DDL-Befehle automatisch mit CVS-Informationen protokollieren

Kunde:	DOAG*News*
Ort, Datum:	Artikel im Heft Q1 / 2006
Thema / Themen:	**Artikel von merlin.zwo**
Projekt:	**DDL-Befehle automatisch mit CVS-Informationen protokollieren**
Autor:	**Stefan Winkler**

- Oracle & Technologien
- Systementwicklung
- Individuelle Lösungen

Im Rahmen einer professionellen Softwareentwicklung ist eine Versionsverwaltung heutzutage selbstverständlich. Die Ausführung von DDL-Befehlen lässt sich mit den Bordmitteln von Oracle sehr gut protokollieren. Dieser Beitrag beschreibt, wie DDL-Logging und CVS-Protokollierung integriert und vor allem automatisiert werden können.

In der Gemeinde der Oracle-Entwickler ist Concurrent Versions System (CVS) wohl eines der weitverbreitesten Systeme zur Verwaltung von Source Code. Entwickler checken ihren Code mit Änderungskommentaren ein, CVS versioniert ihn, speichert Meta-Daten sowie Sourcen und ermöglicht Differenzanzeige, Source Code Merging und vieles andere mehr über Kommandozeile oder GUI. ORACLE-Entwickler, die CVS bzw. den Nachfolger SubVersion benutzen, erstellen Skripte zur Pflege von DB-Objekten wie Tabellen, Views, Stored Procedures. Deren Ausführung auf der Datenbank wird zwar von Oracle im Data Dictionary dokumentiert – jedoch natürlich völlig ohne Verknüpfung der CVS-Versionsinformationen. Auch die Möglichkeit des Auditings von Oracle zeichnet zwar die strukturelle Änderungshistorie auf – aber ebenfalls ohne CVS-Bezug.

Probleme, Anforderungen und Ziele

Durch die Unabhängigkeit von Oracle und CVS treten in der Praxis folgende Probleme und Fragestellungen auf:

- Welche CVS-Skriptversion wurde für ein DB-Objekt installiert?
- Welcher Entwickler hat wann was in welcher Version installiert?
- Wurde die Änderung Skript-basiert mit CVS-Bezug ausgeführt oder interaktiv?
- Wenn mit CVS-Bezug ausgeführt: welche CVS-Version?
- In welcher Reihenfolge wurden welche Skripte installiert?
- Wurden Skripte mehrfach ausgeführt?
- Welcher Entwickler hat von welchem System aus mit welchem Tool Änderungen vorgenommen?

In der Praxis sind die meisten Fragen nicht oder nur mit sehr hohem detektivischem Rechercheaufwand beantwortbar!

merlin.zwo ging dieses Thema im Rahmen zahlreicher Individualsoftware-Projekte mit folgenden Zielen an:

1) DDL-Änderungen sollen automatisch und zwangsweise in einer Oracle-Tabelle protokolliert werden.
2) In der Protokolltabelle sollen möglichst viele CVS-Informationen (CVS-Version, Dateiname, Verzeichnis, Autor...) gemeinsam mit Umgebungsinformationen (angemeldeter User (OS + Oracle), Tool, Ausführungszeit...) und DB-Informationen (DB-Objekt, Typ, der Befehl selbst...) aufgezeichnet werden.
3) Es soll ein Template-Skript geben, damit der Entwickler eine Vorlage hat, mit der er möglichst nichts zusätzliches zu seinem zu programmierendem Code tippen muss.

Das Konzept

Seit Oracle Version 9*i* lassen sich datenbankweite Trigger für DDL-Events (außerhalb von DML-Tabellenevents) erstellen. Zur Nutzung der vielen geforderten Umgebungs- und Datenbankangaben ist es mit Oracle 10g besonders leicht diese sogenannten „attribut functions in client events“ bei DB-Triggern einzusetzen (z.B. „ora_dict_obj_owner“).

Das Problem besteht jetzt darin, die CVS-Informationen in einen DB-Protokollierungstrigger zu bekommen.
CVS bietet die Möglichkeit während des CheckIn bzw. CheckOut von (ASCII-)Files durch Keyword-Substitution CVS-Informationen „in das Source Code-Skript hineinzugenerieren“. Diesen generierten Text schreibt man per Procedurecall in verschiedene Package-Variable, die wiederum vom DB-Trigger angezogen werden.

