

Richtlinien PLSQL und SQL

Ordnungsnummer AA 0080/2009

Version 13.1

Gültig ab 16.10.2025

Ansprechpartner Thomas Heidler, Durchwahl: - 2342

Herausgeber Frank Simon, Durchwahl: - 2658

Bereich IT

Status Änderung

Kommentar Redaktionelle Änderungen

Betrifft IT Informationstechnologie

Inhaltsverzeichnis

Weisungsdokument Arbeitsanweisung Richtlinien PLSQL

Arbeitsanweisung Richtlinien PL/SQL und SQL

Version 13.1 von 10/2025

Status: in Arbeit / vorgelegt / **freigegeben**

Auf Anforderung stellt die IT ein Exemplar mit vollständiger Änderungsmarkierung zur Verfügung.
Alle Regelungen gelten geschlechtsneutral. Geschlechtsspezifische Formulierungen dienen lediglich der Vereinfachung.

Schlagworte

Quelltext, Dateinamen, Dokumentation, Kommentare, SQL, PLSQL, ORACLE, Tabelle, View, Trigger, Package, Exception, Fehlerbehandlung, Wiederaufsetzbarkeit, PISql-Developer, PISqlDoc, Templates, DML, DDL

Gegenstand des Dokumentes

Programmierrichtlinien für PISql und Sql, zugehörige Namenskonventionen und Dokumentationsrichtlinien

Gültigkeitsbereich

IT: IT1, IT2, IT3, IT4, IT5
Projekte

Elektronischer Ablageort

http://172.16.2.171/intranet/weisungen/dv/SW-Entwicklung/0080_2009/aktuell/pdf.pdf

Änderungsübersicht

Version	Datum	Autor	Bemerkung
001	19.08.1996	K. Becker	Neuentwicklung
002	07.02.2003	Ch. Weber	Neufassung
003	24.04.2003	Ch. Weber	Einschränkungen für DDL-Statements hinzugefügt
004	10.08.2005	Ch. Weber	Änderungen im Zusammenhang mit Einführung von PISqlDoc
005	27.10.2005	Ch. Weber	Hinzufügen von Namenskonventionen für Sql- und PISql-Software-Objekte
006	09.03.2006	Ch. Weber	Ergänzung bei View-Namen
007	19.04.2007	Liebl	Konstantenbezeichner ergänzt (Seite 10)
008	27.06.2007	Liebl	1. Allgemeine Überarbeitung 2. Anpassung an neue Dokumentationsvorgaben
009	22.07.2008	Liebl	1. Kap. ‚2.3.3 DML-Statements‘ ergänzt 2. Diverse Soll- durch Muss-Bestimmungen ersetzt
010	16.09.2009	Liebl	Neue Kapitel eingefügt: - Allgemeine Angaben (Kap. 1) - Weisung: Jedes Case-Statement muss einen Else-Zweig besitzen (Kap.2.3.7) - Empfehlung: Variable initialisieren (Kap. 2.3.8) - Empfehlung: Join in der From-Clause ausführen (Kap. 2.3.9) Codebeispiele an aktuelle Templates angepasst
011	14.06.2019	Nencki	Verwendung aktuelles IT-S-Template und Anpassung der Kapitelformate, Anpassung an neue Orgstruktur, redaktionelle Anpassungen
012	09/2021	Kroll	Anpassung an neue Orgstruktur
013	10/2023	Schmid Rauscher	Anpassung an neue Orgstruktur und neues Team IT5; keine inhaltlichen Änderungen, redaktionelle Änderungen
13.1	10/2025	Heidler	Redaktionelle Änderungen

Informationen zum Dokument

Dieses Dokument beruht auf anderen Dokumenten in der jeweils gültigen Fassung, die im Folgenden angegeben sind:

Nr.	Titel
1	
2	
3	
4	
5	

Inhaltsverzeichnis

1	ALLGEMEINE ANGABEN	5
1.1	Vorwort	5
1.2	Richtlinienkennzeichnung	5
2	PROGRAMMIERRICHTLINIEN.....	6
2.1	Namenskonventionen	6
2.1.1	Namen von SQL- und PLSQL-Software-Objekten	6
2.1.2	Namen von Variablen und Konstanten	7
2.1.3	Dateinamen	8
2.2	Dateiaufbau	8
2.3	Statements	9
2.3.1	Deklarationen	9
2.3.2	DDL-Statements	10
2.3.3	DML-Statements	10
2.3.4	Packages	11
2.3.5	Trigger	11
2.3.6	View	11
2.3.7	Case Statement	11
2.3.8	Initialisierung von Variablen	11
2.3.9	Join-Syntax	12
2.4	Fehlerbehandlung	12
2.4.1	Beispiele	13
2.5	Wiederaufsetzbarkeit	16
2.5.1	Variante: Verarbeitung beinhaltet nur eine Transaktion	16
2.5.2	Variante: Verarbeitung beinhaltet mehr als eine Transaktion	17
3	DOKUMENTATIONSRICHTLINIEN.....	19
3.1	Zuordnung zum Programmierauftrag	19
3.2	Dokumentation	19
3.2.1	Layoutanforderung an die Modul-Dokumentation	19
3.2.2	Mindestinhalte von Modul-Dokumentationen	19
3.2.3	Mindestinhalte von Kommentaren in Quelltexten	20
3.2.4	Vorlagen/Rahmen	21
3.3	Dokumentation mit PISqlDoc (optional)	22
3.3.1	Allgemeines	22
3.3.2	Dokumentationskommentare	23
3.3.3	Implementationskommentare	25
4	INKRAFTTRETEN	28

1 Allgemeine Angaben

1.1 Vorwort

Die hier beschriebenen Richtlinien definieren Standards, die bei der Entwicklung von SQL - und PLSQL-Software-Objekte, die in der LfA bzw. im Auftrag der LfA erstellt werden, einzuhalten sind. Sie gelten nicht für PLSQL-Code anderen Ursprungs, der in der LfA verwendet wird (z.B. DB-Änderungsscripte, die vom Power-Designer generiert werden).

1.2 Richtlinienkennzeichnung

Bei den in diesem Dokument aufgeführten Richtlinien handelt es sich generell um Weisungen, solange nicht explizit mit dem Symbol  darauf hingewiesen wird, dass es sich bei der rechts neben dem Symbol stehende Richtlinie um eine Empfehlung handelt.

