

Toad[®] User KONFERENZ 2 0 1 2



Code Qualität

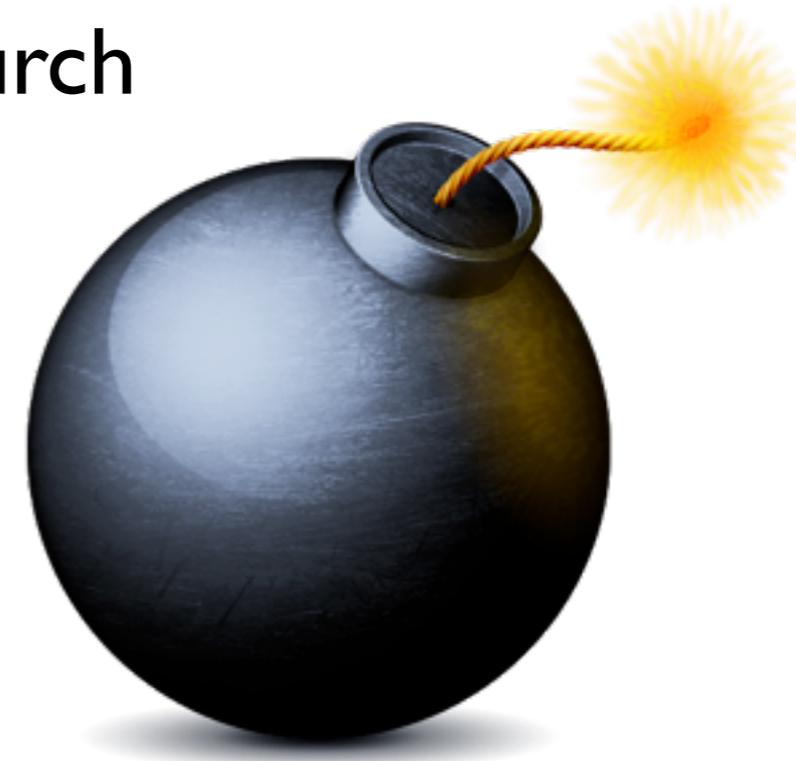
Welche Regeln sollte man
befolgen und warum?

Martin Friemel

www.webag.com

Zeitbomben

- Wie vermeide ich Probleme bei der PL/SQL-Entwicklung durch kluge Vorgaben an das Entwickler-Team?
- Mit welchen Regeln kann ich Schäden durch fehlerhafte PL/SQL-Module vermeiden?
- Was muss ich beachten, damit der PL/SQL-Code auch von neuen Mitarbeitern gewartet werden kann?
- **Wie wird mir TOAD dabei helfen?**



Beispiel-Situation

- Ich bin der neue DB-Entwickler in einem Oracle-Projekt
- Zum Start erstmal eine winzige Änderung in einem kleinen Modul
- Die Gutscheinverwaltung des Shop-Systems soll verbessert werden
- Kein Problem, der PL/SQL-Code ist nur wenige Zeilen lang

So sieht es aus ...

```
/* GS berechnen, KU 10/12/11 */
create or replace procedure gsvg is begin
for r in
(select * from personen where to_char(geburtstag, 'MM') =
to_char( sysdate, 'MM'))
loop
insert into gutscheine values (substr(sys_guid(),-8), r.persid,
(select least(greatest( ceil( (sum(einzelpreis)/100)/5 * 5), 5),100)
from positionen p, auftraege a where p.aufid = a.aufid
and a.persid = r.persid), sysdate+365);
end loop;
commit;
end;
/
```

So sieht es aus ...

```
/* GS berechnen, KU 10/12/11 */
create or replace procedure gsvg is begin
for r in
(select * from personen where to_char(geburtstag, 'MM') =
to_char( sysdate, 'MM'))
loop
insert into gutscheine values (substr(sys_guid(),-8), r.persid,
(select least(greatest( ceil( (sum(einzelpreis)/100)/5 * 5), 5),100)
from positionen p, auftraege a where p.aufid = a.aufid
and a.persid = r.persid), sysdate+365);
end loop;
commit;
end;
/
```

Kann mir TOAD dabei helfen?

