6.1.3 Beispiel für einen anonymen PL/SQL-Block

```
-- Deklarationsblock
declare
   cursor c is select gehalt from angestellter
               for update of gehalt;
   v_gehalt angestellter.gehalt%type;

-- Ausführungsblock
begin
   -- Cursor öffnen
   open c;

   -- erste Zeile lesen
   fetch c into v_gehalt;

   -- alle Zeilen der Ergebnistabelle lesen
   while c%found loop
    if   v_gehalt > 10000
    then update angestellter set gehalt = gehalt * 1.03
                where current of c;
    else update angestellter set gehalt = gehalt * 1.02
                where current of c;
    end if;
    fetch c into v_gehalt;
   end loop;

end;
/
```

Tabelle ANGESTELLTER

PER_NR	NAME	...	GEHALT
1001001	Reiter	...	15.900
1001002	Kirsch		15.900
1001003	Daum		10.100
1004008	Müller		16.200
1005013	Balzert		9.800
1005112	Weiland		7.900
9001011	Schmitt		4.200
...	...		...

GEHALT = 16.377

Der PL/SQL-Block soll bei Ausführung die Gehälter erhöhen, und zwar
- alle Gehälter ≥ 10.000,-- um 3 %,
- alle Gehälter < 10.000,-- um 2 %.

1. **Cursorstruktur**: Spalte GEHALT
2. **PL/SQL-Variable**: v-gehalt
3. Der Cursor wird als **Updatecursor** deklariert.
4. Beim **open** wird das select ausgeführt.
5. Die bool‘sche Variable **c%found** ist true, falls die vorhergehende fetch-Anweisung Daten gelesen hat.
6. Mit **... current of c** wird der gerade gelesene Datensatz aktualisiert.

6.1.3 Ergänzung zum Cursorkonzept – der „einzeilige“ Cursor (ESQL)

- Zum Lesen und Weiterverarbeiten von Tabellendaten deklariert man i.d.R. einen Cursor.
- Dies ist jedoch nicht notwendig wenn man weiss, dass das entsprechende SELECT-Statement **genau einen** Datensatz mit mehreren oder sogar nur einem Feld (= Tabelle mit 1 Zeile und 1 Spalte) zurückgibt.
- In diesem Fall kann man statt eines fetch auf den ersten (und einzigen) Datensatz des Cursors mit Hilfe des Schlüsselwortes into den/die Wert(e) auch direkt in die Hostvariablen schreiben.

- *Beispiele*

  Der maximale Wert aller Preise in der Tabelle produkt wird ermittelt und der Variablen vmax zugewiesen: ⟶

```
...
vmax integer;
...
begin
  select max(preis) into vmax
  from   produkt;
...
```

```
select c1,c2,...
into   v1,v2,...
from t
where ... ;
```

⟵ Ebenso können mehrere Spalten eines einzeiligen Cursors mit into in mehrere Variablen gelesen werden.

