Gespeicherte Prozeduren

Lehrveranstaltung Datenbanktechnologien

Prof. Dr. Ingo Claßen Prof. Dr. Martin Kempa

Hochschule für Technik und Wirtschaft Berlin

Blöcke

- Java:

```
try {
  // Variablendeklaration
  // und Anweisungen
}
catch (Exception exp) {
  // Ausnahmebehandlung
}
```

- PL/SQL:

```
declare
  -- Variablendeklaration
begin
  -- Anweisungen
exception
  -- Ausnahmebehandlung
end;
```

Konstanten und Variablen

Konstanten-Definition

- Java:
```
private static final int ATTENDEE_LIMIT = 44;
```
- PL/SQL:
```
c_attendee_limit constant integer := 44;
```

Variablen-Deklaration

- Java:
```
int v_matr_nr;
BigDecimal v_grade;
String v_name;
boolean v_has_grade;
```
- PL/SQL:
```
-- SQL-Datentyp
v_matr_nr number(6);
v_grade number(2,1);
v_name varchar(25);
-- nur PL/SQL-Datentyp
v_has_grade boolean;
```

Zuweisungen

Einfache Zuweisungen

- Java:

```
{
  int v_matr_nr = 50101;
  Date v_due_date;

  v_due_date = new Date();
}
```

- PL/SQL:

```
declare
  -- Variable mit
  -- Initialisierung
  v_matr_nr number(6) := 50101;
  v_due_date date;
begin
  v_due_date := sysdate;
end;
```

Zuweisung über SQL-Anweisung

```
declare
  v_name varchar(25);
begin
  -- SQL-Anweisung muss degenerierte Tabelle liefern
  select NAME into v_name
  from STUDENT
  where MATRNR = v_matr_nr;
end;
```

Komplexe Typen

Definition eines Record-Typs

- Java

```
class t_stud {
  int matrnr = 50101;
  String name;
}
```

- PL/SQL

```
type t_stud is record (
  matrnr number(6) not null := 50101,
  name varchar(200)
);
```

Variablendeklaration und Zuweisung

- Java

```
{
  t_stud v_stud = new t_stud();
  v_stud.name = "Svenson";
}
```

- PL/SQL

```
declare
  v_stud t_stud;
begin
  v_stud.name := 'Svenson';
end;
```

Assoziierte Typen

```
declare
  v_matr_nr number(6) := 50101;
  -- Variablendeklaration mit assoziiertem Typ
  v_name STUDENT.NAME%type;
  -- Variablendeklaration mit assoziiertem Record-Typ
  v_student STUDENT%rowtype;
begin
  v_student.vorname := 'antonio';       -- Einfache Zuweisungen

  -- Zuweisung über SQL-Anweisung
  -- (die Anweisung darf nur eine Zeile liefern)
  select * into v_student
  from STUDENT
  where MATRNR = v_matr_nr;
end;
```

- Assoziierte Typen gibt es in Java nicht

Kollektionen

Varray (entspricht Array in Java)

- Java

```
Integer[] v_numbers = new Integer[5];
```

- PL/SQL

```
type t_array is varray(5) of integer; -- Typdefinition

v_numbers t_array;                    -- Variablendeklaration
```

Nested Tables (entspricht Set oder Bag in Java)

- Java

```
Set<String> v_names = new HashSet<String>();
```

- PL/SQL

```
type t_set is table of varchar(30); -- Typdefinition

v_names t_set;                      -- Variablendeklaration
```

Kollektionen

Associative Array (entspricht Map in Java)

- Java

```
Map<Integer, String> v_names_by_id =
  new HashMap<Integer, String>()
Map<String, Integer> v_ids_by_name =
  new HashMap<String, Integer>()
```

- PL/SQL

```
-- Typdefinition
type t_map_by_int is table of varchar(32) index by integer;
type t_map_by_string is table of integer index by varchar(10);

-- Variablendeklaration
v_names_by_id t_map_by_int;
v_ids_by_name t_map_by_string;
```