Packages

- Verwenden Sie keine einzelnen Procedures und Functions, sondern grundsätzlich Packages
- Vorteile von Packages:
 - Modularität: Funktionen zusammenfassen, Interfaces werden in der Package Spezifikation beschrieben
 - Interne Funktionalitäten und Strukturen werden im Body verborgen
 - Globale Variablen und Strukturen bleiben für die Session erhalten
 - Geschwindigkeit: Das gesamte Package wird in den Speicher geladen

Kommentare

- Warum - Wann - Wer
- Einheitliches Datumsformat: **2012-10-01** statt **01/10/12**. So kann man z.B. danach suchen, wer im Oktober 2010 welche Änderungen durchgeführt hat.
- Namenskürzel auflösen, denn in einigen Jahren weiß niemand mehr, dass „**MF**“ Martin Friemel war
- Immer **--** statt **/* */** verwenden, dann steht **/* */** für Ad-Hoc-Ausblendungen zur Verfügung und es gibt keine Verwechslungen mit Optimizer-Hints.

Kommentare

(...)

```
FUNCTION get_key IS
BEGIN
  -- 10/11/12 KU: Datum und Guid aneinanderhaengen
  RETURN to_char(sysdate, 'yyyymmddhh24miss')
        || sys_guid();
END;
```

(...)

Das braucht niemand: Der Kommentar ist lediglich eine deutsche Übersetzung des PL/SQL-Codes. Wir wissen aber bereits, was `to_char`, `sysdate` und `sys_guid()` bedeuten.

Kommentare

(...)

```
-- Feinkonzept Kap. 2.3.1: Der Session-Key fuer die  
-- Warenkorb-Steuerung wird aus Datum/Zeit und einer  
-- GUID zusammengesetzt. (2012-08-04 M. Friemel)
```

```
FUNCTION get_shop_session_key IS  
BEGIN  
    RETURN to_char(sysdate, 'yyyymmddhh24miss')  
           || sys_guid();  
END;
```

(...)

Kommentare

- Beginnen Sie die Package Spezifikation immer mit einem Standard-Kommentarkopf für die Änderungshistorie
- Verlassen Sie sich nicht auf das Versionierungssystem. Wer weiß, ob der PL/SQL Sourcecode nicht in einigen Jahren an einen anderen Systempartner übergeben wird.
- Legen Sie im Toad ein Code-Template für Package Spezifikationen mit Kommentarkopf

Kommentare

- Verwenden Sie in Kommentaren keine Umlaute oder Sonderzeichen
- Geben Sie Regeln für die Versionsnummer des Packages vor, z.B:
 1. Stelle: Grundsätzliche Überarbeitung
 2. Stelle: Funktionale Änderungen
 3. Stelle: Bugfixes ohne funktionale Änderung

Package Spec

```
CREATE OR REPLACE
PACKAGE gutschein_verwalter IS
```

```
-- =====
-- Beschreibung:
--   In diesem Package werden Funktion zur Vergabe von Gutscheinen
--   angeboten.
--
-- Autor:
--   Martin Friemel (MF), mfriemel@webag.com
--   Karl Unentbärlich (KU), karl.unentbaerlich@webag.com
--
--
-- Datum      Version  Autor  Beschreibung
-- -----
-- 2012-09-10  2.0.0   MF     - Algorithmus neu implementiert: Eigene Funktionen
--                               fuer die Gutscheinberechnung und die Speicherung
--                               der Gutscheine
--
-- 2012-09-03  1.0.0   MF     - Erste Version als Package: Uebernahme der Procedure
--                               gsvg
-- =====
```