2 Programmierrichtlinien

2.1 Namenskonventionen

2.1.1 Namen von SQL- und PLSQL-Software-Objekten

In der LfA gelten folgende Namenskonventionen für folgende Datenbankobjekte:

2.1.1.1 View

Namensaufbau: [<FremdDB>_]<Präfix>_<Zeichenkombination>

Präfix: ,V'

Zeichenkombination: beliebig, jedoch möglichst sprechend

Empfehlung: falls möglich an Tabellennamen anlehnen,
ggf. Systemkürzel einarbeiten

FremdDB: bei Views auf Fremddatenbanken muss noch ein eindeutiges Kürzel der
Fremd-DB vorangestellt werden (z.B. TMS_V_A_FXRATES_01)

Ausnahme: die View ersetzt eine ehemals vorhandene Tabelle (z.B. BLZDAT, KONTO).
In diesem Fall wird der Tabellename unverändert als Viewname
übernommen.

Länge: maximal 30 Stellen

2.1.1.2 Materialized View

Namensaufbau: <Zeichenkombination>

Zeichenkombination: beliebig, jedoch möglichst sprechend

Empfehlung: falls möglich an Tabellennamen anlehnen,
ggf. Systemkürzel einarbeiten

Hinweis: Materialized Views dürfen nicht Bestandteil des von außen sichtbaren Teils
von zugesicherten externen Schnittstellen sein.

Länge: maximal 30 Stellen

2.1.1.3 Packages, Prozeduren und Funktionen

Namensaufbau: <Zeichenkombination>[<2-stellige Nummer>]

Zeichenkombination: beliebig, jedoch möglichst sprechend
(möglichst mit System-/Funktionskürzel am Anfang)

2-stellige Nummer: optional zur Durchnummerierung

Länge: maximal 30 Stellen

savName	Savepoint		
exName	Exception		
#	l = lokale Variable g = globale Variable p = Parameter		
Name	Kann sich aus mehreren Wörtern zusammensetzen Jedes Wort beginnt mit einem Großbuchstaben, der Rest in Kleinbuchstaben z.B. clSchlBilanz, rISchlBilanz, slAppName, KngAktivJa, flTempDat		

2.1.3 Dateinamen

Ein Dateiname, der PL/SQL-Text bzw. SQL-Text enthält, endet mit einem der Suffixe,

- „.SPE“ für Package-Spezifikationen
- „.BDY“ für Package-Bodys
- „.TRG“ für Trigger
- „.PRC“ für Prozeduren
- „.FCT“ für Funktionen
- „.VIW“ für Views
- „.DDL“ für Datenmodelländerungen (d.h. Power Designer Scripts)
- „.SQL“ für DML-Scripts (z.B. Sonderprozeduren zur Datenänderung)

Für den Namensteil vor dem „.“, d.h. für den Präfix gelten folgende Regeln:

- Sonderscript/-prozedur: Wenn die Datei ein Script oder eine Prozedur enthält, die i.d.R. nur einmalig ausgeführt wird (z.B. direkte Datenänderung, Migrationslauf, Initialisierung), muss der Präfix mit „SOND“ beginnen, gefolgt von einem 2-stelligen Systemkürzel (wenn möglich) und zwei Ziffern (→ Durchnummerieren).
- Wenn in der Datei Quelltext enthalten ist, um ein DB-Objekt anzulegen (Package-Spezifikation, Package-Body, View, Trigger, Prozedur, Funktion) so ist der Präfix gleich dem Objektnamen.
- Wenn mehrere Trigger mit einem Quelltext angelegt werden, dann ist der Präfix gleich dem Programmfunktions-Namen.

Beispiele: KrSplit01.spe
 KrSplit01.bdy
 KrSplitting.trg
 Kr_Delete_Ka_Vertrag.prc
 SONDRA20.sql

Hinweis: Die Namen von Scripten zur Datenmodelländerung sind in den Richtlinien zur Datenmodelländerung beschrieben.

2.2 Dateiaufbau

SQL- und PL/SQL-Quelltext ist so in Dateien aufzuteilen, dass eine einzelne Quelldatei folgendes enthalten kann:

- Eine Folge von SQL-Statements, die eine abgeschlossene Aufgabe zur Datenmanipulation erfüllen (Z.B. eine Sonderprozedur zur Datenänderung, eine abgeschlossene Migrationsaufgabe, die Initialisierung einer Tabelle).
- eine Anweisung zur Erzeugung bzw. Änderung eines der DB-Objekte Prozedur, Funktion, Package-Spezifikation, Package-Body, View, Trigger.
- Mehrere DDL-Anweisungen zur Änderung des Schemas, die vom Power-Designer generiert werden.
- Anweisungen zum Erzeugen bzw. Ändern mehrerer Trigger, sofern es sich um Standard-Trigger handelt oder sie zu einer gemeinsamen Programmfunktion gehören.

Hinweis: In Quelldateien zum Anlegen von DB-Objekten wird immer, wenn es syntaktisch erlaubt ist „create or replace ...“ verwendet, nicht „create ...“.

2.3 Statements

2.3.1 Deklarationen

Es muss je ein Bezeichner je Zeile deklariert werden. (Bezeichner ist ein Sammelbegriff für: Parameter, Variable, Konstante, Exception, ...)

Immer wenn es möglich ist %Type – Deklarationen verwenden. Dies reduziert den Pflegeaufwand bei Format-änderungen an Tabellen. (z.B. werden Längenänderungen, sofern sie keine Code-Änderungen nach sich ziehen durch %Type - Deklarationen automatisch vollzogen).

%Type-Deklarationen sind nicht möglich, wenn nicht feststeht, aus welcher Tabelle/Spalte ein Parameter/eine Variable befüllt wird (z.B. weil es verschiedene für diese Prozedur relevante Tabellen/Spalten gibt oder wenn manipulierte Spalteninhalte aufgenommen werden müssen usw.). Namen müssen sprechend sein. Sie können z.B. an Namen von Tabellenspalten angelehnt sein oder/und den Verwendungszweck im Code-Zusammenhang enthalten.

Alle Bezeichner müssen einen Kommentar aufweisen (Empfehlung: in der nachfolgenden Zeile).