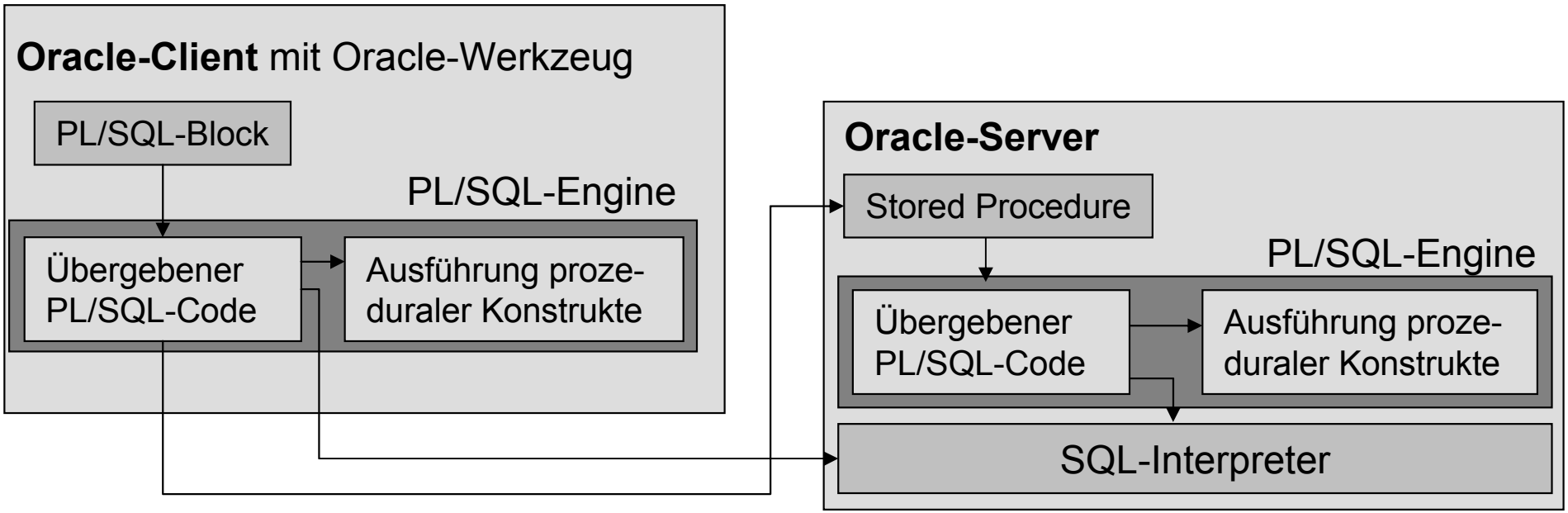
- Eine **Stored Procedure** entspricht einer Prozedur oder Funktion, die in der PL (oder auch SPL = Stored Procedure Language) des jeweiligen DBMS geschrieben wird. Source und Objectcode einer Stored Procedure werden in den Tabellen des Systemkatalogs, also im Datenbankserver, abgelegt und von dort zur Ausführungszeit von der **PL/SQL-Engine** des Systems ausgeführt.

- Innerhalb einer Stored Procedure können SQL-Befehle ausgeführt werden. Umgekehrt kann aus einer SQL-Anweisung heraus auch eine Stored Procedure aufgerufen werden.

- Eine spezielle Anwendung erfahren Stored Procedures durch die Kopplung mit **Triggern**. Ein Trigger ist ein Prozess, der bei einer der drei Datenbankoperationen **Insert**, **Update** oder **Delete** bzgl. einer bestimmten Tabelle eine Aktivität auslöst. Dies kann insbesondere der Aufruf einer Stored Procedure sein.

- Auch Trigger werden in entsprechenden Tabellen des Systemkatalogs verwaltet und auf dem DB-Server zur Laufzeit aufgerufen und ausgeführt.

- Unsere bisherige Benutzung von PL/SQL-Blöcken reduzierte sich auf anonyme PL/SQL-Blöcke. Der anonyme PL/SQL-Block wird vom Client an den Server übergeben und dort von der PL/SQL-Engine ausgeführt.
- Verfügt ein lokal verwendetes Werkzeug über eine PL/SQL-Engine, so wird diese zur Ausführung (z.B. *Oracle*Forms*) der PL/SQL-Blöcke aufgerufen, andernfalls die des Servers.
- Werden jedoch Prozeduren auf dem DB-Server gespeichert, so wird zu deren Ausführung immer die PL/SQL-Engine des Servers benutzt, und es kann zu einer gemischten Nutzung der Engine kommen:



6.2.1 Stored Procedures: Procedures & Functions in PL/SQL (Oracle)

- Grundsätzlich können Prozeduren und Funktionen in PL/SQL-Blöcken deklariert werden, ohne dass sie auf dem DB-Server gespeichert werden.
- **Deklaration einer PL/SQL-Procedure**:

```
PROCEDURE <PROC_NAME> [PARAMETERLISTE]
IS
    <Deklarationsteil>
BEGIN
    <Anweisungen>
END <PROC_NAME>;
```