PL/SQL Datei-Ablage

- Code-Basis ist das Filesystem und am besten ein Versionierungssystem wie Subversion
- Entwickeln Sie niemals nur in der Datenbank, auch wenn Tools wie Toad dazu verführen
- Speichern Sie Spec und Body immer in getrennten Dateien
- Geben Sie Dateinamenskonventionen vor, z.B.:
 - package_name.**pks** für die Spezifikation
 - package_name.**pkb** für den Body
- Verwenden Sie für die Dateinamen nur Kleinbuchstaben. Unter Windows unerheblich, doch das Software-Verzeichnis kann einst auf Linux landen - und dann laufen die Installations-Skripte u.U. nicht mehr korrekt

Formatierung

- Wieviele Stellen einrücken?
- Tabs oder Blanks? Tabs in Blanks umwandeln?
- Schlüsselwörter groß oder klein schreiben?
- Leerzeilen vor und/oder nach SQL-Befehlen?
- Parameterleisten:
 - Klammer auf am Ende der Zeile oder in der nächsten Zeile?
 - Kommas vorne oder hinten?
 - Wie ausrichten?
 - Mehrere Parameter in einer Zeile?

Formatierung

- Wieviele Stellen einrücken?
- Tabs oder Blanks? Tabs in Blanks umwandeln?
- Schlüsselwörter groß oder klein schreiben?
- Leerzeilen vor und/oder nach SQL-Befehlen?
- Parameterleisten:
 - Klammer auf am Ende der Zeile oder in der nächsten Zeile?
 - Kommas vorne oder hinten?
 - Wie ausrichten?
 - Mehrere Parameter in einer Zeile?

Das ist ein Job für TOAD

Namenskonventionen

```
PROCEDURE geburtstags_guthaben IS
    bearbeitungs_datum DATE;
BEGIN

    bearbeitungs_datum := TRUNC( add_months(SYSDATE, -12) );

    UPDATE kunden
        SET guthaben = guthaben + 100
        WHERE registriert_am = bearbeitungs_datum;

END;
```

Namenskonventionen

☆ **Martin Friemel** 4. September 2012 16:12

An: Martin Friemel
Datenmodelländerung

Liebe Kollegen!

Ab diesem Release werden alle Datenänderungen in den Tabellen protokolliert. Dazu füllen Trigger zwei neue Spalten, die in jeder Tabelle angelegt wurden:

```
BEARBEITUNGS_DATUM DATE  
BEARBEITUNGS_USER VARCHAR2(30 CHAR)
```

Viele Grüße
Euer Datenmodell-Guru

--

www.webag.com

Düsseldorfer Str. 42
47051 Duisburg / Germany
Tel: +49 (0)203-2952550

Namenskonventionen

☆ **Martin Friemel**

4. September 2012 16:12

An: Martin Friemel

Datenmodelländerung

Liebe Kollegen!

Ab diesem Release werden alle Datenänderungen in den Tabellen protokolliert. Dazu füllen Trigger zwei neue Spalten, die in jeder Tabelle angelegt wurden:

BEARBEITUNGS_DATUM DATE
BEARBEITUNGS_USER VARCHAR2(30 CHAR)

Viele Grüße
Euer Datenmodell-Guru

--

Zeitbombe explodiert

- Dumm gelaufen:
Das Skript gibt plötzlich jedem Kunden ein Guthaben von 100 EUR, dessen Kundensatz vor einem Jahr geändert worden ist.
- Wo waren noch die Unterlagen für die IT-Haftpflichtversicherung?
- Das wäre mit vorgegebenen Prefixen nicht passiert!



Namenskonventionen

- Geben Sie Prefix-Konventionen vor, z.B.:
 - l_lokale_variable
 - g_globale_variable
 - c_constante
 - i_eingabe_parameter
 - o_ausgabe_parameter
 - io_ein_ausgabe_parameter
 - cur_cursor_name
 - r_record_name
 - e_exception_name

Namenskonventionen

```
PROCEDURE geburtstags_guthaben IS
    l_bearbeitungs_datum DATE;
BEGIN

    l_bearbeitungs_datum := TRUNC( add_months(SYSDATE, -12) );

    UPDATE kunden
        SET guthaben = guthaben + 100
        WHERE registriert_am = l_bearbeitungs_datum;

END;
```

So kann nichts schief gehen ...

