we focus on students

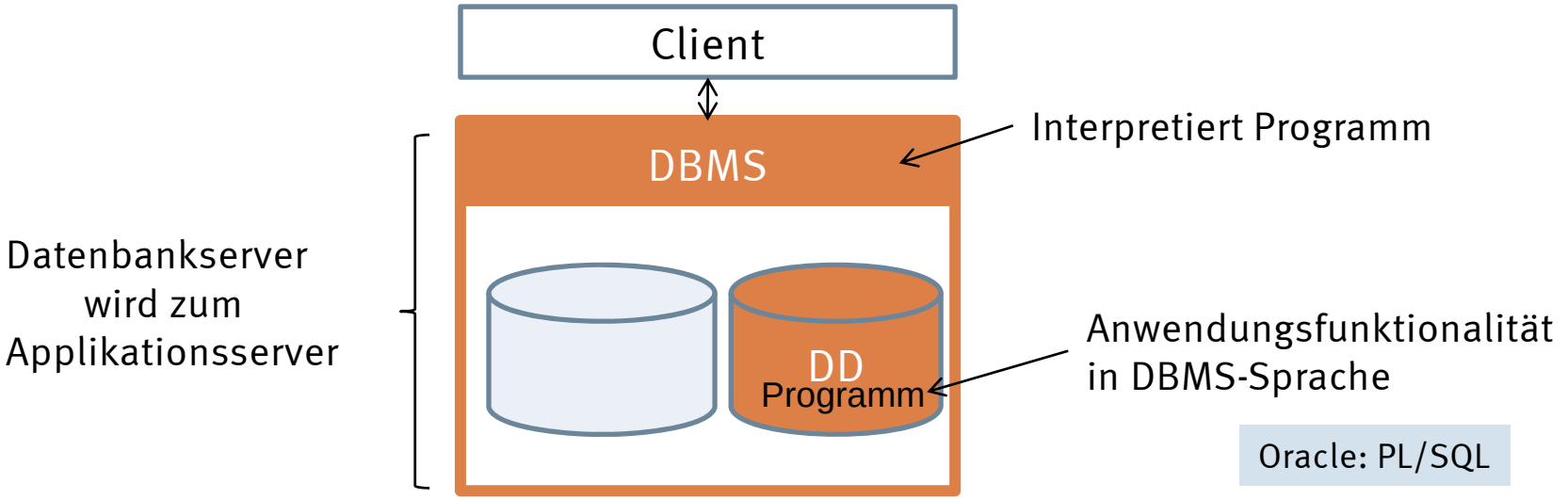
Datenbankprogrammierung

Aktive Datenbank

Fachhochschule Dortmund
University of Applied Sciences

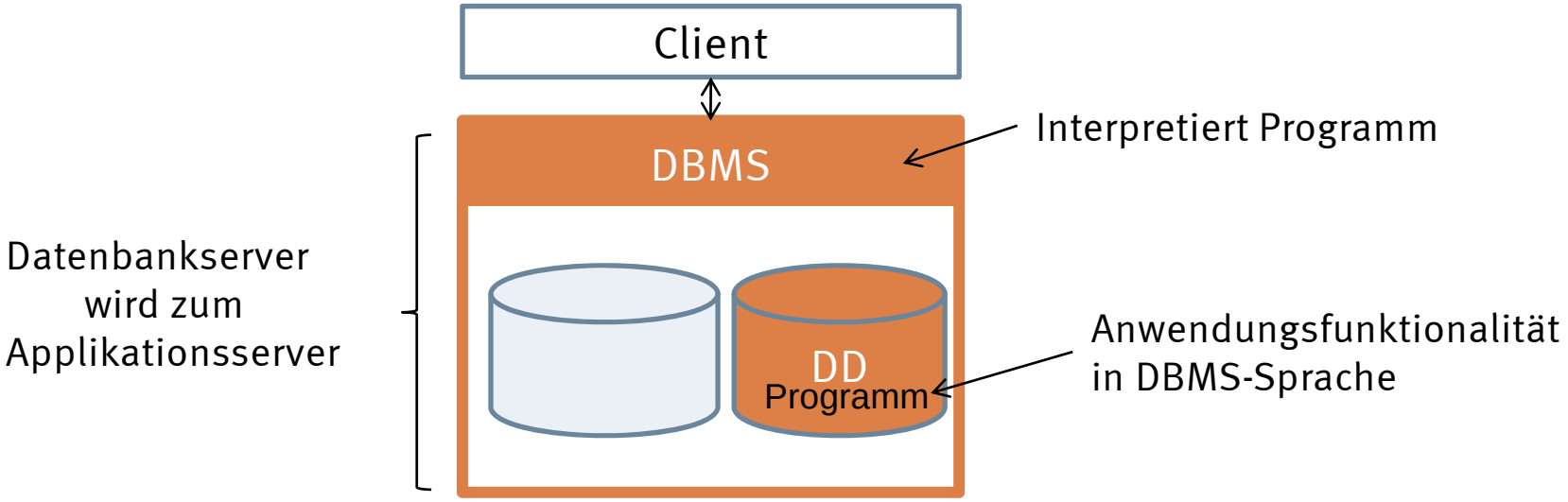
Aktive Datenbank

Bei einer **aktiven Datenbank** übernimmt das DBMS Anwendungsfunktionalitäten vom Client.



i.d.R. nicht direkt portierbar

Aktive Datenbank



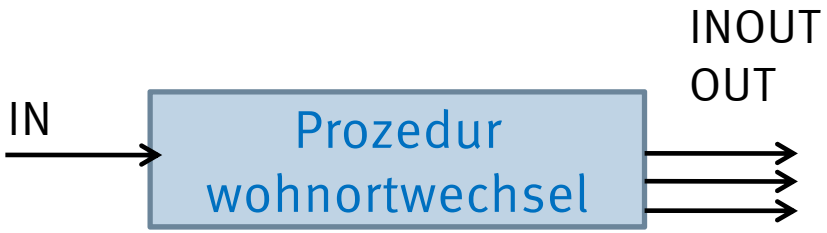
- Vorteile
  - Clients werden schlanker
  - Schnellere Daten-Updates (kein Netz-Traffic)
  - Verfügbarkeit allgemein nutzbarer Funktionen

Programmierkonzepte (Datenbank)

- Gespeicherte Funktionen (stored function)