DDL-Befehle automatisch mit CVS-Informationen protokollieren

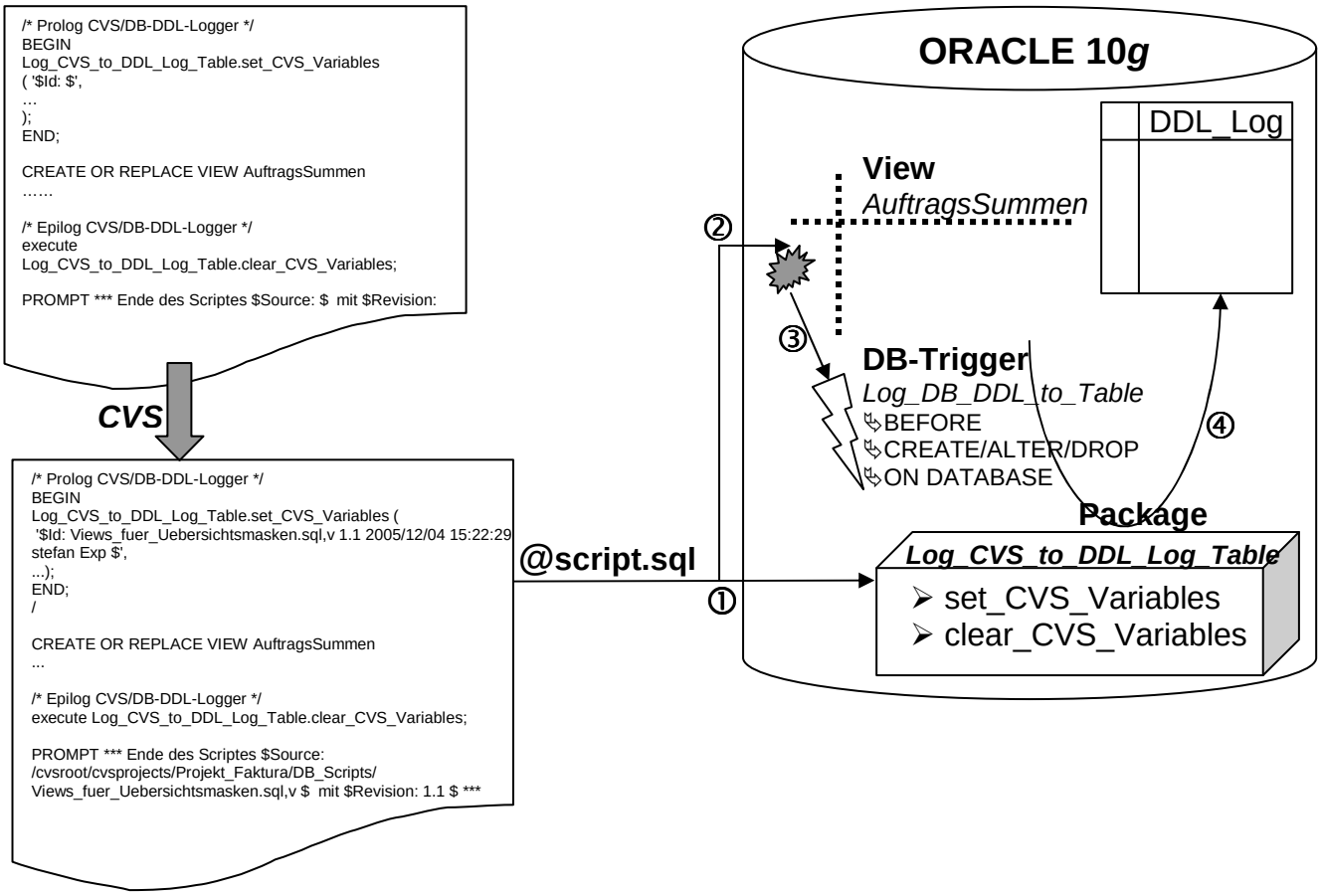

Abbildung 1: Konzeptioneller Überblick der Lösung

Der WorkFlow einer Skript-Installation

... geschieht wie folgt:

a) Template benutzen und den Code wie sonst auch (zwischen Prolog und Epilog) programmieren.

b) Abspeichern in dem CVS-Sandbox-Verzeichnis als Skript.

c) Einchecken des Skripts ins CVS (dabei werden die CVS-Angaben in den Packageaufruf hineingeneriert).

d) Skript installieren...

