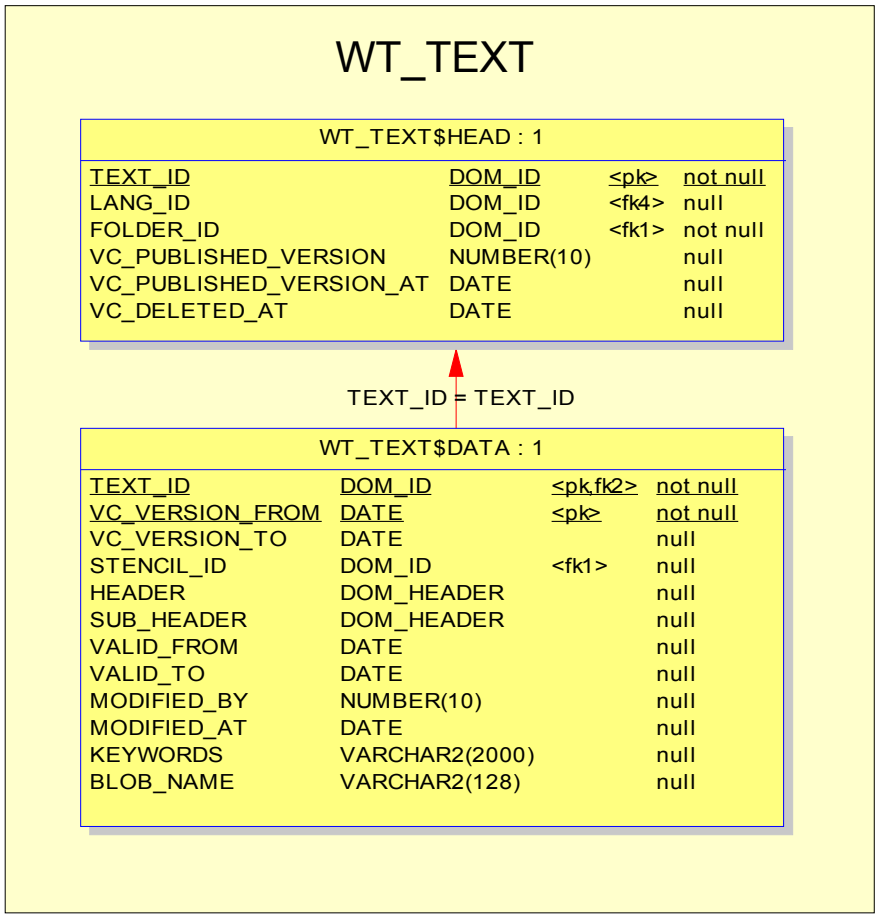
TEXT_ID	DOM_ID	<pk,fk2>	not null
VC_VERSION_FROM	DATE	<pk>	not null
VC_VERSION_TO	DATE		null
STENCIL_ID	DOM_ID	<fk1>	null
HEADER	DOM_HEADER		null
SUB_HEADER	DOM_HEADER		null
VALID_FROM	DATE		null
VALID_TO	DATE		null
MODIFIED_BY	NUMBER(10)		null
MODIFIED_AT	DATE		null
KEYWORDS	VARCHAR2(2000)		null
BLOB_NAME	VARCHAR2(128)		null

Versionierung und Historisierung

View mit bisherigem Tabellennamen

Die View liefert alle Spalten der ursprünglichen Tabelle durch einen Join der HEAD- mit der DATA-Tabelle.

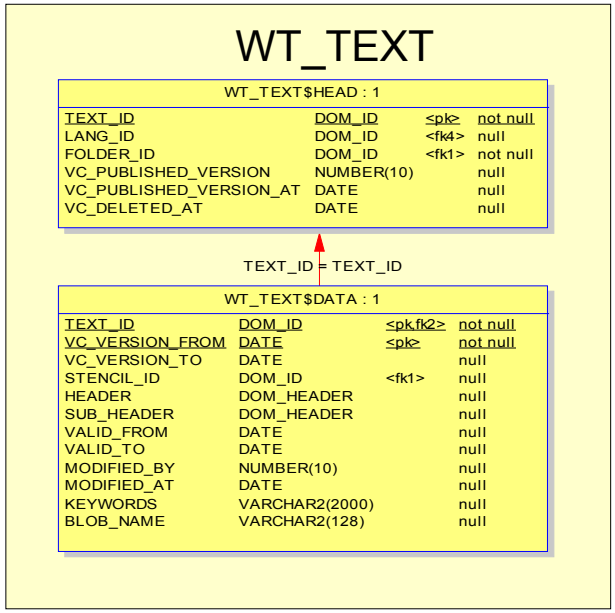
Von den DATA-Sätzen wird nur der eine Satz gelesen, den die Anwendung „sehen“ soll.