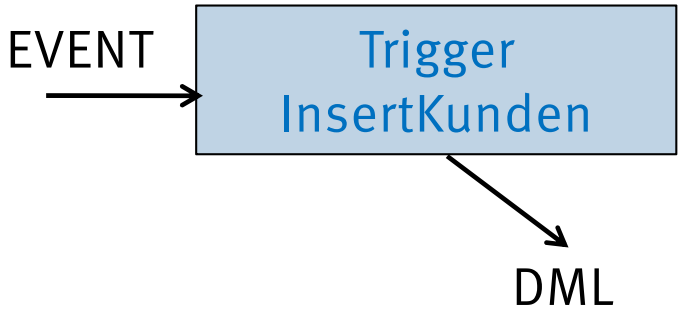

```
SELECT kundenanrede(Kundennummer)
FROM    Kunde
```

- Gespeicherte Prozeduren (stored procedures)



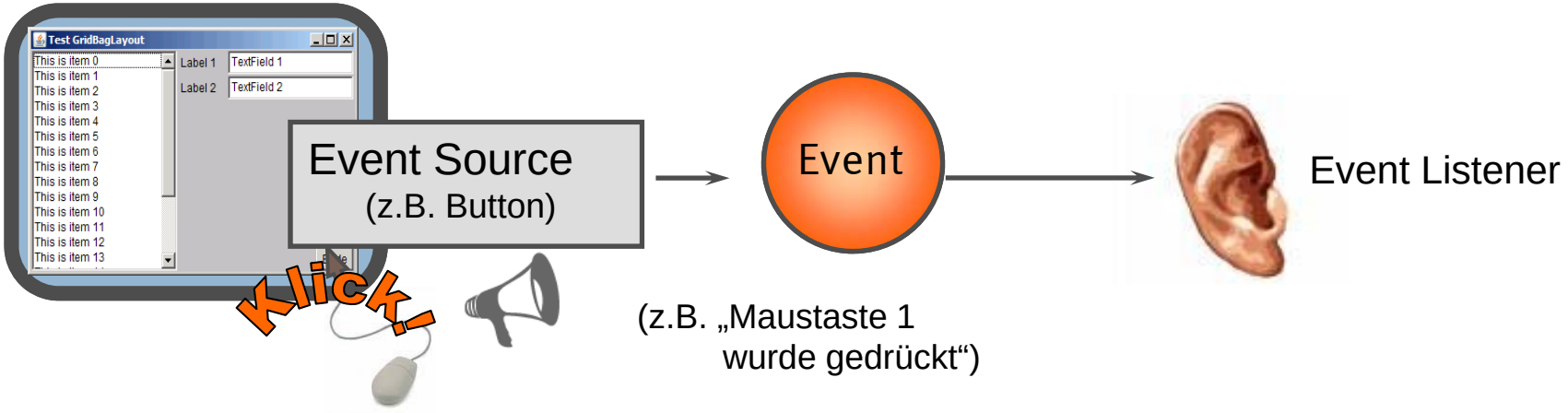

```
CALL kundenliste('Dortmund')
```

- Eventgesteuerte Prozeduren (Trigger)

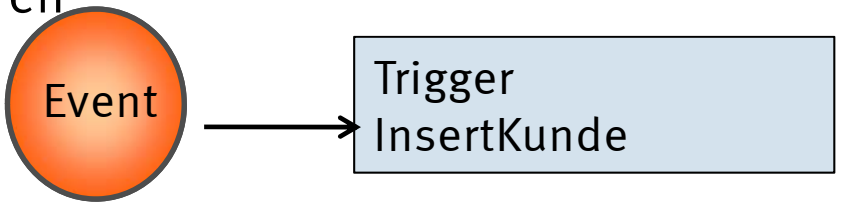


Ereignissteuerung

- In Java:



- Bei Datenbanken
  - Eventgesteuerte Prozeduren (Trigger)



z.B. INSERT INTO Kunde (...)

Statische und dynamische Integritätsbedingungen

Statische Integritätsbedingungen müssen zu jedem Zeitpunkt eingehalten werden.

Beispiel:

Das Geburtsdatum muss vorgegeben werden.

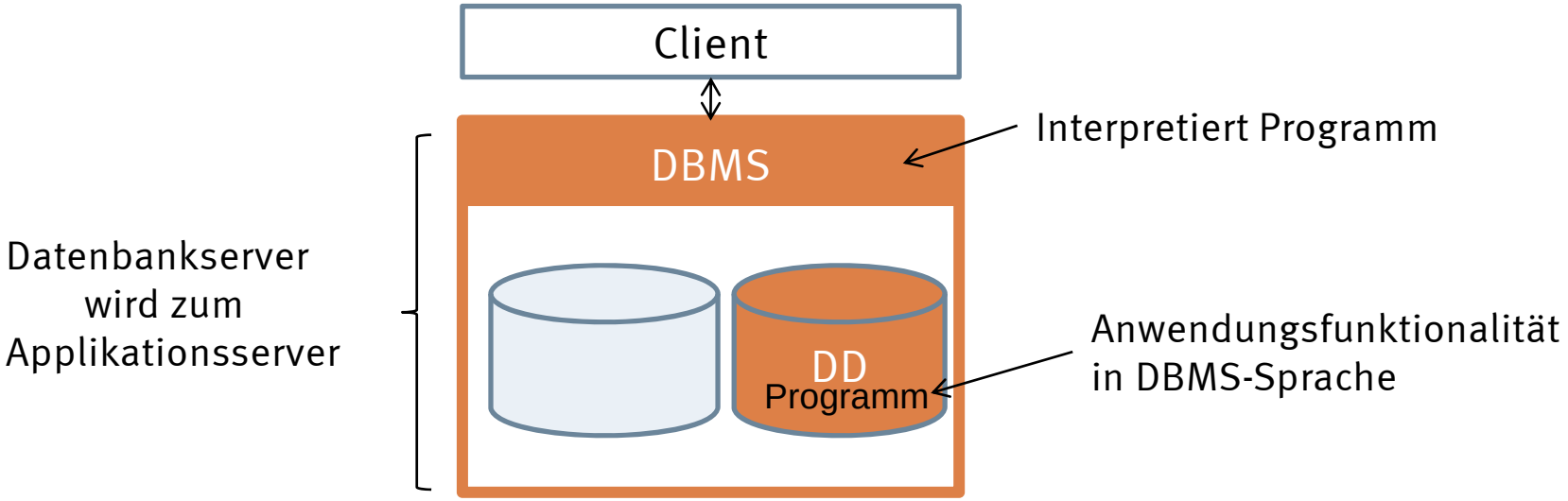
Dynamische Integritätsbedingungen beziehen sich auf den **aktuellen** Inhalt der Datenbanktabellen und deren Änderungen.