Ausnahme: String-Konstante, die nur Ausgabertext beinhalten und selbsterklärend sind (Fehler-, Meldungs-, Protokolltexte)

```
Beispiel:      function fct_umbrechen ( spName1                in Person.Name1%Type
-- 1. Bestandteil des Personennamens
,spName2                in Person.Name2%Type
-- 2. Bestandteil des Personennamens
,spName3                in Person.Name3%Type
-- 3. Bestandteil des Personennamens
,spWelcherName          in varchar2
--Wertebereich: ('NAME1' , 'NAME2' , 'NAME3')
--Bedeutung: welcher Bestandteil des umgebrochenen
--Namens soll geliefert werden
) return varchar2(40)
-- der ermittelte Namensbestandteil
as
    /* Konstanten */
    KnITyp                CONSTANT number := 1;
    -- Umbrechungsmodus: Rückgabe in Großbuchstaben
    KslHinweisNoName      CONSTANT varchar2(40) := 'Kein Name vorhanden';
    ...
```

2.3.2 DDL-Statements

Direkte DDL-Statements auf Bestands-Tabellen sind wegen Verletzung des Zugriffsschutzes nicht möglich.

Nur in begründeten Ausnahmefällen dürfen aber temporäre DB-Tabellen, die nur innerhalb einer Session existieren per Create + Drop bereitgestellt und wieder entfernt werden. Sie müssen nicht im Datenmodell definiert werden, da sie im Schema LFA_APP angelegt werden. Der Tabellenname muss mit dem Präfix TMP beginnen und ein Merkmal (z.B. Zeitstempel oder Session-Id) beinhalten, um Eindeutigkeit herzustellen.

Im Standardfall sind auch „temporäre“ Tabellen im Datenmodell permanent anzulegen und während einer Session zu befüllen und wieder zu leeren (mit DBDDL9.Truncate_Table). Alternativ bzw. ergänzend dazu kann auch mit PISql-Tabellen gearbeitet werden.

Um ausgewählte DDL-Statements auch auf Bestands-Tabellen (LFA_DAT) ausführen zu können (z.B. Rebuild Index, Disable/Enable Trigger, Truncate Table, ...) steht das Package DBDDL9 zur Verfügung.

2.3.3 DML-Statements

Um eine gewisse „Resistenz“ gegen nachträgliche Änderungen an der Tabellen-Struktur herstellen zu können, wird bei Insert- und Select- Statements folgendes Vorgehen favorisiert:

Select-Statements

Grundsätzlich müssen Select-Statements so verwendet werden, dass spätere Änderungen der Struktur der Tabelle keine fehlerhaften Auswirkungen haben. Zum Beispiel sind folgende Schreibweisen unabhängig von der Tabellen-Struktur:

Verwendung der Spalten-Namen

```
"select <Spaltenname1>, ... <SpaltennameN> from <Tabellenname(n)> into Var1, ...  
VarN;"
```

Verwendung %rowtype

```
„r|Var <Tabellenname>%rowtype;  
select * from <Tabellenname> into r|Var;"
```

Insert-Statements

Insert-Statements müssen so definiert werden, dass spätere Änderungen der Struktur der Tabelle keine fehlerhaften Auswirkungen haben. Zum Beispiel sind folgende Schreibweisen unabhängig von der Tabellen-Struktur:

Verwendung der Spalten-Namen

Einfügen von Werten:

```
„Insert into <Tabellenname>  
    (<Spaltenname1>, ... <SpaltennameN>)  
values ( Wert1, ... WertN);“
```

Einfügen von Werten mit Subquery:

```
„Insert into <Tabellenname>  
    (<Spaltenname1>, ... <SpaltennameN>)  
    (select ... from ...);“
```

(Insert-Statements bleiben dann valid, wenn nachträglich hinzugefügte Spalten nullable sind oder default-Werte haben).

Ausnahme: Zur Realisierung von Kopierfunktionen (z.B. Datensätze innerhalb derselben Tabelle duplizieren, Bestandskopien anfertigen) darf mit ‚Select * from ...‘ und ‚Insert into <Tabellenname> values ...‘ gearbeitet werden.

2.3.4 Packages

Für Package-Spezifikation und Package-Body werden zwei getrennte Quelltext-Dateien je Package angelegt.

Dabei beginnt die Spezifikation von Prozeduren und Funktionen in der Spezifikation mit dem Statement

```
„CREATE OR REPLACE PACKAGE/FUNCTION/PROCEDURE <name> AUTHID CURRENT  
USER“.
```

Damit wird spezifiziert, dass die Funktion/Prozedur nicht mit den Berechtigungen des Besitzers des Packages sondern mit den Berechtigungen des Benutzers (LFA_APP) ausgeführt wird.

An den Bodys der Prozeduren/Funktionen ist die AUTHID-Klausel nicht zu wiederholen.

2.3.5 Trigger

Trigger_Code muss möglichst kurz gehalten werden. Wenn die Aufgabe des Triggers umfangreichen Quellcode erfordert, muss er in eine Funktion/Prozedur ausgelagert werden.

Grund: Trigger-Code wird wie ein SQL-Script Zeile für Zeile zur Ausführungszeit interpretiert. Der Quellcode einer Prozedur wird als pCode in geparster Form im DB-Kernel gehalten. Er kann sehr viel schneller ausgeführt werden. Man kann auf diese Weise Performance-Verluste durch feuermde Trigger kleiner halten.

Das Ausführen eines Commit oder Rollback im Trigger und den daraus aufgerufenen Funktionen/Prozeduren ist verboten.

2.3.6 View

Wird bei der Select-Liste ein Wert über eine umfangreichere Formel/Berechnung (z. B. Funktion, Case-Konstrukt) ermittelt, so muss für diesen Wert ein Aliasname vergeben werden.

2.3.7 Case Statement

Jedes Case-Statement muss einen Else-Zweig besitzen.

2.3.8 Initialisierung von Variablen

Variable sind vor ihrer Erstverwendung zu initialisieren.

E

2.3.9 Join-Syntax

E Wenn möglich das Joinen innerhalb der From-Clause (nicht in der Where-Clause) vornehmen, wobei jedoch das Verwenden des Using-Joins verboten ist.

Beispiele:

```
Select * from Person
Inner Join Anschrift On Person.Anschrift_Seq = Anschrift.Anschrift_Seq

Select * from Ka_Vertrag
Left Join Ka_Angebot On Ka_Vertrag.Kav_Seq = Ka_Angebot.Kav_Seq and
Ka_Angebot.Aktiv_Jn = 1
where Ka_Vertrag.Kav_Seq = 12345
```

2.4 Fehlerbehandlung

Blöcke müssen grundsätzlich folgendes Format besitzen:

```
declare
    <Deklarationen>
begin
    <Anweisungen>
exception
    <Ausnahmebehandlung>
end;
```

Dabei kann ggf. auf den Deklarationsteil verzichtet werden, wenn es nichts zu deklarieren gibt.