Versionierung und Historisierung

View mit bisherigem Tabellennamen

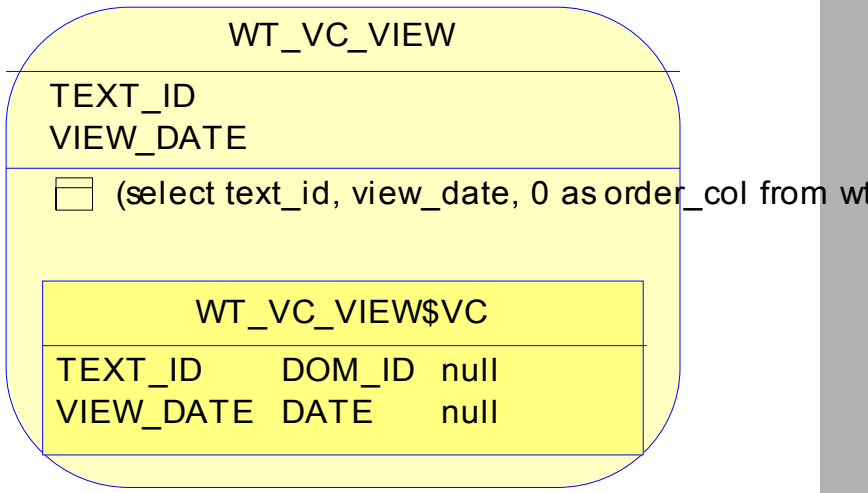
```
create or replace view WT_TEXT  as
   SELECT th.TEXT_ID, th.LANG_ID, th.FOLDER_ID, th.NAME,
          td.STENCIL_ID, td.HEADER, td.SUB_HEADER,
          td.MODIFIED_BY, td.MODIFIED_AT,
          td.VALID_FROM, td.VALID_TO,
          td.KEYWORDS, td.BLOB_NAME,
          td.vc_version_from,
          td.vc_version_to,
          th.vc_published_version,
          th.vc_published_version_at,
          tv.version,
          tv.version_at
     FROM wt_text$head        th,
          wt_text$data        td,
          wt_vc_text_version tv,
          wt_vc_view          vi
    where td.text_id = th.text_id
      and tv.text_id (+) = th.text_id
      and tv.version (+) = 1
      and (vi.text_id = th.text_id or vi.text_id is null)
      and (td.vc_version_from <= NVL(vi.view_date,th.vc_published_version_at))
      and (td.vc_version_to   >  NVL(vi.view_date,th.vc_published_version_at));
```



Versionierung und Historisierung

Welchen Stand soll die Anwendung sehen?

Eine View sorgt dafür, dass für jede TEXT_ID immer genau ein Satz gelesen werden kann - Entweder mit einem zuvor gesetzten VIEW_DATE oder mit NULL als VIEW_DATE.

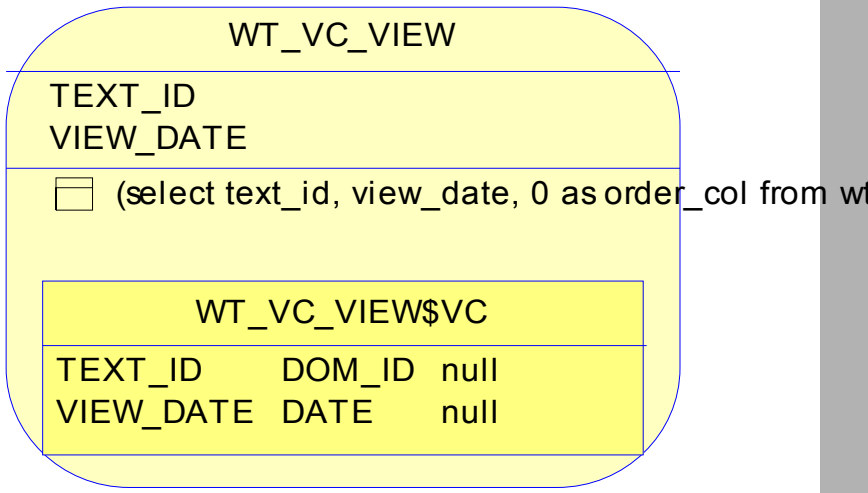



```
create or replace view WT_VC_VIEW as
   select text_id, view_date
     from (
            -- Wenn vorhanden:
            select text_id,
                   view_date,
                   0 as order_col
              from wt_vc_view$vc
            union
            select to_number(NULL) as text_id,
                   to_date(NULL)   as view_date,
                   1               as order_col from dual
            order by order_col
          )
    where rownum < 2;
```

Versionierung und Historisierung

Welchen Stand soll die Anwendung sehen? - PL/SQL-API

Eine PL/SQL-API bietet Operationen zum Setzen des Zeitpunktes, dessen Zustand für die Anwendung „sichtbar“ sein soll.




```
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;
/
```

Versionierung und Historisierung

Wie werden Versionen verwaltet?