Beispiele:

- Das Geburtsdatum darf nicht in der Zukunft liegen.
- Ein Kunde erhält im Monat seines Geburtstags einen 10%igen Geburtstagsrabatt.

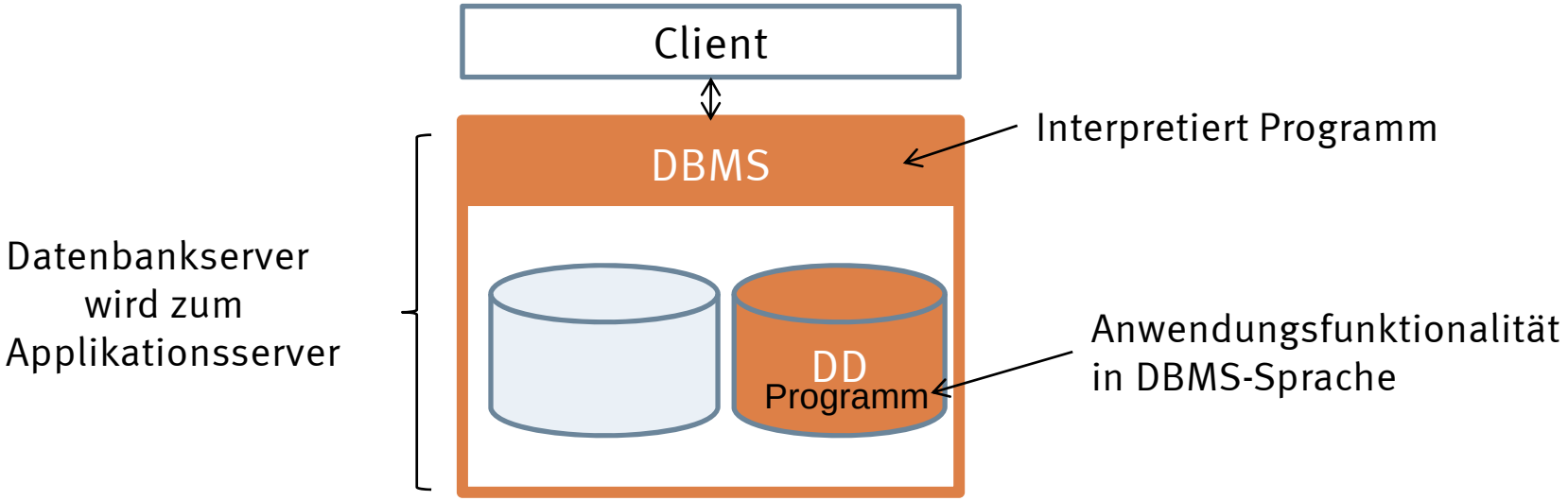
Zusammenfassung

Bei einer **aktiven Datenbank** übernimmt das DBMS Anwendungsfunktionalitäten vom Client.



1. Gespeicherte Funktionen (stored function)
2. Gespeicherte Prozeduren (stored procedures)
3. Eventgesteuerte Prozeduren (Trigger)

Aktive Datenbank



- Verwendung
  - Integritätssicherung (Trigger)
  - Anwendungsfunktionalität wird bereitgestellt (Funktionen, Prozeduren)
  - Datenschutz (Prozeduren)