Namenskonventionen

- Optional: Erweiterte Prefix-Vorgaben, um den Datentyp von Variablen zu benennen, z.B.:
 - **I_n**Numerische_Variable
 - **I_d**Datumswert
- Wie kann Toad bei den Prefix-Konventionen helfen?

%TYPE %ROWTYPE

- Deklarieren Sie Variablen mit %TYPE, wenn Sie sich auf Tabellenspalten beziehen
- Verwenden Sie wenn möglich %ROWTYPE, wenn es um Sätze einer Tabelle geht
- Es drohen sonst ähnliche Zeitbomben wie beim Prefix-Beispiel

UTF8 Zeitbomben

- Deklarieren Sie VARCHAR2-Variablen immer mit „CHAR“, also z.B.: `l_nachname VARCHAR2(40 CHAR)`
- Die Anzahl Bytes sind Ihnen i.d.R. egal: Sie wollen 40 Buchstaben speichern!
- Alle Tests mit den Namen „Schmidt“, „Müller“ und „Meier“ waren erfolgreich, aber nach der Inbetriebnahme wurden solche Namen gespeichert: **Мануэль Нойер Роман Вайденфеллер**
- Es gibt DB-Parameter wie `NLS_LENGTH_SEMANTICS`, aber wir wollen ja Zeitbomben vermeiden!

Interface-Parameter

- Verwenden Sie immer die Syntax mit Angabe der Parameternamen

```
l_gutschein_code :=  
    erzeuge_gutschein (  
        i_persid      => pers_rec.persid,  
        i_betrag      => l_betrag,  
        i_guelteig_bis => SYSDATE+365  
    );
```

- Verlassen Sie sich nie auf die Reihenfolge der Parameter, indem Sie die Werte einfach mit Komma aneinander hängen
- Ausnahme: SQL-Funktionen und Aufrufe innerhalb von SQL-Statements

Beispiel von Oracle

```
select substr('abcdefghij', 1, 3)
from dual;
```

Ergebnis:
abc

Und jetzt die CLOB-Variante:

```
declare
  l_text clob := 'abcdefghij';
begin
  dbms_output.put_line(dbms_lob.substr(l_text, 1, 3));
end;
```

Ergebnis:
c

Beispiel von Oracle

Oracle hat in der DBMS_LOB.SUBSTR-Implementation die beiden Parameter Amount und Offset vertauscht:

```
DBMS_LOB.SUBSTR (  
  lob_loc      IN      CLOB      CHARACTER SET ANY_CS,  
  amount      IN      INTEGER := 32767,  
  offset      IN      INTEGER := 1  
)  
RETURN VARCHAR2 CHARACTER SET lob_loc%CHARSET;
```

Beispiel von Oracle

Sorgfältig formuliertes Interface:

```
declare
  l_text    clob := 'abcdefghij';
begin
  dbms_output.put_line (
    dbms_lob.substr (
      lob_loc  => l_text,
      offset   => 1,
      amount   => 3
    )
  );
end;
/
```

Ergebnis:
abc

Zeitbombe

Function Interface:

```
FUNCTION erzeuge_gutschein (  
    i_persid          IN NUMBER,  
    i_betrag         IN NUMBER,  
    i_gueltig_bis    IN DATE DEFAULT SYSDATE+90  
)  
RETURN VARCHAR2;
```

Schlampiger Aufruf ohne Parameter-Namen:

```
l_gutschein_code :=  
    erzeuge_gutschein (  
        pers_rec.persid, l_betrag, SYSDATE+31  
    );
```