Die Tabelle WT_speichert zu jeder TEXT_ID die Versionsnummern und den Stand, den die jeweilige Version repräsentiert.

In WT_TEXT$HEAD wird hinterlegt, welche der Versionen die aktuell veröffentlichte Version ist.

TEXT_ID = TEXT_ID

WT_VC_TEXT_VERSION			
TEXT_ID	NUMBER(10)	<pk,fk>	not null
VERSION	NUMBER(10)	<pk>	not null
VERSION_AT	DATE		null
AUTHOR_USER_ID	NUMBER(10)		null
IS_PUBLISHED_VERSION	NUMBER(10)		null
PUBLISHED_AT	DATE		null
PUBLISHED_BY	NUMBER(10)		null

WT_TEXT$HEAD : 4			
TEXT_ID	DOM_ID	<pk>	not null
LANG_ID	DOM_ID	<fk4>	null
FOLDER_ID	DOM_ID	<fk1>	not null
VC_PUBLISHED_VERSION	NUMBER(10)		null
VC_PUBLISHED_VERSION_AT	DATE		null
VC_DELETED_AT	DATE		null

WT_TEXT$DATA : 2			
TEXT_ID	DOM_ID	<pk,fk2>	not null
VC_VERSION_FROM	DATE	<pk>	not null
VC_VERSION_TO	DATE		null
STENCIL_ID	DOM_ID	<fk1>	null
HEADER	DOM_HEADER		null
SUB_HEADER	DOM_HEADER		null
VALID_FROM	DATE		null
VALID_TO	DATE		null
MODIFIED_BY	NUMBER(10)		null
MODIFIED_AT	DATE		null
KEYWORDS	VARCHAR2(2000)		null
BLOB_NAME	VARCHAR2(128)		null