Laden von Daten in Tabellen (bulk loading)

```
declare
  type t_studiengang_set is table of STUDIENGANG%rowtype;
  v_studiengaenge t_studiengang_set;
begin
  select * bulk collect into v_studiengaenge
  from STUDIENGANG;

  for v_i in 1 .. v_studiengaenge.count
  loop
    dbms_output.put_line(v_studiengaenge(v_i).bezeichnung);
  end loop;
end;
```

Identifikation geänderter/gelöschter Datensätze

```
declare
  type t_int_set is table of integer;
  v_matr_nrs t_int_set;
begin
  delete from EINSCHREIBUNG
  where STUDIENGANG = 'AI'
  returning MATRIKELNR bulk collect INTO v_matr_nrs;

  for v_i in 1 .. v_matr_nrs.count
  loop
    dbms_output.put_line(v_matr_nrs(v_i));
  end loop;
end;
```

If-Anweisung ohne bzw. mit einer Alternative

Fallunterscheidung ohne Alternative

- Java:

```
if (v_grade == 'a') {
  System.out.println(
    "excellent");
}
```

- PL/SQL:

```
if v_grade = 'a' then
  dbms_output.put_line(
    'excellent');
end if;
```

Fallunterscheidung mit einer Alternative

- Java:

```
if (v_grade == 'a') {
  System.out.println(
    "excellent");
}
else {
  System.out.println(
    "not excellent");
}
```

- PL/SQL:

```
if v_grade = 'a' then
  dbms_output.put_line(
    'excellent');
else
  dbms_output.put_line(
    'not excellent');
end if;
```

Verkettete If-Anweisungen

- Java:

```
if (v_grade = 'a') {
  System.out.println(
    "excellent");
}
else if (v_grade = 'b') {
  System.out.println(
    "very good");
}
else if (v_grade = 'c') {
  System.out.println("good");
}
else if (v_grade = 'd') {
  System.out.println("fair");
}
else if (v_grade = 'f') {
  System.out.println("poor");
}
else {
  System.out.println(
    "no such grade");
}
```

- PL/SQL:

```
if v_grade = 'a' then
  dbms_output.put_line(
    'excellent');
elsif v_grade = 'b' then
  dbms_output.put_line(
    'very good');
elsif v_grade = 'c' then
  dbms_output.put_line(
    'good');
elsif v_grade = 'd' then
  dbms_output. put_line(
    'fair');
elsif v_grade = 'f' then
  dbms_output.put_line(
    'poor');
else
  dbms_output.put_line(
    'no such grade');
end if;
```

Case-Anweisung

- Java:

```
switch (v_grade) {
  case 'a':
    System.out.println("excellent");
    break;
  case 'b':
    System.out.println("very good");
    break;
  case 'c':
    System.out.println("good");
    break;
  case 'd':
    System.out.println("fair");
    break;
  case 'f':
    System.out.println("poor");
    break;
  default:
    System.out.println("no such grade");
}
```

Case-Anweisung

- PL/SQL:

```
case v_grade
  when 'a' then dbms_output.put_line('excellent');
  when 'b' then dbms_output.put_line('very good');
  when 'c' then dbms_output.put_line('good');
  when 'd' then dbms_output.put_line('fair');
  when 'f' then dbms_output.put_line('poor');
  else dbms_output.put_line('no such grade');
end case;
```

- PL/SQL:

```
case
  when v_grade = 'a' then dbms_output.put_line('excellent');
  when v_grade = 'b' then dbms_output.put_line('very good');
  when v_grade = 'c' then dbms_output.put_line('good');
  when v_grade = 'd' then dbms_output.put_line('fair');
  when v_grade = 'f' then dbms_output.put_line('poor');
  else dbms_output.put_line('no such grade');
end case;
```