we
focus
on
students

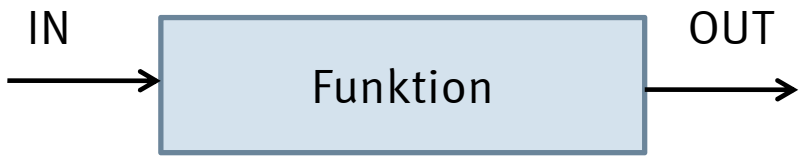
Datenbankprogrammierung

Gespeicherte Funktionen

Fachhochschule
Dortmund
University of Applied Sciences

Fragestellung

Konzept der Funktion



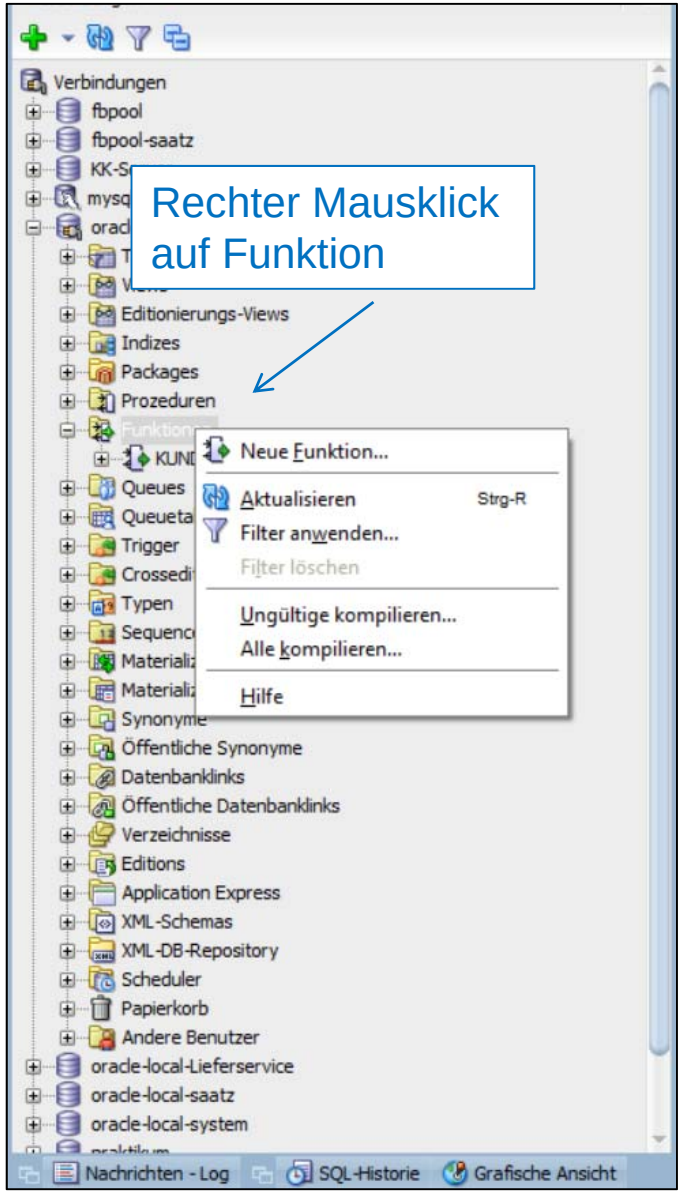
In Datenbanken

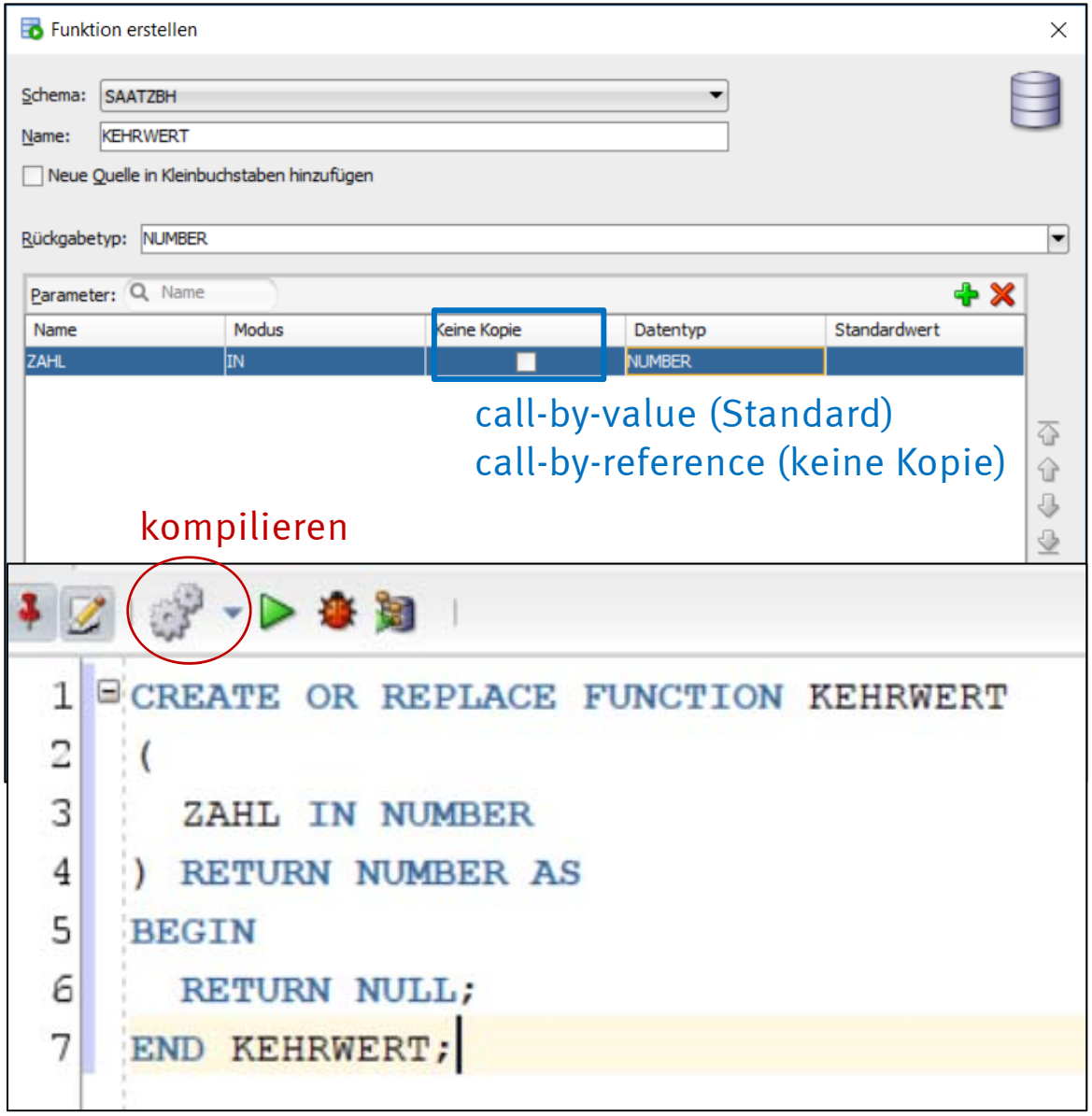
- Durch das DBMS bereitgestellte Funktionen
  - Mathematische Funktionen (z.B. round(), sin(), …)
- Benutzerdefinierte Funktion

```
SELECT Kehrwert(2) FROM dual
```

Wie kann eine benutzerdefinierte Funktion implementiert werden?

SQLDeveloper - Funktionen anlegen





Variablenzuweisung durch SQL-Anfragen

Deklaration lokaler Variablen

```
variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value]
```

Wertzuweisung

var_name := *expr* | variable | konstante

SELECT *spalte*[,...] INTO *var_name*[,...] FROM tabelle

```
CREATE OR REPLACE FUNCTION kehrwert (zahl IN INTEGER) RETURN
NUMBER
AS
  rueckgabe NUMERIC( 9,8);
BEGIN
    rueckgabe:=1/zahl;
   RETURN rueckgabe;
END;
```

Vorsicht Falle!