Im Prolog werden die generierten CVS-Angaben an das Package übergeben. Der eigentliche CREATE- / ALTER- / DROP-Befehl löst den DB-Trigger aus, der seine Informationen um die in der Package geparkten CVS-Informationen ergänzt und schließlich im Transaktionskontext des DDL-Befehles den INSERT in die Protokolltabelle vornimmt.

Der Entwickler braucht im Vergleich zu seinem Arbeitsablauf nichts zu ändern, außer dass er den Prolog und Epilog (z.B. durch Benutzung des Templates) anwenden muss => alle drei gesetzte Ziele werden erfüllt!

Umgebung und Voraussetzungen der Lösung

In unserem Haus positionieren wir die entwickelte Lösung analog zu Oracles Auditing-Mechanismus und haben uns daher entschlossen die programmierten DB-Objekte in einem eigenen Schema zu kapseln – wir nennen es DDL_Log_User. Aus Sicherheitsgründen wurde nach der Installation der Account gelockt – die Lösung protokolliert jedoch weiterhin.

Damit der DDL_Log_User seine Protokollierung durchführen kann, benötigt er diverse Rechte:
CREATE SESSION, CREATE TABLE, CREATE TRIGGER, CREATE PROCEDURE, CREATE ADMINISTER DATABASE TRIGGER, TS-Quota und SELECT auf SYS.V_$OPEN_CURSOR.

Einige interessante Stellen (➔ Listing 1-6) ...

In der Tabelle sind 2 Spalten besonders zu erwähnen:

- OBJEKT_BESITZER – muss auf NULLable gesetzt werden, damit auch Befehle wie ALTER USER-unfallfrei aufgezeichnet werden (hier gibt es keinen „Objekt-Besitz“).
- AUSFUEHRENDER_ORACLE_USER – muss auf NULL geändert werden, wenn manche „Oracle supplied Packages“ ausgeführt werden. Während der Evaluierung des neuen Features „asynchrones, verteiltes CDC (Change Data Capture)“ haben wir festgestellt, dass die attribute function ora_login_user zu NULL wird; alternativ könnte man den DB-Trigger nochmals auf V$-Tabellen ansetzen.

In der packaged Procedure Log_CVS_to_DDL_Log_Table.set_CVS_Variables werden mit substr() die geparkten CVS-Angaben um den „CVS-Ballast“ wie $ u.s.w. erleichtert.
ora_name_list_t ist ein PL/SQL-Table-Typ und die Oracle-Funktion ora_sql_text() gibt im RETURN die Anzahl der SQL-Textfragmente vom Typ BINARY_INTEGER zurück und erwartet als OUT-Argument eine Variable vom Typ ora_name_list_t. Auf die einzelnen Einträge der PL/SQL-Table (d.h. die Fragmente des SQL-Textes) wird mit sql_text(i) zugegriffen.
Bei Tabellen / Spalten ist eine Kommentierung ebenso wie bei Views möglich. Im View-Template wird das benützt und die CVS-Informationen zusätzlich als COMMENTS in Oracles Data Dictionary eingebracht: COMMENT ON TABLE ViewName IS 'Beschreibung zum View ... ($Source: $ mit $Revision: $)'; => es werden Dateiname, Pfadangabe und CVS-Versionsnummer hineingeschrieben.

Ferner ist im Template die Stelle

```
'
 $Log: $
'
```

zu beachten: CVS hat eine eigene Logik wie Zeichen vor den Substitutions-Variablen behandelt werden. Der Eintrag '$Log: $' in einer Zeile ergibt nicht die gewünschte PL/SQL-konforme String-Syntax!

Erfahrungen und Erweiterungswünsche

Die Erstinstallation der Lösung an sich funktionierte auf allen von uns getesteten System von 9*i* bis 10*g*R2 auf Linux und Windows völlig problemlos. Allerdings sollten User, die zum Installationszeitpunkt angemeldet sind und anschließend DDL absetzen wollen, sich unbedingt re-connecten; ansonsten tritt ein „ORA-604 Fehler auf rekursiver SQL-Ebene 1" (package state von log_cvs_to_ddl_log_table discarded) auf.