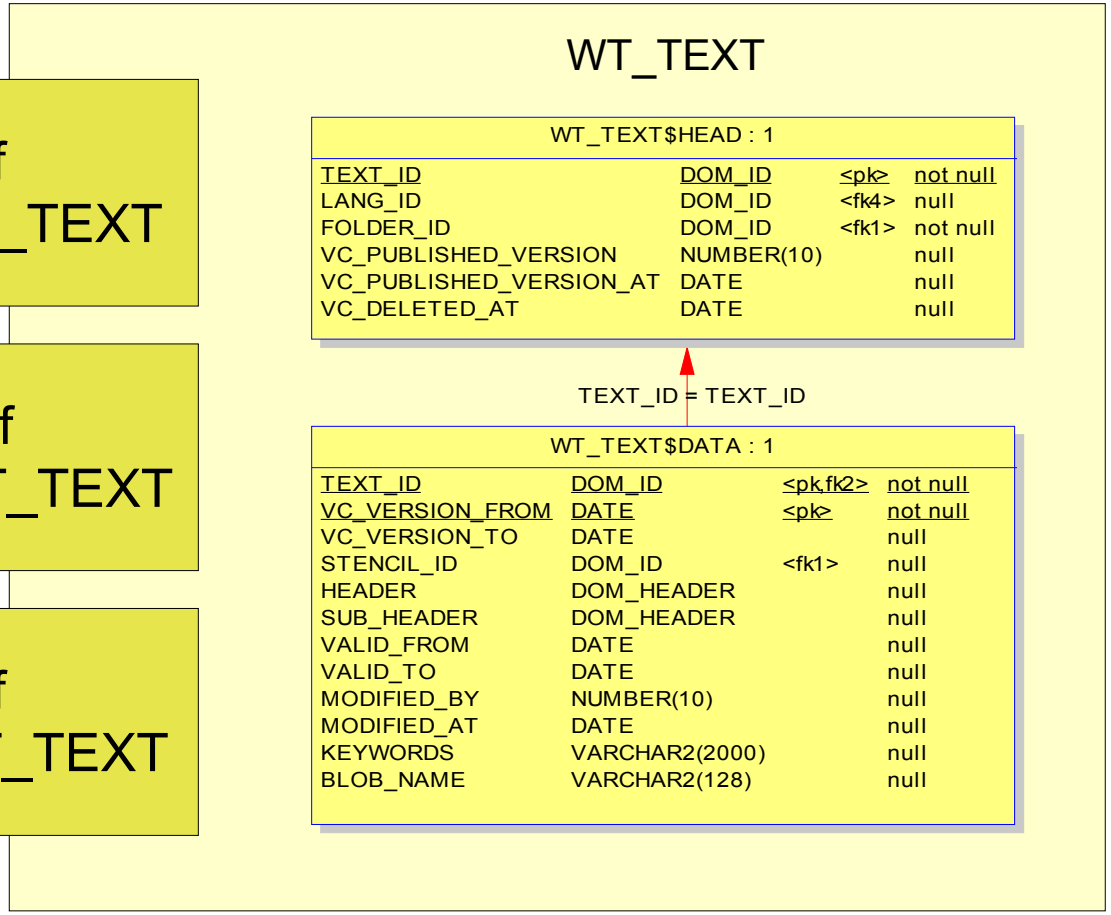
Versionierung und Historisierung

Ändern mit INSTEAD-OF-Triggern

Instead of
INSERT on WT_TEXT

Instead of
UPDATE on WT_TEXT

Instead of
DELETE on WT_TEXT



Versionierung und Historisierung

Ändern mit INSTEAD-OF-Triggern

Instead of
INSERT on WT_TEXT

Instead of
UPDATE on WT_TEXT

Instead of
DELETE on WT_TEXT

Die Trigger verteilen die DML's auf die beiden unter der View liegenden Tabellen.

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.

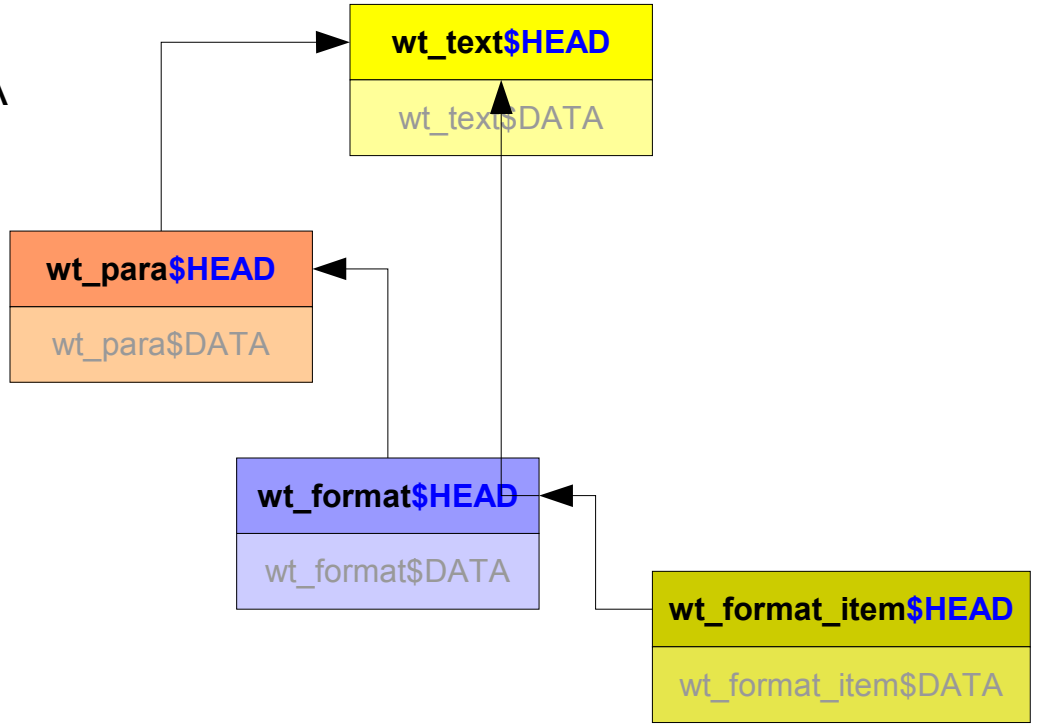
Versionierung und Historisierung

Detail-Tabellen

Detail-Tabellen 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.

Die HEAD-Tabellen sind über FK-Constraints verbunden.

Eine Versionsnummer erhält jedoch nur das oberste Objekt WT_TEXT.



Versionierung und Historisierung

Anwendung

WebAG Automat
WebCMS / Autorensystem

Versionsverwaltung für Webseiten