Nicht verzichtet werden kann i.a. auf die Ausnahmebehandlung.

Eine sinnvolle Ausnahmebehandlung ist die Voraussetzung dafür, dass

- die Ursachen für Laufzeitfehler schnell gefunden und behoben werden können
- Batch-Routinen wiederaufsetzbar gemacht werden können
- der Programmier-Aufwand beim Verwenden einer Prozedur/Funktion sinkt.

Wünschenswert ist es, die exception handler so zu programmieren, dass bei jedem Fehler aus der Fehlermeldung hervorgeht

- a) an welcher Stelle der Applikation
- b) welcher Fehler
- c) bei der Behandlung welcher Daten

aufgetreten ist.

Oracle liefert ohne geeignetes exception handling nur die Information b).

Die Mindestanforderung an einen exception handler ist, dass er den Namen des scheiternden Moduls zusätzlich zum aufgetretenen ORA-Fehler nennt.

Im Fall einer Trigger-Definition muss ein solcher einfacher exception handler nicht explizit hinzugefügt werden, ein scheiternder Trigger nennt seinen Namen automatisch in der Fehlermeldung. Für Prozeduren und Funktionen muss dies selbst programmiert werden.

Bei modularer Programmierung und entsprechender Interaktion zwischen den Modulen und mit der Forderung nach Wiederaufsetzbarkeit wachsen die Anforderungen an die exception handler. Es muss auch die Reihenfolge der einander aufrufenden Module im Fehlerfall ermittelt werden können. Dafür müssen die exception handler der beteiligten Module ihre Fehlermeldungen im Error-stack ablegen.

Eine weitere sinnvolle Variante für Fehlerbehandlung kann das Protokollieren der Fehler in Applikations-Fehlertabellen sein.

Je nach Anwendungsfall muss eine geeignete Art der Ausnahmebehandlung festgelegt und umgesetzt werden.

In diesem Zusammenhang sind grundsätzlich auch Festlegungen über die Art der Transaktionssteuerung erforderlich (d.h. in welcher „Programmebene“ erfolgen Commit bzw. Rollback. ? Z.B.: enthalten die Prozeduren/Funktionen ein Commit/Rollback? Oder erfolgt das in der rufenden SQL Windows-Applikation?).

2.4.1 Beispiele

2.4.1.1 Variante: Erzeugen einer Fehlermeldung mit Name des gescheiterten Moduls

Lösung a) Verwendung des Package Global_Error:

```
EXCEPTION
  WHEN OTHERS THEN
    GLOBAL_ERROR.Init_Funktion('CalcTmpCFLaufzbEinzelValutiert');
    GLOBAL_ERROR.Error_Msg;
end CalcTmpCFLaufzbEinzelValutiert;
```

Lösung b) Inhaltlich dieselbe Funktionalität mit Raise_Application_Error:

```
EXCEPTION
  WHEN OTHERS THEN
    rollback;
    RAISE_APPLICATION_ERROR(-20000, 'Programmfehler: ' || sqlerrm ||
                                   ' CalcTmpCFLaufzbEinzelValutiert' );
end CalcTmpCFLaufzbEinzelValutiert;
```

In beiden Varianten wird die Fehlernummer 20000 und eine Meldung mit ORA-Error-Message und Modulname erzeugt.

In beiden Varianten kann die Message durch weitere Angaben wie Abbruchzeit ... erweitert werden.

2.4.1.2 Variante: Erzeugen einer Fehlermeldung mit Name des gescheiterten Moduls und Aufrufparametern

Lösung c) Verwendung von Fehlerstack und Savepoint:

(Dies ist die empfohlene Standard-Variante, wenn nicht ausdrücklich eine andere Verfahrensweise zu Transaktionssteuerung und Exception handling für eine Applikation erforderlich ist)

```
...  
BEGIN  
savepoint sp_primaer01_fct_Konto_Bezog;  
  
...  
exception  
  WHEN others THEN  
    rollback to sp_primaer01_fct_Konto_Bezog;  
    raise_application_error(-20010, 'Ora-Fehler: ' || sqlcode  
                                || ' in primaer01.fct_Konto_Bezog('  
                                || pPerson_Seq || ') '  
                                , true);  
  
END;
```

Dies ist eine komfortablere Variante:

Dieser Exception handler rollt die begonnene Transaktion zum Savepoint zurück (.d.h. hier: nur den Teil, der innerhalb der gescheiterten Prozedur stattfand, alles was in der „rufenden“ Prozedur /Transaktion evtl. bereits vor Beginn der gescheiterten Prozedur/Funktion erfolgreich getan wurde bleibt erhalten.). Er schreibt eine Fehler-meldung mit der Fehlernummer -20010 in den Stack, ohne bereits darin liegende Meldungen zu überschreiben.

Dies wird gesteuert durch den dritten Parameter der Prozedur `raise_application_error`.

(dritter Parameter = true	→ Hinzufügen zu stack
dritter Parameter = false (default)	→ Überschreiben des stack).

Der Text der Fehlermeldung enthält den aufgetretenen Fehlercode und den Funktionsaufruf mit Parameter, bei dem der Fehler auftrat.

Bei fehlerhafter Beendigung der Prozedur wird der vollständige Inhalt des Fehlerstack als Meldung ausgegeben. Das heißt, auch die Fehlermeldungen der rufenden Routine(n) werden aus dem stack mit ausgegeben.

Bemerkung:

Wenn die Transaktion in der „rufenden“ Ebene (bezüglich der hier betrachteten Prozedur/Funktion) gesteuert wird, muss ein exception handler nicht „rollback“ enthalten, er kann es aber. Rollback to savepoint sollte aber trotzdem sinnvoll sein .

Grundsätzlich wird die Verwendung von Savepoints (wie oben) ausdrücklich empfohlen, und sollte nur dann nicht verwendet werden, wenn die gewählte Variante der Transaktions-Steuerung dies verbietet. (siehe Variante d .)

Die Namen der Savepoints müssen wegen der notwendigen Eindeutigkeit u.a. aus den Namen von Package und Prozedur gebildet werden.