Die Aufzeichnung des SQL-Befehles wurde auf 4000Byte begrenzt, weil der Code ohnehin im Data Dictionary bzw. im CVS vorhanden ist und dies zur Identifikation des SQL-Befehles völlig ausreicht. Ansonsten müsste man die Tabelle und die Triggervariable auf CLOB umstellen.

Ferner haben wir zum einfacheren Umgang auf unseren Entwicklungsdatenbanken PUBLIC Synonyme und GRANTs für EXECUTE auf die Package und die Tabelle erstellt.

Weiterhin ist zu beachten, dass die technische Lösung immer nur so gut sein kann, wie ihre organisatorische Umsetzung: wenn Entwickler weder Prolog noch Epilog verwenden und in Tools wie SQL*Plus, SQL*Navigator, TOAD u.ä. DDL interaktiv absetzen, dann wird das zumindest aufgezeichnet. Wenn aber innerhalb der gleichen Session der CVS-Prolog benutzt wird und kein Epilog verwendet wird, dann erhalten nachfolgende DDL-Befehle die identischen CVS-Angaben – was natürlich falsch sein kann.

Einen unangenehmen Seiteneffekt haben wir beim Hineingenerieren des CVS-Textes in die Package-Werte festgestellt: gibt ein Entwickler innerhalb des CheckIn-Kommentares ein Hochkomma ein, so wird die Syntax des Package-Aufrufs zerstört. Wenn die organisatorische Anweisung, ohne Hochkommas zu kommentieren, nicht durchsetzbar ist, besteht die Möglichkeit ,in den internen CheckIn-Prozess von CVS (d.h. bevor im SourceCode die CVS-Variablen substituiert werden) einzugreifen. Der CVS-Code ist zugänglich und erfordert Python-Programmierkenntnisse – ist aber machbar.

Für die nähere Zukunft wird merlin.zwo von CVS auf den Nachfolger „SubVersion" umstellen. Da SubVersion sich weitgehend an CVS anlehnt, sind keine nennenswerten Anpassungen der Lösung zu erwarten.

Überraschend war für uns die Anzahl der DDL-Änderungen: obwohl wir CASE-basiert arbeiten und erst mit der Programmierung starten, wenn das Datenmodell eine gereifte Beta-Qualität hat, entstanden in Wochenfrist hunderte von DDL-Logzeilen (auch von Views und stored procedures).
Schließlich entstanden im täglichen SQL-Alltag drei weitere Erweiterungswünsche:
1. die Erstellung eines Views, der von jedem Objekt nur den jeweils neuesten Eintrag zeigt,
2. Verfügbarkeit der CVS-Meta-Daten in ORACLE, um einen Vergleich zwischen CVS-Version und Datenbank-Version durchzuführen. Die Frage „wurde das neueste Skript aus dem CVS auch in der Datenbank installiert?“ ➔ könnte so ohne manuellen CVS-Zugriff aus ORACLE heraus beantwortet werden.
3. Ein Exclude-System für Tabellen/Objekte/User zur Unterdrückung der Protokollierung z.B. beim Aufspielen eines Patches – andererseits ist es vielleicht auch interessant zu sehen, was bei Installationen von Oracle und anderen Anbietern so alles geändert wird ...

Die Skripte zur Realisierung des automatischen DDL- und CVS-Loggers:

```
CREATE TABLE DDL_LOG
(DDL_Typ                        VARCHAR2(30)   NOT NULL,
 Objekt_Typ                     VARCHAR2(30)   NOT NULL,
 Objekt_Name                    VARCHAR2(30)   NOT NULL,
 DDL_Befehl                     VARCHAR2(4000) NOT NULL,
 DDL_Zeitpunkt                  DATE           NOT NULL,
 Objekt_Besitzer                VARCHAR2(30)       NULL,
 ausfuehrender_ORACLE_User      VARCHAR2(30)   NOT NULL,
 ausfuehrender_OS_User          VARCHAR2(30),
 Host                           VARCHAR2(54),
 IP_Adresse                     VARCHAR2(30),
 Modul                          VARCHAR2(48),
 CVS_ID                         VARCHAR2(150),
 CVS_Source                     VARCHAR2(250),
 CVS_RCSFile                    VARCHAR2(100),
 CVS_Revision                   VARCHAR2(30),
 CVS_Date                       DATE,
 CVS_Author                     VARCHAR2(30),
 CVS_TagName                    VARCHAR2(30),
 CVS_Log_History                VARCHAR2(4000),
 AuthentikationsTyp             VARCHAR2(30)   NOT NULL,
 Proxy_User                     VARCHAR2(30),
 Instanz_Nummer                 NUMBER         NOT NULL
);
```