RETURN Numeric → Rückgabewert gerundet ohne Nachkommastellen
RETURN Number → Rückgabewert mit Nachkommastellen

Ausnahmebehandlung


```
CREATE FUNCTION name (...)
AS
      myfehlermeldung    EXCEPTION;
BEGIN
    IF ‹bedingung›
       THEN
            RAISE myfehlermeldung;
       END IF;
EXCEPTION
      WHEN myfehlermeldung
      THEN raise_application_error(-20500,'Mein Fehlertext');
END;
```

Fehler deklarieren

Fehler werfen

Fehler behandeln

Fehlernummer

Abbruch der Bearbeitung signalisieren

Fallunterscheidung und Ausnahmebehandlung


```
IF ‹bedingung› THEN     ‹anweisungen›
   [ELSIF ‹bedingung› THEN ‹anweisungen›]
   [ELSE ‹anweisungen›]
END IF
```

```
CREATE OR REPLACE FUNCTION kehrwert (zahl IN INTEGER) RETURN NUMBER
AS
 rueckgabe NUMERIC( 9,8);
 myfehlermeldung    EXCEPTION;
BEGIN
    IF zahl=0
    THEN
       RAISE myfehlermeldung;
    END IF;
 rueckgabe:=1/zahl;
 RETURN rueckgabe;
EXCEPTION
      WHEN myfehlermeldung
      THEN raise_application_error(-20500,'Kehrwert existiert nicht.');
END;
```

Installationsskript


```
1  CREATE OR REPLACE FUNCTION KEHRWERT
2  (
3    ZAHL IN NUMBER
4  ) RETURN NUMBER AS
5  BEGIN
6    RETURN 1/zahl;
7  END KEHRWERT;
8  /
9  SET serveroutput ON;
10 SELECT Kehrwert(2) FROM dual;
```

Skriptausgabe

Aufgabe abgeschlossen in 0,015 Sekunden

```
Function KEHRWERT kompiliert


                              KEHRWERT(2)
----------------------------------------
```

SQLDeveloper:
Trennzeichen / nach der Definition von DB-Programmen

Einschalten der Konsolenausgabe (optional)

Datenbankprogrammierung

Gespeicherte Prozeduren (Teil 1)

Fachhochschule
Dortmund
University of Applied Sciences

Fragestellung

View KundenAus Dortmund

Kunden-nummer	Nach-name	Vorname	Anrede	Geburts-datum	Ort
2310	Meitner	Lise	Frau	17.11.1878	Berlin
8365	Einstein	Albert	Herr	14.03.1879	Princeton
8523	Dekanat	Informatik	NULL	NULL	Dortmund
8524	Meier	Max	Herr	24.12.1987	Dortmund

Fragestellung

View KundenAus Dortmund

Wohnortwechsel

View KundenAus Bochum

```
CREATE View KundenAusDortmund
AS
        SELECT Kundennummer, Nachname, Ort FROM Kunde
        WHERE  Ort='Dortmund'
        WITH CHECK OPTION
```

Fragestellung

View KundenAus Dortmund

Wohnortwechsel →

View KundenAus Bochum

```
CALL Wohnortswechsel (8524, 'Dortmund', 'Bochum')
```

Wie wird die Prozedur Wohnortwechsel implementiert?

Gespeicherte Prozedur Wohnortwechsel


```
CREATE OR REPLACE PROCEDURE Wohnortwechsel (
                Knr         IN     INT,
                alterOrt    IN     VARCHAR2,
                neuerOrt    IN     VARCHAR2)
IS
      wohnort               VARCHAR2(200);
      falscherWohnort    EXCEPTION;
BEGIN
     SELECT  Ort  INTO wohnort
      FROM  Kunde
     WHERE  Kundennummer = Knr;
     IF alterOrt = RTRIM(wohnort,' ') THEN
            UPDATE Kunde SET Ort = neuerOrt
            WHERE  Kundennummer=Knr;
      ELSE
            RAISE falscherWohnort;
      END IF;
EXCEPTION
      WHEN falscherWohnort
      THEN raise_application_error
           (-20500,'Aktueller Wohnort fehlerh.');
END;
```

Selektion des bisherigen Wohnortes

Prüfung der Eingaben
Änderung des Wohnortes

Fehlermeldung werfen

Fehler behandeln

Syntax

```
CREATE PROCEDURE ‹procedure_name›
    [(‹argument1›, ...) ]
{IS | AS}
  [‹Deklarationen lokaler variablen›]
BEGIN
   ‹ausfuehrbare anweisungen›

[EXCEPTION ‹ausnahmebehandlung›]

END [‹procedure_name›]
```

Übersicht

	Gespeicherte Prozedur	Gespeicherte Funktion
Aufruf	CALL	SELECT
Aufruf-parameter	IN INOUT OUT	IN
Rückgabe-werte	Mehrere OUT-Parameter	Ein Wert
Erlaubte Befehle	DRL DML DDL DCL	DRL
Aufruf von	Funktionen Prozeduren	Funktionen

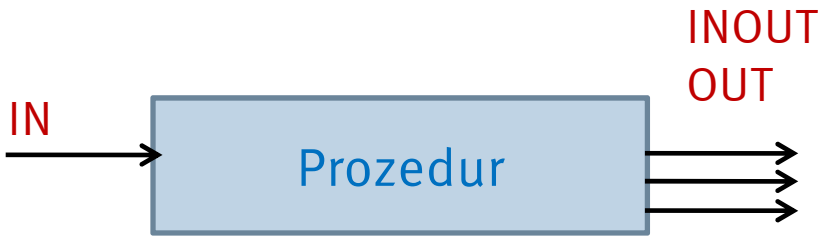
we focus on students

Datenbankprogrammierung

Rückgabeparameter nutzen

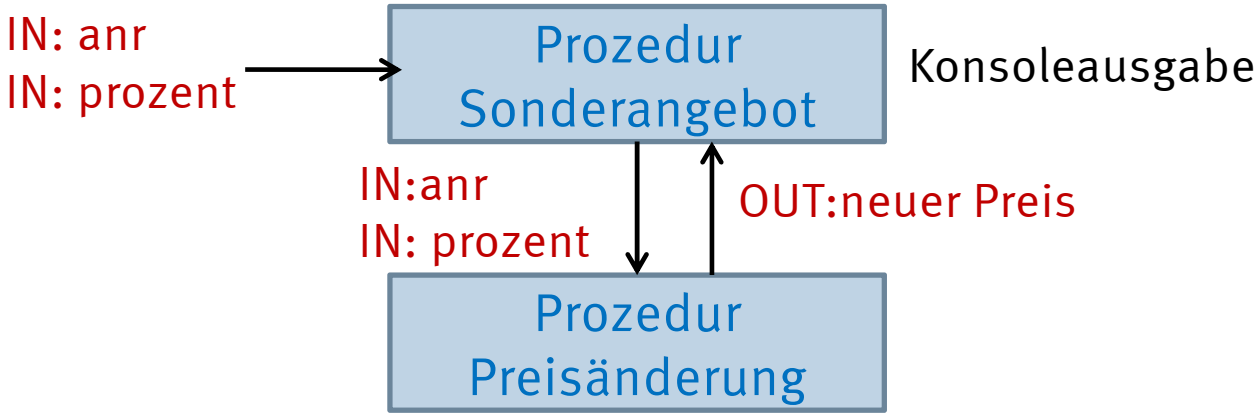
Fachhochschule Dortmund
University of Applied Sciences