Zu beachten ist, dass bei großer Schachtelungstiefe Längen-Probleme mit den aneinander gehängten Fehlermeldungen auftreten können. Es entsteht dann ein Laufzeitfehler (Wert-Fehler) während der Fehlerbehandlung. Um das zu verhindern müssen die entstandenen Strings mit der substring-Funktion gekürzt werden, was aber wieder zu Informationsverlust führen kann.

2.4.1.3 Variante: Fehlerprotokollierung in einer Fehler-Tabelle

Lösung d)

```
Procedure Abgleich_KRED_PRIMAER(pKBS in Person.KBS%Type)
...
BEGIN
...
exception
  WHEN others THEN
    rollback ;
    slFehlertext := substr(sqlerrm,1,255);
    insert into ABGLEICH_KRED_PRIMAER_LOG (
      ABGLEICH_KPL_SEQ
    , DATUM
    , PROZEDUR
    , FEHLERMELDUNG
    )
      values( Abgleich_KRED_PRIMAER_LOG_SEQ.nextval
      , sysdate
      , ' primaer02.Abgleich_KRED_PRIMAER '
      , slFehlertext
    );
    commit;
    raise_application_error(-20000 ,substr(
      'Ora-Fehler in
primaer02.Abgleich_KRED_PRIMAER '
      ||sqlerrm, 1, 255)
      , true);
end; -- Abgleich_KRED_Primaer
```

Dieser Exception handler rollt die begonnene Transaktion vollständig zurück, schreibt die Fehlermeldung mit Datum in eine Log-Tabelle, committet diesen Protokolleintrag, trägt die Fehlermeldung in den Error-stack ein und beendet die Ausführung der Prozedur mit Fehler. (Es erscheint als Fehlermeldung der gesamte Inhalt des Fehler-Stack mit den Fehlermeldungen aller rufenden Routinen, sofern auch die exception handler dieser Routinen den dritten Parameter auf true gesetzt hatten)

Diese letzte Fehlerprotokollierung in Log-Tabellen kann für die Kontrolle von Batch-Komponenten sehr nützlich sein. Sie vermeidet Probleme mit String-Längen, die ggf. bei Ketten von Fehlermeldungen in der Message auftreten können. Die zueinander gehörenden Fehlermeldungen erscheinen in mehreren Zeilen der Fehlertabelle.

Rollback to Savepoint ist in dieser Variante i.a. nicht sinnvoll, da anschließend vom Committen des LOG-Eintrages alle evtl. vor dem Savepoint bereits ausgeführten Teile der Transaktion mit erfasst würden.

Alle beschriebenen Varianten von exception handlern haben Vor- und Nachteile. Daher ist es nicht sinnvoll, eine für alle Situationen passende festlegen zu wollen.

Es darf aber auf eine Ausnahmebehandlung nur dann total verzichtet werden, wenn dies für die Erfüllung der Aufgabe des Moduls ausdrücklich erforderlich ist. Das kann z.B. vorkommen, wenn eine Funktion ein Pragma erfüllen muss, das durch die Statements eines exception handlers verletzt würde.

Eine weitere Ausnahme bildet, wie oben beschrieben, eine Trigger-Definition. Die Einschränkungen bezüglich Rollback und Savepoint (sind beide in Triggern nicht zulässig) erstrecken sich auch auf Funktionen/Prozeduren, die von einem Trigger aufgerufen werden.

2.5 Wiederaufsetzbarkeit

Jedes PLSQL-Objekt (Package, Prozedur, Funktion) das in der LfA innerhalb der Abend- bzw. Morgenverarbeitung zum Einsatz kommt - also unter UC4 im sogenannten Batchbetrieb läuft – muss besonderen Ansprüchen genügen → es muss **wiederaufsetzbar** sein.

Unter dem Begriff **Wiederaufsetzbarkeit** ist folgender Sachverhalt zu verstehen:

Tritt während der Verarbeitung eines PLSQL-Objekts ein Zustand ein, der eine fehlerfreie Verarbeitung verhindert (z.B. unvorhergesehener Fehler, Timeout, Fehler aufgrund einer nicht korrekten Datenkonstellation), so muss von dem betroffenen PLSQL-Objekt gewährleistet werden, dass es jederzeit vom Operating erneut aufgerufen werden kann, ohne dass dazu spezielle manuelle Eingriffe nötig sind - mit Ausnahme der Fehlerbehebung natürlich.

Die Anforderung nach **Wiederaufsetzbarkeit** kann beispielsweise folgendermaßen gewährleistet werden:

2.5.1 Variante: Verarbeitung beinhaltet nur eine Transaktion

In diesem Fall müssen folgende Punkte eingehalten werden:

- Am Verarbeitungsbeginn einen Savepoint *SpStartA* setzen.
- Commit erst am Ende der gesamten Transaktion ausführen.
- Im Fehlerfall einen Rollback auf den Savepoint *SpStartA* ausführen und im Fehlertext einen Hinweis darauf ausgeben, dass diese Prozedur/Funktion wiederaufsetzbar ist und zwar mit folgendem Aufruf (Beispiel): `RCLOADJA.NEU`

Hinweis: Es ist zu beachten, dass die Menge der in einer Transaktion verarbeitbaren Daten von der Größe des verwendeten Rollbacksegments abhängig ist. Wenn das gerade benutzte Rollback-Segment zu klein ist, führt dies zu einem Laufzeit-Fehler. Ggf. muss für sehr große Transaktionen explizit ein ausreichend großes Rollbacksegment für die Transaktion verwendet werden.

Beispiel:

```
create or replace procedure testproc(spName in VARCHAR2) is
BEGIN
    SET TRANSACTION USE ROLLBACK SEGMENT < Name des sehr großen Rollback
Segments >;
    -- muss immer erstes Statement einer Transaktion sein
    -- andernfalls entsteht ein Laufzeitfehler
    ...
END testproc;
```

2.5.2 Variante: Verarbeitung beinhaltet mehr als eine Transaktion

In diesem Fall muss jede Transaktion in einer eigenen Prozedur/Funktion hinterlegt werden. Zur Steuerung der Ablauffolge der einzelnen Transaktionen untereinander, wird eine separate Prozedur, die sogenannte Startprozedur benötigt. Der Aufruf erfolgt generell nur über diese Startprozedur. Mit Hilfe eines Aufrufparameters kann gesteuert werden, an welcher Stelle der Ablauffolge wiederaufgesetzt werden soll.