Listing 1: Anlage der Protokoll-Tabelle

```
CREATE OR REPLACE PACKAGE Log_CVS_to_DDL_Log_Table
AS
   PROCEDURE set_CVS_Variables (akt_CVS_ID          IN VARCHAR2,
                                akt_CVS_Source      IN VARCHAR2,
                                akt_CVS_RCSFile     IN VARCHAR2,
                                akt_CVS_Revision    IN VARCHAR2,
                                akt_CVS_Date        IN VARCHAR2,
                                akt_CVS_Author      IN VARCHAR2,
                                akt_CVS_TagName     IN VARCHAR2,
                                akt_CVS_Log_History IN VARCHAR2 );
   CVS_ID          VARCHAR2(150)  := NULL;
   CVS_Source      VARCHAR2(250)  := NULL;
   CVS_RCSFile     VARCHAR2(100)  := NULL;
   CVS_Revision    VARCHAR2(30)   := NULL;
   CVS_Date        DATE           := NULL;
   CVS_Author      VARCHAR2(30)   := NULL;
   CVS_TagName     VARCHAR2(30)   := NULL;
   CVS_Log_History VARCHAR2(4000) := NULL;

   PROCEDURE clear_CVS_Variables;

END Log_CVS_to_DDL_Log_Table;
/
CREATE OR REPLACE PACKAGE BODY Log_CVS_to_DDL_Log_Table
AS
   PROCEDURE set_CVS_Variables (akt_CVS_ID          IN VARCHAR2,
                                akt_CVS_Source      IN VARCHAR2,
                                akt_CVS_RCSFile     IN VARCHAR2,
                                akt_CVS_Revision    IN VARCHAR2,
                                akt_CVS_Date        IN VARCHAR2,
                                akt_CVS_Author      IN VARCHAR2,
                                akt_CVS_TagName     IN VARCHAR2,
                                akt_CVS_Log_History IN VARCHAR2)
   IS
   BEGIN
      SELECT         substr(akt_CVS_ID,         6, LENGTH(akt_CVS_ID)       -  7),
                     substr(akt_CVS_Source,    10, LENGTH(akt_CVS_Source)   - 13),
                     substr(akt_CVS_RCSFile,   11, LENGTH(akt_CVS_RCSFile)  - 14),
                     substr(akt_CVS_Revision, 12, LENGTH(akt_CVS_Revision) - 13),
             to_date(substr(akt_CVS_Date,       8, LENGTH(akt_CVS_Date)     -  9)
                     , 'YYYY/MM/DD HH24:MI:SS'),
                     substr(akt_CVS_Author,    10, LENGTH(akt_CVS_Author)   - 11),
                     substr(akt_CVS_TagName,    8, LENGTH(akt_CVS_TagName)  -  9),
                     SUBSTR(akt_CVS_Log_History,1,4000)
      into CVS_ID,   CVS_Source, CVS_RCSFile, CVS_Revision,
           CVS_Date, CVS_Author, CVS_TagName, CVS_Log_History
      FROM dual;
   END set_CVS_Variables;

   PROCEDURE clear_CVS_Variables
   IS
   BEGIN
      CVS_ID          := NULL;
      CVS_Source      := NULL;
      CVS_RCSFile     := NULL;
      CVS_Revision    := NULL;
      CVS_Date        := NULL;
      CVS_Author      := NULL;
      CVS_TagName     := NULL;
      CVS_Log_History := NULL;
   END clear_CVS_Variables;

END Log_CVS_to_DDL_Log_Table;
/
```