- Die **Deklaration einer PL/SQL-Function** erfolgt analog – sie beginnt jedoch mit dem Schlüsselwort **FUNCTION**. Jede Function gibt einen Wert vom Typ eines PL/SQL-Datentyps zurück und enthält im Ausführungsteil eine entsprechende **return**-Anweisung:

```
FUNCTION <FUNC_NAME> [PARAMETERLISTE]
RETURN <PL/SQL-Datentyp> IS  . . .
```

- Die Übergabe der Parameter kann durch *call-by-value*, *call-by-reference* oder in einer Mischform erfolgen.

6.2.1 Stored Procedures & Stored Functions (Oracle)

- Stored Procedures und Stored Functions werden genauso deklariert, wie PL/SQL-Procedures und Functions, die nicht auf dem DB-Server gespeichert werden sollen.
- Zusätzlich werden diese dann durch den CREATE-Befehl zu Datenbankobjekten, d.h. Source- und Objectcode werden im DB-Server gespeichert und auch immer auf der dortigen PL/SQL-Engine ausgeführt:

```
CREATE PROCEDURE <PROC_NAME> [(PARAMETER)]
AS ...
```

```
CREATE FUNCTION <FUNC_NAME> [(PARAMETER)]
RETURN <PL/SQL-Datentyp>
AS ...
```

- Procedures und Functions dienen zur Modularisierung von PL/SQL-Blöcken und müssen am Ende des Deklarationsteils vollständig definiert werden. Sie sind dann - wie alle anderen PL/SQL- Variablen dieses Blockes - nur lokal im Block verfügbar.

6.2.1 Beispiel für eine PL/SQL-Procedure ohne Übergabeparameter (Oracle)

- Das Schlüsselwort declare vor der Variablendeklaration wird nicht benötigt, wenn der PL/SQL-Block als Rumpf einer Procedure- oder Function-Deklaration verwendet wird:

```
procedure  neugehalt
is
   cursor c is select gehalt
                  from angestellter
               for update of gehalt;
   v_gehalt angestellter.gehalt%type;

-- Ausführungsblock
begin
   -- unverändert gegenüber der anonymen Version
end;
/
```

- Aufruf aus einem PL/SQL-Block: **neugehalt;**

6.2.1 Beispiel für eine PL/SQL-Procedure mit Übergabeparameter (Oracle)

- Die Datentypen der Parameter werden nur in ihrer allgemeinen und nicht in ihrer eingeschränkten Form zugeordnet, d.h. der Parameterdatentyp varchar2(20) ist nicht zulässig, sondern nur der generische Datentyp varchar2.

```
procedure neugehalt (proz1 number, proz2 number)
is
-- Deklaration wie oben ...
-- Ausführungsblock wie oben ...
    if      v_gehalt > 10000
    then update angestellter set gehalt = gehalt * proz1
            where current of c;
    else  update angestellter set gehalt = gehalt * proz2
            where current of c;
    end if;
-- ...
```

- Aufruf aus einem PL/SQL-Block: **neugehalt(1.03,1.02);**

6.2.1 Beispiel für eine gespeicherte PL/SQL-Procedure (Oracle)

- Die **create**-Anweisung bewirkt, dass Source- und Objektcode der Procedure **neugehalt** im Systemkatalog der Datenbank gespeichert werden.