Interface-Parameter

Neue Version der Function:

```
FUNCTION erzeuge_gutschein (  
    i_persid          IN NUMBER,  
    i_betrag         IN NUMBER,  
    i_gueltig_ab     IN DATE DEFAULT SYSDATE,  
    i_gueltig_bis    IN DATE DEFAULT SYSDATE+90  
)  
RETURN VARCHAR2;
```

Der schlampige Aufruf läuft problemlos weiter ...

```
l_gutschein_code :=  
    erzeuge_gutschein (  
        pers_rec.persid, l_betrag, SYSDATE+31  
    );
```

Interface-Parameter

... nur leider sind die Gutscheine
nicht wie geplant
ab sofort für 31 Tage gültig,
sondern

sie werden erst in 31 Tagen gültig.



INSERT-Statements

Auch in INSERT-Statements muss sauber formuliert werden:

Falsch:

```
INSERT INTO person  
VALUES (l_pers_id, l_vorname, l_nachname);
```

Richtig:

```
INSERT INTO person (  
    pers_id,  
    vorname,  
    nachname  
)  
VALUES (  
    l_pers_id,  
    l_vorname,  
    l_nachname  
);
```



Protokollierung

- Erstellen Sie ein zentrales Package mit Logger-Funktionen
- Alle PL/SQL-Module sollen ihre Aktivitäten, Fehler und Abbrüche dort protokollieren
- Die Protokoll-Aufrufe sollen mit einem Log-Level versehen werden. Die Levels werden als Konstanten vordefiniert
- Vorteil: Innerhalb des Logger-Packages können Sie an zentraler entscheiden, wohin Sie die Protokolle schreiben wollen und wie ausführlich sie auf den gewählten Kanälen senden wollen

Exception-Handler

- Exception-Handler sollen nicht verhindern, dass ein Package abbricht, sondern sie geben die Möglichkeit, vor dem Abbruch zu reagieren
- Alle Exceptions werden in den Exception-Handlern mit dem Logger-Package protokolliert

Exception-Handler

(...)

```
EXCEPTION
  WHEN others THEN

    ROLLBACK;

    logger.write_log (
      i_module      => $$plsql_unit,
      i_message     => SQLERRM,
      i_log_level   => logger.c_log_level_error
    );

    RAISE; -- Fehler zum aufrufenden Programm hochreichen

END gutscheinvergabe;
```

Exception-Handler

Deklarieren Sie eigene Exceptions für fachliche Abbruch-Situationen und vermeiden Sie, Standard-Exceptions zu missbrauchen