Listing 2: Anlage der Package für CVS-Variable (set + clear)

```
CREATE OR REPLACE TRIGGER Log_DB_DDL_to_Table
 BEFORE
 CREATE OR ALTER OR DROP
 ON DATABASE
DECLARE
   anz_SQL_Teile BINARY_INTEGER;
   sql_text      ora_name_list_t;
   v_stmt        VARCHAR2(4000);
BEGIN
   anz_SQL_Teile := ora_sql_txt(sql_text);

   IF nvl(anz_SQL_Teile ,0) > 0 then
      FOR i IN 1..anz_SQL_Teile LOOP
          IF LENGTH(v_stmt) + LENGTH(sql_text(i)) > 4000 THEN
             EXIT;
          ELSE
             v_stmt := v_stmt || sql_text(i);
          END IF;
      END LOOP;
   END IF;

   INSERT into DDL_LOG
   (DDL_TYP, DDL_ZEITPUNKT, DDL_BEFEHL, OBJEKT_NAME, OBJEKT_BESITZER, OBJEKT_TYP,
    AUSFUEHRENDER_ORACLE_USER, AUSFUEHRENDER_OS_USER, PROXY_USER,
    AUTHENTIKATIONSTYP,
    HOST, IP_ADRESSE, MODUL, INSTANZ_NUMMER, CVS_ID, CVS_SOURCE, CVS_REVISION,
    CVS_DATE, CVS_RCSFile, CVS_Author, CVS_TagName, CVS_Log_History )
   VALUES (ora_sysevent, SYSDATE, v_stmt, ora_dict_obj_name, ora_dict_obj_owner,
           ora_dict_obj_type, ora_login_user, SYS_CONTEXT('USERENV','OS_USER'),
           SYS_CONTEXT('USERENV','PROXY_USER'),
           SYS_CONTEXT('USERENV','AUTHENTICATION_TYPE'),
           SYS_CONTEXT('USERENV','HOST'), SYS_CONTEXT('USERENV','IP_ADDRESS'),
           SYS_CONTEXT('USERENV','MODULE'), ora_instance_num,
           Log_CVS_to_DDL_Log_Table.CVS_ID, Log_CVS_to_DDL_Log_Table.CVS_Source,
           Log_CVS_to_DDL_Log_Table.CVS_Revision,
Log_CVS_to_DDL_Log_Table.CVS_Date,
           Log_CVS_to_DDL_Log_Table.CVS_RCSFile,
Log_CVS_to_DDL_Log_Table.CVS_Author,
           Log_CVS_to_DDL_Log_Table.CVS_TagName,
           Log_CVS_to_DDL_Log_Table.CVS_Log_History )      ;
END Log_DB_DDL_to_Table;
/
```

Listing 3: Der DB-Trigger, der die DDL-Befehle auffängt und mit den CVS-Angaben aufzeichnet

```
/* Prolog CVS+DB-DDL-Logger */
BEGIN
Log_CVS_to_DDL_Log_Table.set_CVS_Variables
(
 '$Id: $',
 '$Source: $',
 '$RCSfile: $',
 '$Revision: $',
 '$Date: $',
 '$Author: $',
 '$Name:  $',
 '
  $Log: $
 '
);
```

```
END;
/

/* Template für Create View */
CREATE OR REPLACE VIEW ViewName
(
 spalte_VC2,
 spalte_NUMBER,
 spalte_DATE
)
AS
SELECT 1 a, 2 b, 3 c
FROM   dual
;

COMMENT ON TABLE ViewName
IS 'Beschreibung zum View ... ( $Source: $  mit  $Revision: $ )';

COMMENT ON COLUMN ViewName.spalte_VC2
IS 'Beschreibung zur Spalte ...';

/* ggfls hier a) Grants und/oder b) Synonyme einfügen... */


/* Epilog CVS+DB-DDL-Logger */
execute Log_CVS_to_DDL_Log_Table.clear_CVS_Variables;

PROMPT *** Ende des Scriptes $Source: $  mit $Revision: $ ***
```