- Die **replace**-Anweisung bewirkt, dass eine evtentuell schon vorhandene Procedure gleichen Namens überschrieben wird - man erspart sich so das vorherige **drop** dieser Procedure.

```
create or replace procedure neugehalt (proz1 number, proz2 number)
as
   -- unverändert gegenüber der vorherigen Procedure
   -- . . .
```

- Aufruf aus einem PL/SQL-Block: **neugehalt(1.03,1.02);**

- Aufruf interaktiv aus *SQL*PLUS*: **SQL> exec neugehalt(1.03,1.02)**

6.2.1 Stored Procedure im Oracle-Systemkatalog

```
SQL> desc user_objects
 Name                                      Null?    Typ
 ----------------------------------------- -------- ----------------------------
 OBJECT_NAME                                        VARCHAR2(128)
 ...
 OBJECT_TYPE                                        VARCHAR2(18)
 ...
```

```
select object_name
from   user_objects
where object_type = 'PROCEDURE';
```

```
SQL> desc user_source
 Name                                      Null?    Typ
 ----------------------------------------- -------- ----------------------------
 NAME                                               VARCHAR2(30)
 TYPE                                               VARCHAR2(12)
 LINE                                               NUMBER
 TEXT                                               VARCHAR2(4000)
```

```
select text
from  user_source
where name = <(stored procedure|trigger|type)name>
```

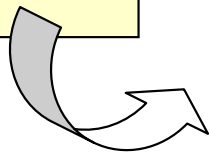
6.2.1 Packages (1) (Oracle)

- In Packages (Paketen) können logisch zusammengehörige Member zu einer größeren, modularen Einheit zusammengefasst werden.
- Member können *Typen*, *Variablen*, *Procedure* oder *Function* sein.
- Member eines Package können *public* oder *private* sein.
- Packages unterstützen eine objektorientierte Arbeitsweise dadurch, dass sie Mechanismen zur Kapselung bereitstellen: Ein Package kapselt private Daten und stellt durch ausführbare Prozeduren Schnittstellen zur Verfügung, mit denen u.a. auch die Daten manipuliert werden können. Im Unterschied zur Klassendeklaration können aus Packagedeklarationen jedoch keine Objekte instantiiert werden!
- Jedes Package enthält einen
  - *Packagekopf,* und einen
  - *Packagerumpf,* die jeweils eigens deklariert werden.
- Alle öffentlichen Member eines Paketes können über <PACKAGENAME>.<MEMBERNAME> adressiert werden.
- Packages sind Datenbankobjekte und werden durch CREATE als solche generiert:

6.2.1 Packages (2) (Oracle)

- *Beispiel*

```
-- Packagekopf
create package emp_mgmt
as
   function hire_emp(name varchar2,
                     job varchar2,
                     mgr number, ...)
   return number;
   procedure fire_emp(emp_id number);
   procedure sal_raise(emp_id number,
                       sal_incr number);
end emp_mgmt;
```



```
-- Packagerumpf
create package body emp_mgmt
as
   function hire_emp (name ...)
   return number
   is
      new_empno number(10);
   begin
      ...
   end hire_emp;

   procedure fire_emp(...)
   is ...;

   procedure sal_raise(...)
   is ...;

end emp_mgmt;
```

6.2.1 Vorteile von Stored Procedures und Packages

- **Stored Procedures** stehen zentral auf dem DB-Server für alle Anwendungsentwickler zur Verfügung.
- Die Kommunikationskosten (netzweit) reduzieren sich, da nur das EXEC PROCEDURE und die aktuellen Parameter über die Leitung geschickt werden müssen.
- In der Regel reduzieren sich die Ausführungszeiten bei Ausführung auf dem Datenbankserver.

- **Pakete** können auf anderen Paketen basieren, sodass Objekte von Paket zu Paket vererbt werden können.
- Objekte, die im Rahmen eines Programmes P1 instanziiert worden sind, bleiben im Hauptspeicher für die Dauer der aktuellen Session auch erhalten, wenn Programm P1 bereits beendet wurde. Sie können also von anderen Programmen verwendet werden und dienen somit insbesondere als „Schnittstelle“ für die Übergabe von Parametern.
- Pakete können auf verschiedenen Netzknoten verteilt sein; dies ist für die Thematik „Lastverteilung und Performance in verteilten Systemen“ von Bedeutung.