While-Anweisung

- Java:

```
while (v_i <= 3) {
  System.out.println(String.valueOf(v_i));
  v_i = v_i + 1;
}
```

- PL/SQL:

```
while v_i <= 3 loop
  dbms_output.put_line(to_char(v_i));
  v_i := v_i + 1
end loop;
```

For-Anweisung

- Java:

```
for (v_i = 1; v_i < 3; v_i = v_i + 1) {
  System.out.println(String.valueOf(v_i));
}
```

- PL/SQL:

```
for v_i in 1..3 loop
  dbms_output.put_line(to_char(v_i));
end loop;
```

Loop-Anweisung

- Java:

```
do {
  System.out.println("inside loop");
  v_i = v_i + 1;
}
while (v_i <= 3);
```

- PL/SQL:

```
loop
  dbms_output.put_line('inside loop');
  v_i := v_i + 1
  if v_i > 3 then
    exit;
  end if;
  -- Alternative
  -- exit when v_i > 3;
end loop;
```

Loop-Anweisung mit Continue

- Java:

```
while (true) {
  // nach continue geht es hier weiter
  System.out.println("inside loop");
  v_i = v_i + 1;
  if (v_i < 3) continue;
  System.out.println("inside loop, after continue");
  if (v_i == 5) break;
}
```

- PL/SQL:

```
loop
  -- nach continue geht es hier weiter
  dbms_output.put_line('inside loop');
  v_i := v_i + 1;
  continue when v_i < 3;
  dbms_output.put_line('inside loop, after continue');
  exit when v_i = 5;
end loop;
```

Schleifenbeispiel

- Definition einer Tabellen

```
create table TEMP_EMAIL(
  ID integer,
  EMAIL varchar(50)
);
```

- Befüllung der Tabelle

```
declare
  v_stud_count integer;
begin
  select count(MATRNR) into v_stud_count
  from STUDENT;

  for v_i in 1..v_stud_count loop
    insert into TEMP_EMAIL
    values (v_i, 'to be added later');
  end loop;
end;
```

Fehlerbehandlung durch Ausnahmen

- Java

```
try {
  int v_stud_sum = 973;
  int v_course_sum = 0;
  int v_course_avg;
  // Division durch 0
  v_course_avg = v_stud_sum / v_course_sum;
}
catch (ArithmeticException exp) {
  ... // Fehler bei Division durch 0 behandeln
}
catch (Exception exp) {
  ... // alle anderen Fehler behandeln
}
```

Fehlerbehandlung durch Ausnahmen

- PL/SQL

```
declare
  v_stud_sum integer := 973;
  v_course_sum integer := 0;
  v_course_avg integer;
begin
  -- Division durch 0
  v_course_avg := v_stud_sum / v_course_sum;
exception
  when zero_divide then
    ... -- Fehler bei Division durch 0 behandeln
  when others then
    ... -- alle anderen Fehler behandeln
end;
```

Vordefinierte Ausnahmen (Auswahl)

Ausnahme	**Auslösung durch**
DUP_VAL_ON_INDEX	Einfügen eines Datensatzes mit einem Primärschlüssel der bereits existiert
NO_DATA_FOUND	Select-Anweisung liefert keine Daten
TOO_MANY_ROWS	Select-Anweisung, die nur eine Zeile liefern darf, liefert mehrere Zeilen
VALUE_ERROR	Fehler bei Datenkonversion, z. B. Umwandlung einer Zeichenkette, die Buchstaben enthält, in eine Zahl

Nicht behandelte Ausnahmen werden weitergeleitet

```
declare
  v_stud_sum integer := 973;
  v_course_sum integer := 0;
  v_course_avg integer;
begin
  -- Anweisungen, die Daten aus einer Tabelle laden
  ...
  -- Division durch 0
  v_course_avg := v_stud_sum / v_course_sum;
exception
  when no_data_found then
    -- Fehler bei no_data_found behandeln
    ...
    -- Fehler bei Division durch 0 wird nicht behandelt
    -- und an die aufrufende Einheit weiter geleitet
end;
```