Fragestellung



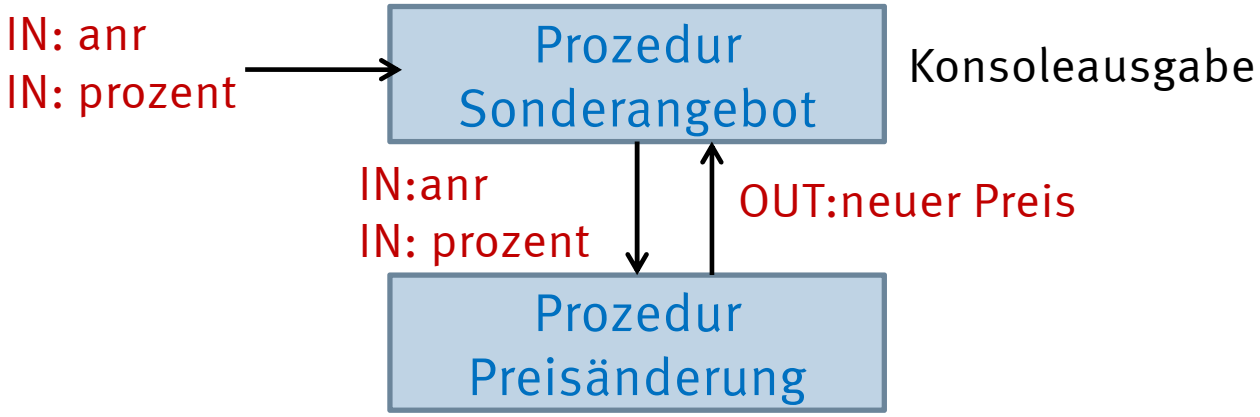
Wie werden (IN)OUT Parameter verwendet?

Prozedur Sonderangebot



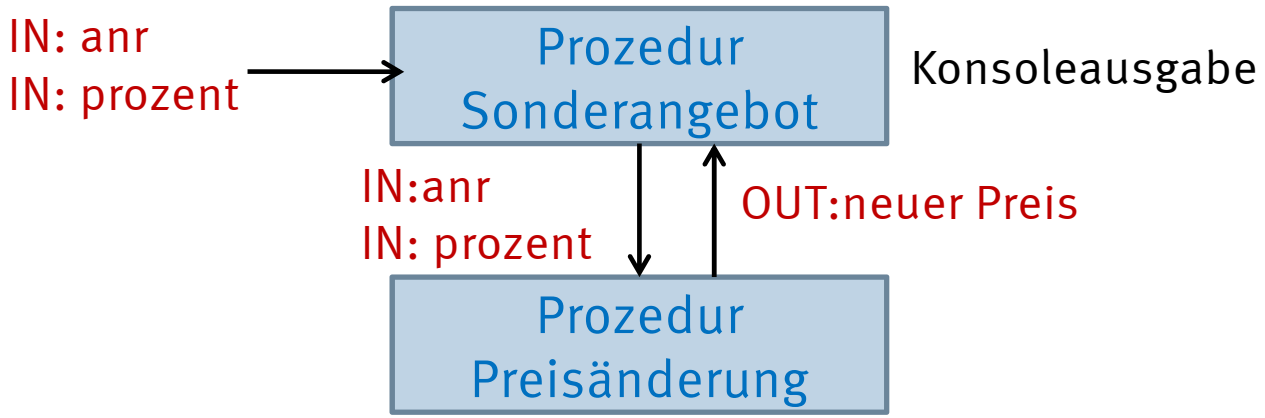

```
CREATE OR REPLACE PROCEDURE SONDERANGEBOT (anr IN INTEGER, prozent IN INTEGER)
IS sonderpreis NUMBER;
BEGIN
  PREISAENDERUNG(anr,prozent, sonderpreis);
  dbms_output.put_line('Der Sonderpreis des Artikels: ' || anr || ' ist: ' || sonderpreis );
END SONDERANGEBOT;
```

Prozedur Preisänderung




```
PROCEDURE PreisAenderung (anr IN INTEGER, aenderung IN Number, neuerPreis OUT Number)
AS
BEGIN
  UPDATE Artikel
  SET preis= preis*(1+aenderung/100)
  WHERE Artikelnummer=anr
  RETURNING preis INTO neuerPreis;
END;
```

Die Returning-Klausel



- RETURNING-Klausel:

```
INSERT INTO ‹table› (c1, c2, .., cn) VALUES (v1, v2, .., vn)
        RETURNING ‹expression› INTO ‹variables›
UPDATE ‹table› SET (c1) = (v1), (c2) = (v2), (cn) = (vn) WHERE ‹condition›
        RETURNING ‹expression› INTO ‹variables›
DELETE FROM ‹table› WHERE ‹condition›
        RETURNING ‹expression› INTO ‹variables›
```

Zusammenfassung

Aufruf der Prozedur Preisänderung aus der Prozedur Sonderangebot:

```
CREATE OR REPLACE PROCEDURE SONDERANGEBOT (anr IN INTEGER, prozent IN INTEGER)
IS sonderpreis NUMBER;
BEGIN
  PREISAENDERUNG(anr,prozent, sonderpreis);
  dbms_output.put_line('Der Sonderpreis des Artikels: ' || anr || ' ist: ' || sonderpreis );
END SONDERANGEBOT;
```

Durchführung der Preisänderung:

```
PROCEDURE PreisAenderung (anr IN INTEGER, aenderung IN Number,neuerPreis OUT Number)
AS
BEGIN
  UPDATE Artikel
  SET preis= preis*(1+aenderung/100)
  WHERE Artikelnummer=anr
  RETURNING preis INTO neuerPreis;
END;
```

RETURNING Klausel

Änderungsoperation können bei Oracle Rückgabewerte mittels der RETURNING-Klausel zurückliefern. Dies ist insbesondere hilfreich bei der Abfrage eines automatisch generierten Primärschlüsselwerts. Hierdurch wird eine zusätzliche Selektieren des (Primärschlüssel-)Wertes vermieden.