Listing 4: Template-Skript VOR dem CVS-CheckIn

```
/* Prolog CVS+DB-DDL-Logger */
BEGIN
Log_CVS_to_DDL_Log_Table.set_CVS_Variables (
 '$Id: Views_fuer_Uebersichtsmasken.sql,v 1.1 2005/12/04 15:22:29 stefan Exp $',
 '$Source:
/cvsroot/cvsprojects/Projekt_Faktura/DB_Scripts/Views_fuer_Uebersichtsmasken.sql,v
$  mit  $Revision: 1.1 $',
 '$RCSfile: Views_fuer_Uebersichtsmasken.sql,v $',
 '$Revision: 1.1 $',
 '$Date: 2005/12/04 15:22:29 $',
 '$Author: stefan $',
 '$Name:  $',
 '
  $Log: Views_fuer_Uebersichtsmasken.sql,v $
  Revision 1.1  2005/12/04 15:22:29  stefan
  Erste finale Version der Summenübersicht für Masken 101 + 102 der Faktura
 '
);
END;
/
```

```
CREATE OR REPLACE VIEW AuftragsSummen
(
 KundenName, Auftrag_Nr, Umsatz, DB1
)
AS
SELECT Kd_Name, Auf_Nr, sum(VK_Preis * Menge), sum((VK_Preis-EK_Preis) *
Menge)
FROM Artikel_Position
GROUP BY Kd_Name, Auf_Nr
;

COMMENT ON TABLE AuftragsSummen
IS 'Verdichtung auf Auftragsebene ... ( $Source:
/cvsroot/cvsprojects/Projekt_Faktura/DB_Scripts/Views_fuer_Uebersichtsmaske
n.sql,v $  mit  $Revision: 1.1 $ )';

COMMENT ON COLUMN AuftragsSummen.KundenName
IS 'standardisierte KD-Name...';

/* ggfls hier a) Grants und/oder b) Synonyme einfügen... */

/* Epilog CVS+DB-DDL-Logger */
execute Log_CVS_to_DDL_Log_Table.clear_CVS_Variables;

PROMPT *** Ende des Scriptes $Source:
/cvsroot/cvsprojects/Projekt_Faktura/DB_Scripts/Views_fuer_Uebersichtsmaske
n.sql,v $  mit $Revision: 1.1 $ ***
```

Listing 5: Entwickler-Skript NACH dem CVS-CheckIn

```
SELECT * FROM DDL_LOG WHERE Objekt_Name = 'AUFTRAGSSUMMEN';

DDL_TYP: CREATE
OBJEKT_TYP: VIEW
OBJEKT_NAME: AUFTRAGSSUMMEN
DDL_BEFEHL: CREATE OR REPLACE VIEW AuftragsSummen ( KundenName, Auftrag_Nr,
Umsatz..
DDL_ZEITPUNKT: 05.12.2005 11:55:26
OBJEKT_BESITZER: FAKT_OWNER
AUSFUEHRENDER_ORACLE_USER: WINKLER
AUSFUEHRENDER_OS_USER: swinkler
HOST: m2WinTS\WinTS
IP_ADRESSE: 192.168.31.99
MODUL: TOAD 8.6.0.38
CVS_ID: Views_fuer_Uebersichtsmasken.sql,v 1.1 2005/12/04 15:22:29 stefan Exp
CVS_SOURCE:
/cvsroot/cvsprojects/Projekt_Faktura/DB_Scripts/Views_fuer_Uebersichtsmasken.sql
CVS_RCSFILE: Views_fuer_Uebersichtsmasken.sql
CVS_REVISION: 1.1
CVS_DATE: 04.12.2005 15:22:29
```

```
CVS_AUTHOR: stefan
CVS_TAGNAME: DEV
CVS_LOG_HISTORY: $Log: Views_fuer_Uebersichtsmasken.sql,v $ Revision 1.1  2005/12/...
AUTHENTIKATIONSTYP: DATABASE
PROXY_USER:
INSTANZ_NUMMER: 1
```

Listing 6: Abfrage des protokollierten Inhaltes

Referenzhinweise

CVS – Concurrent Versions System: http://www.nongnu.org/cvs/ und http://www.wincvs.org/

SVN – SubVersion: http://subversion.tigris.org/

Sollten Fragen zu diesem Artikel bestehen - ich stehe Ihnen gerne zur Verfügung:

Ihr Ansprechpartner:	Kontakt
Stefan Winkler	stefan.winkler@merlin-zwo.de
Geschäftsführer, Systementwickler	Tel.: 07052 / 933 666 Fax: 07052 / 933 670
merlin.zwo InfoDesign GmbH & Co. KG Karmelstraße 9 75378 Bad Liebenzell	www.merlin-zwo.de
merlin.zwo InfoDesign GmbH & Co. KG Taglöhnergärten 43 76228 Karlsruhe	