Für jede Prozedur/Funktion gilt dabei folgendes:

- Am Verarbeitungsbeginn einen Savepoint *SpStartA* setzen.
- Commit am Ende der Transaktion (vor Verlassen der Prozedur/Funktion) ausführen.
- Im Fehlerfall einen Rollback auf den Savepoint *SpStartA* ausführen und im Fehlertext einen Hinweis darauf ausgeben, wie der erneute Aufruf über die zugehörige Startprozedur auszusehen hat (Beispiel): `RCLOAD.STARTDELTA(2)`

Beispiel für eine Startprozedur:

Create or replace package body ...

```
...
procedure startdelta
-----
-- Beschreibung:
-- *****
-- Startprozedur für den Aufbau des RC-Deltabestands
-- Parameter:  npAktion_nr
--             Über die Aktionsnr. wird der Einsprungspunkt festgelegt, falls
--             die Prozedur im Fehlerfall wiederholt aufgerufen werden muss.
-----
--
-- Änderungsgeschichte
--
--  Vers.  Datum    Bearbeiter  Änderungsgrund (lt. Antrag)
--  -----
--  001   05.07.2002  ewa/rl    Neuentwicklung
--
-----
( npAktion_nr in number default 0
  -- Einsprungspunkt
)
as

begin

/* Datenbankpaket initialisieren */
Init_Package;

/* Aufsetzpunkt mittels npAktion_nr */
if npAktion_nr = 2 then goto Jumper_02; end if;
if npAktion_nr = 3 then goto Jumper_03; end if;
if npAktion_nr = 4 then goto Jumper_04; end if;
```

```
rcload.aufbau_delta;  
<<Jumper_02>>  
  rcload.aufbau_person;  
<<Jumper_03>>  
  rcload.aufbau_vertrag;  
<<Jumper_04>>  
  rcload.protokoll;  
  
-- Fehlerbehandlung  
exception  
...  
  
end startdelta;
```

3 Dokumentationsrichtlinien

3.1 Zuordnung zum Programmierauftrag

Für jede Quelltext-Datei gibt es in der Software-Verwaltung einen Programmierauftrag gleichen Namens (ohne Suffix). Hier wird auch die Versionsnummer der Modul-Dokumentation mitgeführt.

Folgende Besonderheit gilt für Packages:

- Pro Package gibt es zwei Quelltexte (Spezifikation und Body), aber nur eine Modul-Dokumentation.
- Die Versionsnr. der Dokumentation für Packages wird beim Programmierauftrag zum Package-Body mitgeführt.

3.2 Dokumentation

Für jedes im Kapitel **Namen von SQL- und PLSQL-Software-Objekten** aufgeführte DB-Objekt und für SQL-Skripte muss es im Intranet eine namensgleiche Datei (ohne Suffix) mit der entsprechenden Modul-Dokumentation geben.

Für Trigger gelten folgende Ausnahmen von dieser Vorschrift:

- Es dürfen auch mehrere Trigger in einer Dokumentation beschrieben sein, sofern sie mit einer gemeinsamen Quelltext-Datei erzeugt werden.
- Die Erzeugung der Standard-Trigger erfolgt zentral und es gibt dafür nur eine allgemeingültige Dokumentation.

3.2.1 Layoutanforderung an die Modul-Dokumentation

Das Layout muss den Vorlagen entsprechen, die für die Erstellung der Modul-Dokumentation mit WORD bindend sind → [Word-Dokumente_](#)

3.2.2 Mindestinhalte von Modul-Dokumentationen

Folgende Regeln gelten für den Mindestinhalt:

- Für alle DB-Objekte und SQL-Skripte:
 0. Deckblatt und Inhaltsverzeichnis
 1. Allgemeine Angaben
 - a. Beschreibung
 - b. Änderungsübersicht
 2. Anwendungsbezogene Beschreibung
 - a. Allgemeine Beschreibung
 - b. Voraussetzungen (nur wenn welche existieren)
 - c. Abhängigkeiten (nur wenn welche existieren)
 - d. Fehlerbehandlung (nur wenn welche generiert werden)
 - e. Meldungen (nur wenn welche generiert werden)

Der genaue Aufbau ist in den entsprechenden Vorlagen für eine Worddokumentation beschrieben → [Word-Dokumente](#)

- Zusätzlich für Packages:

Liste der öffentlichen Prozeduren/Funktionen/Variablen/Konstanten/Typen, jeweils mit Beschreibung und Fehlerbehandlung

Liste der privaten Prozeduren und Funktionen, jeweils mit Beschreibung und Fehlerbehandlung

- Zusätzlich für Prozeduren und Funktionen (separat oder in Packages):
 - Aufruf der Prozedur/Funktion
 - Liste der Parameter, jeweils mit Name, Typ und Kommentar
 - Angabe ob ein Commit oder Rollback ausgeführt wird
 - ggf. Meldungen, die erzeugt werden
- Zusätzlich für Funktionen (separat oder in Packages):
 - Returnwert mit Name, Typ und Kommentar
- Zusätzlich für Trigger:
 - getriggerte Tabelle/View
 - getriggertes Ereignis
 - ggf. Einschränkung auf Spalten
 - Level (Row- oder Statement-Trigger)
 - ggf. Meldungen, die erzeugt werden
- Zusätzlich für Views:
 - Angabe, ob die View Trigger besitzt oder Bestandteil einer zugesicherten externen Schnittstelle ist
 - Der entsprechende Create-Befehl, wobei jede Select-Spalte einen Kommentar besitzen muss
 - Liste derjenigen Spalten, bei denen zur Ermittlung des Wertes umfangreichere Formeln/Berechnungen nötig sind (jeweils mit Kommentar, Datentyp und Herkunft)

3.2.3 Mindestinhalte von Kommentaren in Quelltexten

Im Quelltext der DB-Objekte bzw. SQL-Skripte müssen mindestens folgende Kommentare enthalten sein:

- Name des Objekts bzw. Scripts
- Version
- Datum
- Bearbeiter
- Änderungsgrund
- Funktion (grob): prägnante Angabe, was die Aufgabe des Objekts ist (i.d.R. einzeilig)
- Beschreibung (ausführlicher): z.B. Angaben über die Aufgabe des Objektes aus fachlicher Sicht im Zusammenhang des Verfahrens. Die Beschreibung muss ohne Programmier- und DB-Kenntnisse zu verstehen sein.
- Entsprechende aussagekräftige Inline-Kommentare

Der genaue Aufbau ist bereits in den entsprechenden Templates hinterlegt:

- Sourcedateien bzw.
- LfA-Templates für PISqlDoc (Allgemeines)

3.2.4 Vorlagen/Rahmen

Zur Vereinheitlichung von Sourcedateien und der dazugehörigen Modul-Dokumentation stehen Standardtemplates und Rahmendokumente (für WORD) zur Verfügung, die bei der Erstellung einer neuen Version herangezogen werden sollten.