6.2.2 Trigger

- Ein Trigger ist eine PL/SQL-Prozedur, die nicht durch einen expliziten Aufruf, sondern beim Eintreten eines bestimmten Ereignisses ausgeführt wird.

- Jeder Trigger bezieht sich auf genau eine Tabelle und kann in Bezug auf die folgenden drei Aspekte definiert werden:
  - ereignisgesteuert aufgrund einer **Datenmanipulation** (insert, update, delete)
  - bezogen auf einen bestimmten **Zeitpunkt** (before oder after) bzgl. des Ereignisses,
  - **befehls-** oder **datensatzorientiert** in Bezug auf die ausgelöste Aktivität.

- Für jede Kombination der genannten drei Aspekte und pro Spalte (im Falle eines Updates) kann pro Tabelle maximal ein Trigger definiert werden.

- Trigger sind geeignet, die Integrität von Datenbeständen zu gewährleisten, insbesondere in verteilten Systemen.

- Bei der Deklaration von Triggern ist darauf zu achten, dass eine entsprechende Wirkung nicht vielleicht schon durch einen geeigneten `constraint` erzielt wird - dies ist im Zweifelsfall stets zu bevorzugen.

6.2.2 Aufbau eines Triggers (Oracle)

create trigger <*Trigger_Name*>	Der Trigger-Name sollte andeuten, um welche Art von Trigger es sich handelt, z.B. TEIL_BR_UPDATE (B=before, R=row: zeilenweise)
before | **after**	legt fest, ob vor oder nach dem auslösenden Ereignis reagiert wird.
insert or **update** [of <*col1, ...* >] or **delete on <*Table_Name*>**	definiert das auslösende Ereignis
[**for each row**] Blocks, vor Datensatz	Fehlt diese Klausel, arbeitet der Trigger befehlsorientiert und nicht zeilenorientiert. ***Befehlsorientiert***: einmalige Ausführung des PL/SQL- oder nach dem auslösenden Ereignis ***Zeilenorientiert***: Ausführung des PL/SQL-Blocks pro
[**when** <*Prädikat*>]	Diese Restriktion beschreibt die Bedingung, unter der der Trigger ausgeführt werden soll.
<*PL/SQL-Block*>	Der Trigger-Rumpf besteht aus einem anonymen PL/SQL-Block oder dem Aufruf einer Stored Procedure.

6.2.2 Trigger – Beispiel 1 (Oracle)

- Was bewirkt der folgende Trigger:

```
create trigger TEIL_B
before insert or update or delete on TEIL
when (user != 'SYSTEM')

-- anonymer PL/SQL-Block
declare
   Eingabe_nicht_zulaessig exception;
begin
   if to_char(sysdate, 'HH24:MI')
      not between '08:00' and '17:00'
   then raise Eingabe_nicht_zulaessig;

exception
   when Eingabe_nicht_zulaessig then
   raise_application_error
   (-20001, 'Nur zwischen 8:00 und 17:00 Uhr Daten eingeben! ');
end;
/
```