Wiederauslösung behandelter Ausnahmen

- Java

```
try {
  ... // Anweisungen, die Exceptions auslösen können
}
catch (no_data_found exp) {
    ...        // Fehler bei no_data_found behandeln
    throw exp; // behandelte Ausnahme wieder auslösen
}
```

- PL/SQL

```
begin
  ... -- Anweisungen, die Exceptions auslösen können
exception
  when no_data_found then
    ...    -- Fehler bei no_data_found behandeln
    raise; -- behandelte Ausnahme wieder auslösen
end;
```

Selbstdefinierte Ausnahmen

- Exception-Klasse in Java

```
public class exc_past_due extends Exception {
  ...
}
```

- Verwendung in Java

```
try {
  Date v_due_date = DateUtils.addDays(new Date(),-1);
  Date v_todays_date = new Date();

  if (v_due_date < v_todays_date) {
    throw new exc_past_due();
  }
}
catch (exc_past_due exp) [
    ... // Fehler behandeln
}
```

Selbstdefinierte Ausnahmen

- PL/SQL

```
declare
  exc_past_due exception;
  v_due_date date := sysdate - 1;
  v_todays_date date := sysdate;
begin
  if v_due_date < v_todays_date then
    raise exc_past_due;
  end if;
exception
  when exc_past_due then
    ... -- Fehler behandeln
end;
```

Selbstdefinierte Ausnahmen mit Fehlernummer verbinden

- Oracle definiert nur für ausgewählte Fehler Ausnahmen.
- Das Pragma `EXCEPTION_INIT` ermöglicht die Zuweisung von sprechenden Namen zu Fehlernummern.
- Verwendung in When-Klausel möglich.
- Beispiel

```
declare
  exc_deadlock_detected exception;
  pragma exception_init(exc_deadlock_detected, -60);
begin
  ... -- Operation, die einen ora-00060 Fehler auslöst
exception
  when exc_deadlock_detected then
    ... -- Fehler behandeln
end;
```

Definition eigener Fehlermeldungen

- Beispiel

```
declare
  v_num_student integer;
begin
  select count(*) into v_num_student
  from STUDENT;
  if v_num_student < 1000 then
    -- Prozedur löst eine Ausnahme aus
    -- erlaubte Nummern: -20000 bis -20999
    raise_application_error(-20101,
      'expecting at least 1000 students');
  end if;
end;
```

Prozeduren

- Java

```
public void remove_student(int p_matr_nr) {
  // Code für delete from STUDENT
  //          where STUDENT.MATRNR = p_matr_nr;
}
```

- PL/SQL

```
create procedure remove_student(p_matr_nr integer) as
begin
  delete from STUDENT
  where STUDENT.MATRNR = p_matr_nr;
end remove_student;
```

Funktionen

- Java

```
public BigDecimal get_grade(int p_matr_nr, int p_veranst_nr) {
  ... // Code für select NOTE from BEWERTUNG
      //          where MATRNR = p_matr_nr
      //            and VERANSTARLTUNGSNR = p_veranst_nr;
  BigDecimal v_grade = ...
  return v_grade;
}
```

- PL/SQL

```
create function get_grade(p_matr_nr integer,
  p_veranst_nr integer) return number(2,1) as
  v_grade number(2,1);
begin
  select NOTE into v_grade
  from BEWERTUNG
  where MATRNR = p_matr_nr
    and VERANSTARLTUNGSNR = p_veranst_nr;
  return(v_grade);
end get_grade;
```

Aufrufe von Routinen

- Java

```
{
  int v_matr_nr = 50101;
  int v_veranst_nr = 99841;
  BigDecimal v_grade = get_grade(v_matr_nr, v_veranst_nr);
  remove_student(v_matr_nr);
}
```