3.2.4.1 Word-Dokumente

Im Verzeichnis

K:\IT\03_System-Dokumentationen\SW-Entwicklung\Vorlagen\Doku

stehen für die gängigen DB-Objekte und SQL-Scripte Rahmendokumente zur Verfügung, die zur Erstellung der Modul-Dokumentation mit WORD zu verwenden sind

3.2.4.2 Sourcedateien

Im Verzeichnis

K:\IT\03_System-Dokumentationen\SW-Entwicklung\Vorlagen\Templates

stehen für die gängigen DB-Objekte und SQL-Scripte Templates zur Verfügung, die zur Erstellung der Quelltexte verwendet werden müssen.

3.3 Dokumentation mit PISqlDoc (optional)

3.3.1 Allgemeines

Zur Generierung von technischen Dokumentationen (zusätzlich zur Modul-Dokumentation) kann das Werkzeug PISqlDoc (AddOn für den PISqlDeveloper) eingesetzt werden. PISqlDoc erzeugt HTML-Dokumente aus Kommentaren/Platzhaltern, die in einem fest vorgegebenen Format dem Quelltext hinzugefügt werden müssen.

Diese Form der Dokumentation kann nur dann genutzt werden, wenn bei der Programmierung im PISqlDeveloper eines der im Verzeichnis H:\PLSQLDeveloper\...\Template\Program units hinterlegten LfA-Templates benutzt wird:

- View: LfA_View.tpl
- Package: LfA_Package specification.tpl / LfA_Package-Prozedur.tpl / LfA_Package-Funktion.tpl
- Prozedur: LfA_Procedure.tpl
- Funktion: LfA_Function.tpl
- Trigger: LfA_Trigger.tpl / LfA_Trigger1.tpl / LfA_Trigger3.tpl

Bei Verwendung dieser LfA-Templates:

- kann bei Packages auf einzelne Textpassagen bei der Modul-Dokumentation verzichtet werden.
- können bei Views und Prozeduren einzelne Textpassagen der Modul-Dokumentation aus der technischen Dokumentation 1:1 übernommen werden.

Es wird bei den jeweiligen Vorlagen für die Worddokumentation (siehe Kapitel Word-Dokumente) explizit darauf hingewiesen.

In diesem Kapitel wird zwischen „Dokumentations-“ und „Implementationskommentaren“ unterschieden.

- Dokumentationskommentar: Kommentarblock, aus dem PISqlDoc die Dokumentation generiert und der daher den für PISqlDoc erforderlichen Formatierungs-Regeln genügen muss. Ein Users Guide für PISqlDoc befindet sich unter:
H:\PLSQLDeveloper\...\PISqlDoc\PISqlDoc.doc

- Implementationskommentar:

Beliebiger Kommentar, der zusätzlich in den Programmcode eingefügt wird und nicht im Zusammenhang mit PISqlDoc steht.

Insbesondere muss auch jede nicht öffentliche Funktion/Prozedur einen Kommentarblock haben, der in Inhalt und Aufbau den Dokumentationskommentaren der öffentlichen Funktionen und Prozeduren entspricht.

3.3.2 Dokumentationskommentare

Um die Dokumentationskommentare inhaltlich korrekt aufzubauen und an den Stellen zu platzieren, die PISqlDoc zum Generieren von Dokumentationen verlangt, müssen beim Erstellen der Quelltexte die LfA-Templates im Verzeichnis „H:\PISqlDeveloper\Template\Program units“ verwendet werden. Dazu muss dieses Verzeichnis dem PISqlDeveloper unter

Tools/Preferences/Files/Direktories/Templates

bekanntgegeben werden.

In den LfA-Templates werden zur optischen Aufbereitung der später zu generierenden Dokumentation, die im PISqlDoc-Users Guide beschriebenen „Tags“ verwendet.

3.3.2.1 Packages

PISqlDoc erstellt Package-Dokumentationen aus der Package-Spezifikation, d.h. Dokumentationskommentare für Packages können sich ausschließlich in den Package-Spezifikationen befinden. Das bedeutet, sie enthalten ausschließlich Informationen, die „öffentlich“ sind.

Alle Kommentare in Package-Bodys zählen zu den Implementationskommentaren.

3.3.2.2 Views

Dokumentationen für Views werden von PISqlDoc nicht aus dem Quelltext (Select-Statement) generiert, sondern aus den in der DB hinterlegten „Comments“ zum View und zu den Spalten. Deshalb ist es für das Generieren von View-Dokumentationen mit PISqlDoc unerlässlich, dem Quelltext entsprechende Comment – Statements hinzuzufügen.

Hinweis: Es empfiehlt sich, bei der Definition von Views nach dem Schlüsselwort „SELECT“ den Inhalt des View-Comments zu wiederholen, wenn es gewünscht wird, dass dieser Kommentar auch mit dem Quelltext des Views (Select-Statement), ohne Heranziehen der Dokumentation, angezeigt wird.

3.3.2.3 Beispiel: Package-Spezifikation

create or replace package RaAuftrag01 authid current_user

--
-- DB-Paket : RaAuftrag01
--
-- Version : 001 - Weber vom 31.03.2005
--
-- Funktion: : Bestandteil der DB-Zugriffsschicht Rating
--
-- Beschreibung: Mapping von schreibenden Zugriffen auf den
-- View V_RatingAuftrag auf die zugrundeliegenden Tabellen

-- Änderungsgeschichte

--

Vers.	Datum	Bearbeiter	Änderungsgrund (lt. Antrag)
-------	-------	------------	-----------------------------

-- 001	31.03.2005	Weber	Neuerstellung
-- 002	12.08.2005	Weber	Erweiterung der Prozedur Upd_RaAuftrag um einen Parameter cpResponse_Ori

is

...