- Syntax der RETURNING-Klausel:

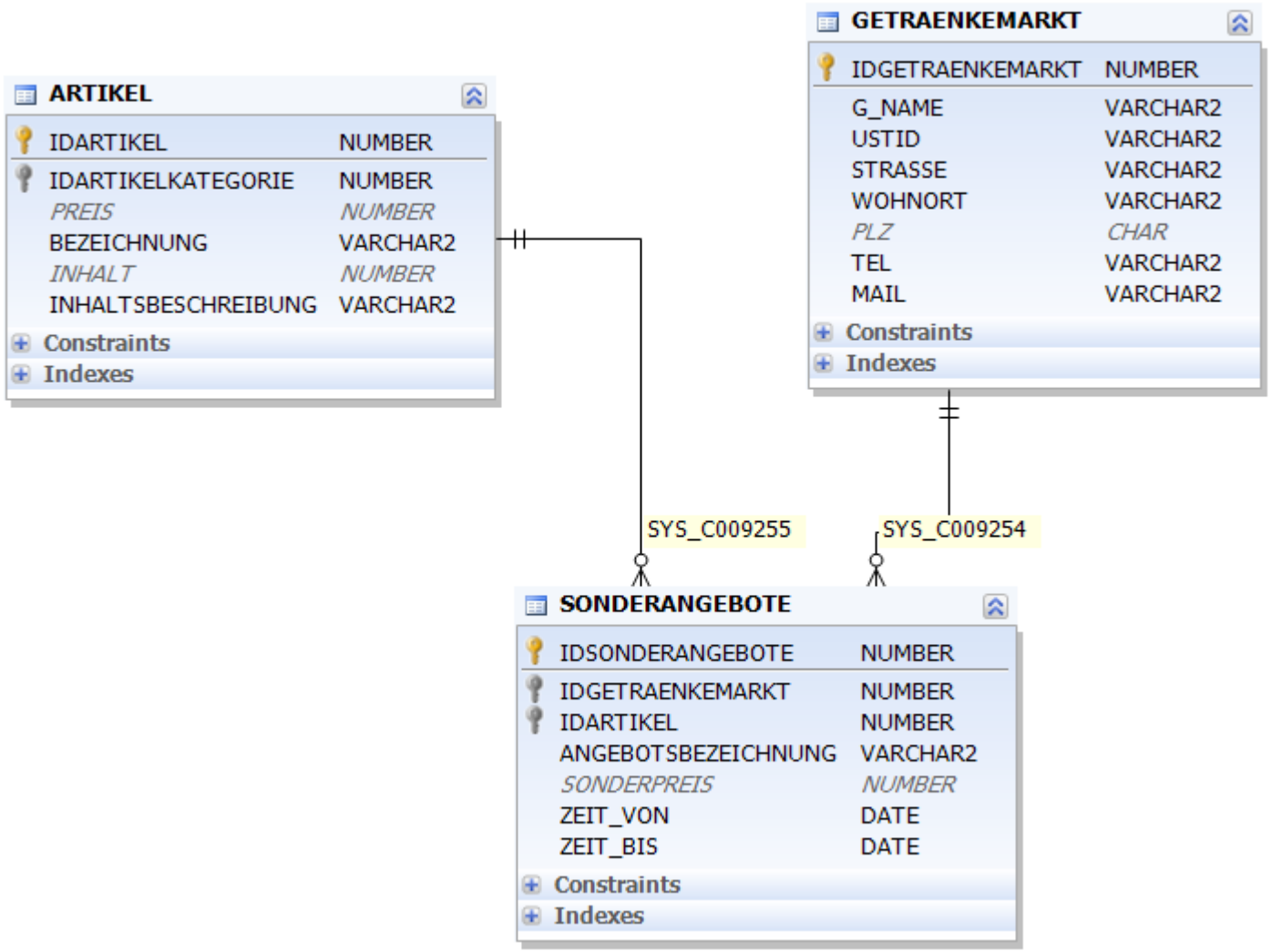
```
INSERT INTO ‹table› (c1, c2, .., cn) VALUES (v1, v2, .., vn)
        RETURNING ‹expression› INTO ‹variables›
UPDATE ‹table› SET (c1) = (v1), (c2) = (v2), (cn) = (vn) WHERE ‹condition›
        RETURNING ‹expression› INTO ‹variables›
DELETE FROM ‹table› WHERE ‹condition›
        RETURNING ‹expression› INTO ‹variables›
```

- Beispiel:

Übername des Spaltentyps (hier Integer) aus der Basistabelle

```
PROCEDURE NeuerKunde (knr OUT Kunde.Kundennummer%TYPE,
            nname IN Kunde.Nachname%TYPE, vname IN Kunde.Vorname%TYPE)
IS
BEGIN
        INSERT INTO Kunde (nachname, vorname) VALUES (nname, vname)
        RETURNING kundennummer INTO knr;
END;
```

Anpassung


ARTIKEL
IDARTIKEL NUMBER
IDARTIKELKATEGORIE NUMBER
PREIS NUMBER
BEZEICHNUNG VARCHAR2
INHALT NUMBER
INHALTSBESCHREIBUNG VARCHAR2
Constraints
Indexes
GETRAENKEMARKT
IDGETRAENKEMARKT NUMBER
G_NAME VARCHAR2
USTID VARCHAR2
STRASSE VARCHAR2
WOHNORT VARCHAR2
PLZ CHAR
TEL VARCHAR2
MAIL VARCHAR2
Constraints
Indexes
SYS_C009255
SYS_C009254
SONDERANGEBOTE
IDSONDERANGEBOTE NUMBER
IDGETRAENKEMARKT NUMBER
IDARTIKEL NUMBER
ANGEBOTSBEZEICHNUNG VARCHAR2
SONDERPREIS NUMBER
ZEIT_VON DATE
ZEIT_BIS DATE
Constraints
Indexes