- PL/SQL

```
declare
  v_matr_nr integer := 50101;
  v_veranst_nr integer := 99841;
  v_grade number(2,1);
begin
  -- Parameterübergabe nach Position
  v_grade := get_grade(v_matr_nr, p_veranst_nr);
  remove_student(v_matr_nr);
end;
```

Default-Werte und benannte Parameter

- Funktion mit Default-Wert

```
create function get_grade(p_matr_nr integer,
  p_veranst_nr integer := 99841) return number(2,1) as
  ...
end get_grade;
```

- Aufrufe mit benannten Parametern

```
declare
  v_matr_nr integer := 50101;
  v_veranst_nr integer := 99841;
  v_grade number(2,1);
begin
  -- Ausnutzung des Default-Werts
  v_grade := get_grade(p_matr_nr => v_matr_nr);

  -- Benannte Parameter, Reihenfolge unerheblich
  v_grade := get_grade(p_veranst_nr => v_veranst_nr,
    p_matr_nr => v_matr_nr);

  -- Gemischte Notation
  v_grade := get_grade(v_matr_nr, p_veranst_nr => v_veranst_nr);
end;
```

Parameterdeklaration

	IN	OUT	INOUT
Funktion	Wertübergabe in Prozedur	Wertübergabe aus Prozedur	beides
Verhalten	wie eine Konstante	wie eine nicht initialisierte Variable	wie eine initialisierte Variable
Standardwert	möglich	nicht möglich	nicht möglich
Wertzuweisung in Prozedur	nicht möglich	erforderlich	möglich
Aufrufparameter	Initialisierte Variable, Konstante oder Ausdruck	muss Variable sein	muss Variable sein

Prozedur mit OUT-Parameter

- Definition

```
create procedure calc_avg(p_matr_nr in integer,
  p_avg_grade out number(2,1)) as
begin
  ...
end calc_avg;
```

- Aufruf

```
declare
  v_matr_nr integer := 50101;
  v_avg_grade number(2,1);
begin
  calc_avg(v_matr_nr, v_avg_grade);
  ...
end;
```

Paketspezifikation als Schnittstelle

- Java

```
public interface bewertung_service {

  void erstelle_bewertung(int p_matr_nr, int p_veranst_nr,
    BigDecimal p_note);

  BigDecimal finde_bewertung(int p_matr_nr, int p_veranst_nr);
}
```

- PL/SQL

```
create or replace package bewertung_service as

  procedure erstelle_bewertung(p_matr_nr integer,
    p_veranst_nr integer, p_note integer);

  function finde_bewertung(p_matr_nr integer,
    p_veranst_nr integer) return integer;

  -- Deklaration von globalen Variablen, Typen,
  -- Ausnahmen sowie Cursorn
end bewertung_service;
```

Paketkörper als Implementierung

- Java

```
public class bewertung_service_impl
  implements bewertung_service {

  public void erstelle_bewertung(int p_matr_nr,
    int p_veranst_nr, BigDecimal p_note) {
    ...
  }
  ...
}
```

- PL/SQL

```
create or replace package body bewertung_service as

  procedure erstelle_bewertung(p_matr_nr integer,
    p_veranst_nr integer, p_note integer) as
    ...
  end erstelle_bewertung;
  ...
end bewertung_service;
```

Cursor

```
declare
  ...
  -- Cursor deklarieren
  cursor cur_students is
    select NAME, MATRNR from STUDENT where STUDIENGANG = 'WI';
begin
  -- Cursor öffnen
  open cur_students;

  loop
    -- Daten in Variablen übertragen
    fetch cur_students into v_name, v_matr_nr;
    -- Überprüfung, ob (noch) Daten vorhanden
    exit when cur_students%notfound;
    ...
  end loop;

  -- Cursor schließen
  close cur_students;
end;
```