Triggerkopf

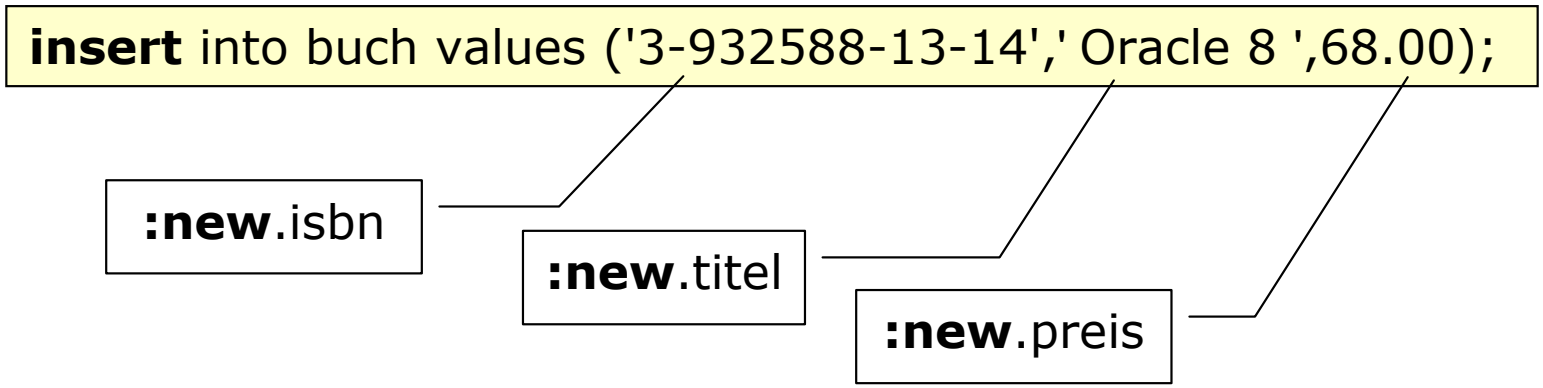
Triggerrumpf

6.2.2 Trigger – die FOR EACH ROW-Klausel (Oracle)

- Bei **zeilenorientierten Triggern** ist es möglich, auf Werte vor und nach einer Änderung mit **:old.**<Spaltenname> und **:new.**<Spaltenname> zuzugreifen:

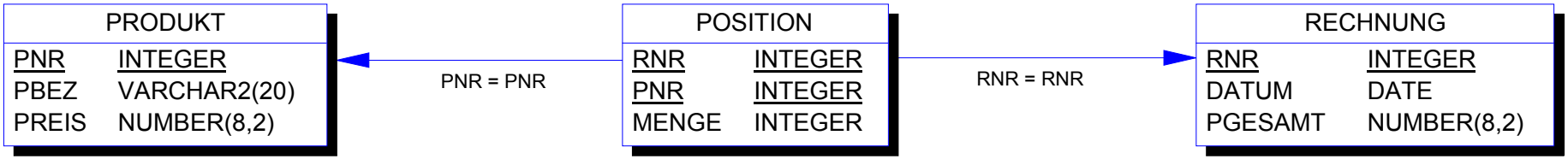
Bei **Update-Triggern** können		**:old.<Spaltenname>** und **:new.<Spaltenname>** benutzt werden,
bei **Insert-Triggern**	nur	**:new.<Spaltenname>**,
bei **Delete-Triggern**	nur	**:old.<Spaltenname>**.

- Die Operatoren :new und :old können, angewandt auf die Spaltennamen der manipulierten Tabelle, z.B als Übergabeparameter von Triggern an Stored Procedure genutzt werden.



6.2.2 Trigger – Beispiel 2: das Datenmodell (Oracle)

- Zu folgendem Datenmodell soll ein Trigger erstellt werden, der gewährleistet, dass der Gesamtbetrag PGESAMT einer RECHNUNG stets der Summe von Preisen für die Einzelpositionen dieser Rechnung entspricht.



- Der Trigger soll immer dann ausgelöst werden, wenn ein entsprechender Datensatz in die Tabelle POSITION eingefügt wird – es handelt sich also um einen INSERT-Trigger der Tabelle POSITION.

- Zur Ermittlung des Teilrechnungsbetrages einer einzelnen POSITION und der Erhöhung des Gesamtbetrags der zugehörigen Rechnung soll eine Stored Procedure erstellt werden, die von diesem Trigger aufgerufen wird.

6.2.2 Trigger – Beispiel 2: Stored Procedure und Trigger (Oracle)

- Die Werte der neu eingefügten POSITION werden mit **:neu.<Spaltenname>** adressiert!