we
focus
on
students

Datenbankprogrammierung

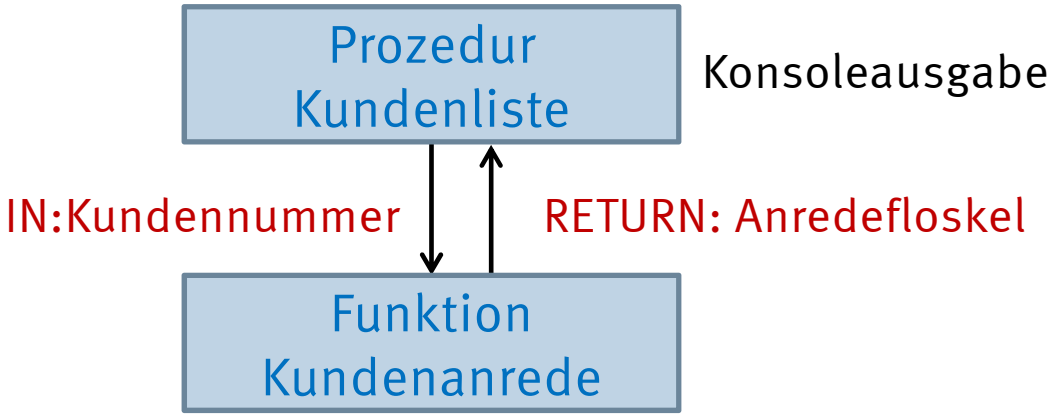
Das Cursor-Konzept

Fachhochschule
Dortmund
University of Applied Sciences

Problemstellung

Anforderungen:
Ein Datenbankprogramm soll entwickelt werden,
welches eine Kundenliste mit den zugehörenden Anredefloskeln erstellt.

```
1. Sehr geehrte Frau Meitner
2. Sehr geehrter Herr Einstein
3. Sehr geehrte Frau Curie
4. Sehr geehrte Damen und Herren
5. Sehr geehrter Herr Meier
```



Funktion Kundenanrede


```
CREATE OR REPLACE FUNCTION Kundenanrede (knr IN INTEGER)
RETURN VARCHAR2
AS
   knachname CHAR(30);
   kanrede   CHAR(5);
   rueckgabe VARCHAR(50);
BEGIN
   SELECT Nachname, Anrede INTO knachname, kanrede
   FROM Kunde
   WHERE Kundennummer=knr;
   CASE kanrede
   WHEN 'Frau' THEN rueckgabe:='Sehr geehrte Frau '|| knachname;
   WHEN 'Herr' THEN rueckgabe:='Sehr geehrter Herr '|| knachname;
   ELSE rueckgabe:='Sehr geehrte Damen und Herren ';
   END CASE;
   RETURN rueckgabe;
END Kundenanrede;
```

Funktion Kundenanrede

Prozedur Kundenliste

Prozedur Kundenliste

```
CREATE OR REPLACE PROCEDURE Kundenliste
AS
  kanrede VARCHAR (50);
BEGIN
   FOR k in (SELECT Kundennummer FROM Kunde) LOOP
      SELECT Kundenanrede(k.Kundennummer) INTO kanrede FROM dual;
      DBMS_OUTPUT.put_line(kanrede);
   END LOOP;
END Kundenliste;
```

Schlechte Implementierung

Cursor-Konzept

- DBMS

Satzanforderung (fetch)

Cursor liefert ein Ergebnistupel

```
SELECT Kundennummer, Nachname, Anrede
FROM Kunde
```

next()

	KUNDENNUMMER	NACHNAME		ANREDE
1	2310	Meitner	...	Frau
2	7562	Einstein	...	Herr

- Cursor

	Metadata	Spalte 1	Spalte 2	Spalte 3
Initial		Kunden-nummer	Nachname	Anrede
next()		2310	Meitner	Frau
next()		7562	Einstein	Herr

Cursor für mengenwertige Abfragen

Definition eines Cursors

```
DECLARE CURSOR kundencursor
    IS    SELECT Nachname FROM Kunde;
```

Öffnen des Cursors (Berechnen der Ergebnismenge)

```
OPEN kundencursor;
```

Zugriff auf Tupel (i.d.R. in einer Schleife)

```
FETCH kundencursor INTO ‹variable›
```

Schließen des Cursors, Freigabe der Tupelmenge.

```
CLOSE kundencursor;
```

Prozedur Kundenliste mit Cursor


```
CREATE OR REPLACE PROCEDURE Kundenliste
AS
  CURSOR kundencursor IS                    Cursor deklarieren
     SELECT Kundenanrede(kundennummer) as kanrede FROM Kunde;
BEGIN
   FOR k in kundencursor                    Implizites Öffnen des Cursors
   LOOP                                     Durchlaufen der Ergebnismenge
      DBMS_OUTPUT.put_line(kundencursor%rowcount ||'. '|| k.kanrede);
   END LOOP;                                Implizites Schließen des Cursors
END Kundenliste;
```

Cursor: Pseudospalten

Den Status eines Cursors beschreiben seine vier Attribute Found, NotFound, Rowcount und isOpen. Mit CURRENT OF wird auf das aktuelle Tupel verwiesen.

%FOUND

- Gibt an, ob der letzte FETCH - Befehl einen Satz gefunden hat => TRUE
- Vor dem ersten Fetch NULL

%NOTFOUND

- Gibt an, ob der letzte FETCH - Befehl einen Satz gefunden hat => FALSE
- Vor dem ersten Fetch NULL

%ROWCOUNT

- Liefert die Anzahl der mit FETCH gelesenen Zeilen
- Vor dem ersten FETCH auf 0

%ISOPEN

- Gibt an, ob ein Cursor geöffnet ist

Kontrollstrukturen - Schleifen

- FOR Schleife

```
FOR ‹var› IN [REVERSE] von … bis
LOOP
 [exit [when ‹bedingung›]]
END LOOP
```

```
FOR i IN  1..10
LOOP
      --  tue was
END LOOP;
```

- WHILE-Schleife

```
WHILE ‹bedingung› LOOP
          ‹anweisungen›
END LOOP
```

```
WHILE  i ‹ 10 LOOP
          i:=i+1;
END LOOP;
```

- REPEAT-Schleife

```
LOOP
       ‹anweisungen›
       EXIT [WHEN ‹bedingung› ];
END LOOP;
```

```
LOOP
        i:= i+1;
       EXIT WHEN i=10;
END LOOP;
```

Zusammenfassung

- Ein Cursor ist ein Zeiger auf ein Tupel der Ergebnismenge einer SQL-Anfrage.

```
CURSOR kundencursor IS
    SELECT Kundenanrede(kundennummer) as kanrede FROM Kunde;
```

- Aus dem Cursor werden solange Tupel für Tupel ausgelesen, bis keine weiteren Tupel im Cursor mehr vorhanden sind.