Cursor-Attribute

Attribut	**Bedeutung**
%FOUND	Datensatz wurde aus dem Coursor geholt
%NOTFOUND	Kein Datensatz mehr vorhanden
%ISOPEN	Cursor ist geöffnet
%ROWCOUNT	Anzahl bisher geholter Datensätze

Anwendung	%FOUND	%NOTFOUND	%ISOPEN	%ROWCOUNT
vor open	Exception*	Exception	false	Exception
nach open	null	null	true	0
nach erstem fetch	true	false	true	1
nach letztem fetch	false	true	true	n
nach close	Exception	Exception	false	Exception

*ORA-01001: Ungültiger Cursor

Cursor mit Parameter und Ergebnisspezifikation

- Cursor-Definition

```
cursor cur_by_title(p_title_filter VERANSTALTUNG.TITEL%type)
  return VERANSTALTUNG%rowtype
is
  select *
  from VERANSTALTUNG
  where TITEL like upper(p_title_filter);
```

- Argumentübergabe bei

```
...
open cur_by_title('DB%');
...
```

Cursor in Paketen

- Schnittstelle

```
create or replace package veranstaltung_service is
  -- Cursor deklarieren
  cursor cur_by_title(p_title_filter VERANSTALTUNG.TITEL%type)
    return VERANSTALTUNG%rowtype;
end veranstaltung_service;
```

- Implementierung

```
create or replace package body veranstaltung_service is
  -- Cursor definieren
  cursor cur_by_title(p_title_filter VERANSTALTUNG.TITEL%type)
    return VERANSTALTUNG%rowtype
  is
    select *
    from VERANSTALTUNG
    where TITEL like upper(p_title_filter);
end veranstaltung_service;
```

Aufgabe Datumstabelle

- Entwickeln Sie eine Prozedur, die eine Datumstabelle füllt
  - ausgehend vom maximalen Datum in der Tabelle sollen die nächsten p_anzahl Tage hinzukommen
  - ist die Tabelle leer, wird das aktuelle Datum verwendet
- Tabelle

Tag	
TagNr	Datum
1	13.11.2009
2	14.11.2009

- Signatur

```
create procedure erzeuge_tage(p_anzahl integer) as
  ...
begin
  ... -- Füllen der Tabelle TAG
end erzeuge_tage;
```

- Hinweise
  - sysdate liefert das aktuelle Systemdatum
  - Operator + addiert Tage zu einem Datum (z. B. sysdate + 7)

Trigger-Konzepte

- Trigger sind Prozeduren, die als Reaktion auf Ereignisse in der Datenbank ausgeführt werden
- Ereignisse: Insert-, Delete- und Update-Anweisungen
- Zeitpunkte der Ausführung: Before, After, Instead-Of
- Ausführungsbereich
  - Gesamte Anweisung (Statement-Trigger)
  - Jede Zeile (Row-Level-Trigger)
- Ausführung von
  - Prozeduren
  - anonymen Prozeduren (PL/SQL-Blöcken)

Anwendungsbeispiele für Trigger

- Einhaltung der Datenintegrität
- Implementierung von Geschäftslogik
- Prüfen von Sicherheitsconstraints
- Protokollieren von Datenänderungen
- Replikationsmechanismen
- Aktualisieren von Sichten
- Erfassung von Statistiken für Tabellennutzungen

Beispiel: Archivierung einer Veranstaltung

- Trigger

```
create or replace trigger trg_archiviere_veranstaltung
after delete on VERANSTALTUNG
for each row
call archiviere_veranstaltung(:old.bezeichnung, :old.dozent)
```

- Tabelle

```
create table VERANSTALTUNG_ARCHIV(
  TITEL varchar(25),
  DOZENT varchar(50),
  ARCHIVIERUNGSDATUM date,
  BEARBEITER varchar(25)
);
```