Schlimm:

```
-- Kunde hat noch keinen Vertrag:  
raise no_data_found;
```

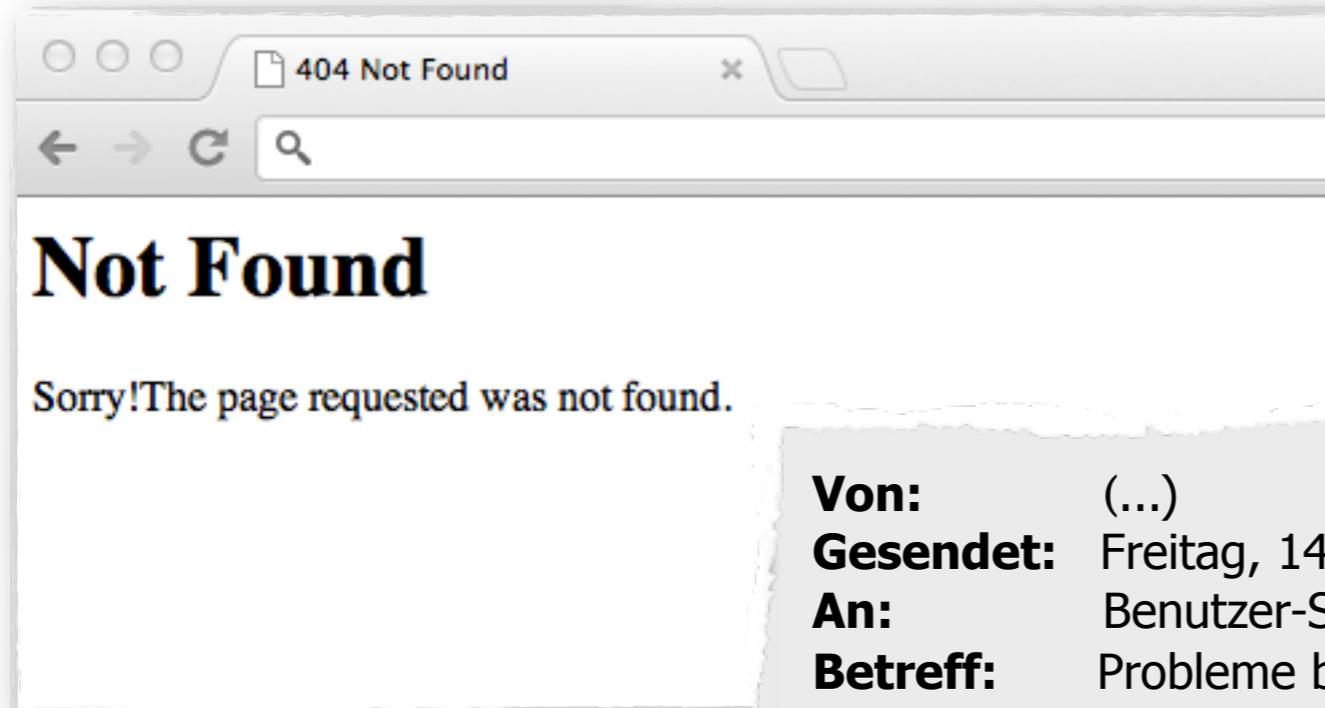
```
-- Kunde hat bereits einen Gutschein:  
raise dup_val_on_index;
```

Exception-Handler

Besser:

```
e_vertrag_fehlt          EXCEPTION;  
e_doppelter_gutschein   EXCEPTION;  
  
PRAGMA EXCEPTION_INIT (e_vertrag_fehlt, -20030);  
PRAGMA EXCEPTION_INIT (e_doppelter_gutschein, -20031);  
  
(...)  
  
-- Kunde hat noch keinen Vertrag:  
raise e_vertrag_fehlt;  
  
-- Kunde hat bereits einen Gutschein:  
raise e_doppelter_gutschein;
```

Dynamische Webseite



Von: (...)
Gesendet: Freitag, 14. September 2012 08:52
An: Benutzer-Service IT
Betreff: Probleme bei (...)

Hallo und guten Morgen zusammen,

bereits seit einiger Zeit versuche ich die (...) zu übernehmen. Leider erhalte ich immer obige Fehlermeldung.

Bitte um Hilfe.
Danke und Gruß
(...)

Exception-Handler

Vermengen Sie keine fachlichen mit technischen Exceptions:

```
BEGIN
  ...

  -- Kundendaten sammeln
  SELECT ... INTO

  -- Kunde aendern
  UPDATE ...

  ...

EXCEPTION
  WHEN no_data_found THEN
    ...
    INSERT INTO ... -- Neuer Kunde. Jetzt anlegen.

  WHEN others THEN
    ...

END;
```

Exception-Handler

Besser: Eigener Exception-Handler für die fachliche Exception

```
BEGIN
  ...

  BEGIN
    -- Kundendaten sammeln
    SELECT ... INTO

    -- Kunde aendern
    UPDATE ...

  EXCEPTION
    WHEN no_data_found THEN
      ...
      INSERT INTO ... -- Neuer Kunde. Jetzt anlegen.
  END;
  ...

EXCEPTION
  WHEN others THEN
    ...

END;
```

Code Qualität

Alle Zeitbomben entschärft



Vielen Dank
für's Zuhören!

Martin Friemel
mfriemel@webag.com