-- Prozeduren

--
procedure Ins_RaAuftrag (npAuftrag_Seq in Ra_Auftrag.Raauftrag_Seq%TYPE
-- Identifikator des Ratingauftrags
, npZustand_Seq in Ra_Auftrag.Auftragz_Seq%TYPE
-- Identifikator des Bearbeitungszustandes des Auftrags
, npRating_Seq in V_RATINGAUFTTRAG.Rating_Seq%TYPE
-- Identifikator des Ratings dessen Maschinenrating mit diesem
Auftrag
-- eingeholt wird
, dpRequest_Dat in Ra_Auftrag.Request_Dat%TYPE
-- Zeitstempel des Absendens des Requests
, npAktiv_Jn in Ra_Auftrag.Aktiv_Jn%TYPE
-- Ja/Nein-Flag für ,aktiv/inaktiv'

```
,npTime_Open in Ra_Auftrag.Time_Open%TYPE default NULL
-- Zeitdauer für den Verbindungsaufbau
,npFehler_Code in Ra_Auftrag.Fehler_Code%TYPE default NULL
-- Fehlercode
,spFehler_Text in Ra_Auftrag.Fehler_Text%TYPE default NULL);
-- Fehlertext
```

```
--
-- Funktionen
```

```
--
...
```

```
end RaAuftrag01;
```

3.3.3 Implementationskommentare

Implementationskommentare gliedern und erläutern den Programmcode. Sie werden in den Quelltext eingefügt und beschreiben die programmtechnische Umsetzung der im Dokumentationskommentar beschriebenen Aufgaben und Teilaufgaben. Implementationskommentare müssen einen Überblick über den Quelltext geben und zusätzliche Informationen zum Verständnis liefern, die nicht ohne weiteres aus dem Quelltext ersichtlich sind. Insbesondere sollen auch Hinweise auf die durch den Code umgesetzten Regeln, Konzepte, Algorithmen, ... beigefügt werden.

Grundsätzlich gilt:

Es soll auf alle Kommentare verzichtet werden, die den Quelltext wiederholen oder in Umgangssprache übersetzen. Jeder Kommentar muss Informationen liefern, die beim Lesen des Quelltextes nicht unmittelbar zu ersehen sind.

Diese Kommentare enthalten immer:

- Informationen darüber, wie die Aufgabe ausgeführt wird, d.h. sie enthalten i.a. die Beschreibung des realisierten Algorithmus und ggf. Beschreibungen der technischen Umsetzung.
- Die Änderungsgeschichte eines Prozedur/Funktions-Bodys enthält genau die Versionsnummer der Package-Body-Änderungsgeschichte, für die diese Prozedur/Funktion entwickelt/angepasst werden musste. D.h., die Versionen eines Prozedur/Funktions-Bodys sind im allgemeinen nicht fortlaufend.
Diese Vorschrift dient dazu, schnell feststellen zu können, an welchen Stellen des Package-Bodys Korrekturen vorgenommen wurden um eine konkrete Version des Packages erstellen zu können.

Kommentare sollten weiterhin nicht verwendet werden, um trickreichen Code zu erläutern. Stattdessen sollte der Code vereinfacht werden.

Ausnahme: Komplexe und „trickreiche“ SQL-Statements

Sie sind genau dann erlaubt, wenn sie das Ergebnis von SQL-Tuning-Maßnahmen sind und somit aus Gründen der besseren Performance auf die gute Lesbarkeit verzichtet werden muss. In diesem Fall muss sehr ausführlich kommentiert werden, es sollten ggf. auch Gründe für gewählte Lösungen vermerkt werden, damit ein Nachfolger nicht erfolglose Versuche wiederholt.

3.3.3.1 Beispiel: Inline-Kommentar

-- diese Transaktion soll nur solche Sätze der TMP_PRIMAER_LOG behandeln,
-- die bereits bei Start des Abgleichs vorhanden waren. Alle, die während des laufenden
-- Abgleichs erst eingestellt werden sollen erst beim nächsten Abgleich
-- mit erfasst werden:

```
set transaction isolation level serializable;
```

-- alle noch fehlenden KBS ermitteln

-- dazu erst Person_seq ermitteln, wo nur eine Anschrift_Seq gegeben ist:

```
update TMP_PRIMAER_LOG  
  set TMP_PRIMAER_LOG.PERSON_SEQ = (select Person_Seq  
                                     from anschrift
```

...

-- dann alle KBS ermitteln, wo die Person_Seq gegeben ist:

```
update TMP_PRIMAER_LOG  
  set TMP_PRIMAER_LOG.KBS = (select kbs  
                              from person
```

...

3.3.3.2 Beispiel: Package-Body und Prozeduren

create or replace package body RaAuftrag01

```
--
-- DB-Paket   :   RaAuftrag01
--
-- Version    :   001 - Weber vom 23.08.2005
--
-- Funktion:   :   DB-Zugriffsschicht für Rating-Aufträge
--
-- Beschreibung: enthält Funktionalität zur Datenmanipulation
--                  für den View V_RatingAuftrag
```

```
--
-- Änderungsgeschichte
```

```
--
-- Vers.   Datum   Bearbeiter   Änderungsgrund (lt. Antrag)
-- -----
-- 001    31.03.2005   Weber       Neuerstellung
-- 002    12.08.2005   Weber       Erweiterung der Prozedur Upd_RaAuftrag um
--                  einen Parameter cpResponse_Ori
```

```
is
```

```
-- Prozeduren
```

procedure Set_DvProgParamNumRAOFFL01

```
-- Beschreibung:
-- *****
-- Ändert einen DV-Programm-Parameter für RAOFFL01
```

```
--
-- Änderungsgeschichte
```

```
--
-- Vers.   Datum   Bearbeiter   Änderungsgrund (lt.Antrag)
-- -----
-- 001    01.10.2004   Weber       Neuentwicklung
```

```
( npDvProgNr in DV_PROG_PARAM.DVPROGP_NR%Type
  -- Nummer des Programmparameters
  ,npValueN  in DV_PROG_PARAM_WERT.DVPROGPW_VALUE_N%Type
  -- Wert des Programmparameters
)
```

```
as
-- Variablen
...
-- Verarbeitung
begin
...
    -- Hier muss ein Update erfolgen, da ...
    update DV_PROG_PARAM_WERT
...
-- Fehlerbehandlung
exception
...

end Set_DvProgParamNumRAOFFL01;
...

-----
-- Funktionen
-----

...

end RaAuftrag01;
```

4 Inkrafttreten

Diese Arbeitsanweisung tritt mit der Veröffentlichung im Intranet in Kraft.

gez. Dr. Simon

gez. Aust