Beispiel: Archivierung einer Veranstaltung

- Beispiel

```
create procedure archiviere_veranstaltung(
  p_titel varchar(25), p_personal_nr integer) as
  v_name varchar(50);
begin
  select NAME into v_name
  from DOZENT D
  where D.PERSONALNR = p_personal_nr;

  insert into VERANSTALTUNG_ARCHIV(TITEL, DOZENT,
    ARCHIVIERUNGSDATUM, BEARBEITER)
  values (p_titel, v_name, sysdate, user);
end archiviere_veranstaltung;
```

Beispiel: Automatische Erzeugung eines Primärschlüssels

- Definition einer Sequenz für die automatische Erzeugung von Werten

```
create sequence S_MATRIKELNR;
```

- Der Trigger verwendet die Sequenz, um Werte an die Spalte MATRIKELNR des neuen Datensatzes zuzuweisen

```
create or replace trigger trg_student_insert
before insert on STUDENT
for each row
call next_matrikelnr(:new.MATRIKELNR)
create or replace procedure next_matrikelnr(
  p_matrikelnr out integer) as
begin
  p_matrikelnr := S_MATRIKELNR.nextval;
end next_matrikelnr;
```

- Die Tabelle DUAL ist eine Dummy-Tabelle mit einer Zeile und einer Spalte. Der Aufruf `S_MATRIKELNR.nextval` zählt die Sequenz hoch

Bedingte Ausführung

- Beispiel

```
create or replace trigger trg_note
after update of NOTE on BEWERTUNG
for each row
when (new.NOTE != old.NOTE)
  -- Protokolliere Änderung
call log_change(:old.MATRIKELNR,:new.NOTE,:old.NOTE)
```

Auslösung des selben Triggers durch verschiedene Ereignisse

- Beispiel

```
create or replace procedure veranstaltung_audit() as
begin
  if inserting then
    ... -- Aktion beim Einfügen
  elsif updating then
    ... -- Aktion beim Ändern
  elsif deleting then
    ... -- Aktion beim Löschen
  end if;
end veranstaltung_audit;
```

- Beispiel

```
create or replace trigger trg_veranstaltung_audit
after insert or update or delete on VERANSTALTUNG
for each row
call veranstaltung_audit
```

Problem mutierender Tabellen

- Problem: Ein Trigger liest Daten der auslösenden (sich ändernden) Tabelle
- Prozedur liest aus Tabelle STUDENT

```
create procedure put_anzahl_studenten() as
  v_student_anzahl integer;
begin
  select count(*) into v_student_anzahl
  from STUDENT;
  dbms_output.put_line(v_student_anzahl);
end put_anzahl_studenten;
```

- Trigger reagiert auf Ereignisse in STUDENT

```
create or replace trigger trg_student_anzahl
after insert on STUDENT
for each row
call put_anzahl_studenten
```

- Beim Einfügen eines Datensatzes kommt es zu einem Fehler

```
insert into STUDENT values (50101, 'Sven', 'Svenson');
```

Lösung mutierender Tabellen

- Lösungsmöglichkeiten
  - Statement-Trigger verwenden
  - Workaround mit temporären Tabellen oder Packages
  - Autonome Transaktionen
  - Compound Trigger
- Beispiel mit einem Compound Trigger

```
create or replace trigger trg_student_anzahl
for insert on STUDENT
compound trigger
  v_student_anzahl integer;
  ...
```

Lösung mutierender Tabellen

```
  ...
  -- hier wird Tabelle STUDENT gelesen
  before statement is
  begin
    select count(*) into v_student_anzahl from STUDENT;
  end before statement;

  -- der zwischengespeicherte Wert wird pro Zeile erhöht
  after each row is
  begin
    v_student_anzahl := v_student_anzahl + 1;
  end after each row;

  -- zum Schluss erfolgt die Weiterverarbeitung
  after statement is
  begin
    dbms_output.put_line(v_student_anzahl);
  end after statement;
end trg_student_anzahl;
```