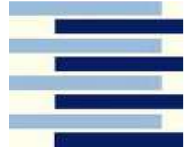
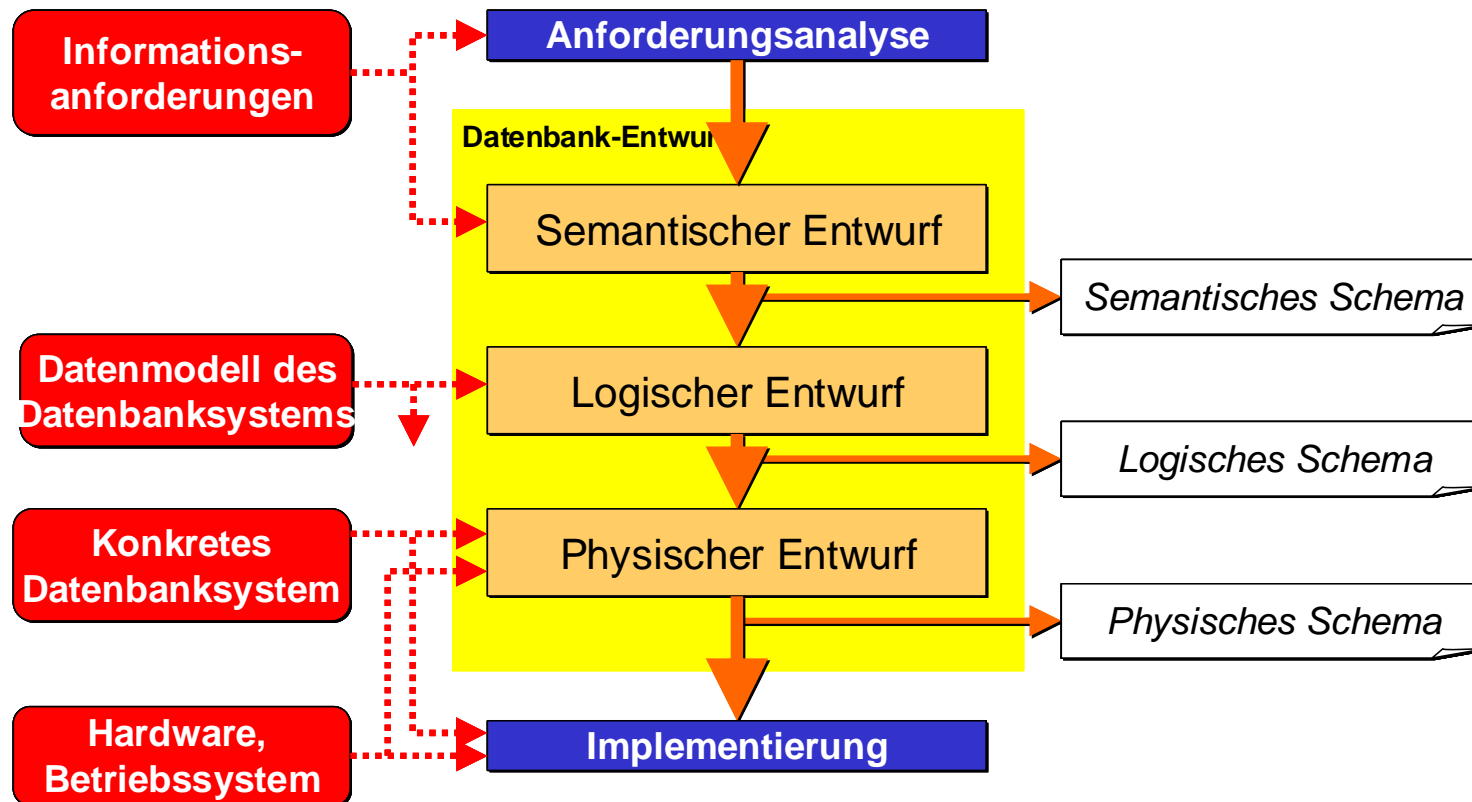


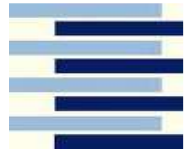
# WP Datenbankdesign



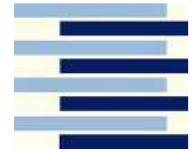
## Kapitel 2: Physische Modellierung



# Vom Relationsschema zur relationalen DB



- Create database, create tablespace, create table
- Constraints, *welche?*
- Weitere Integritätsregeln mit PL/SQL  
⇒ Trigger bzw. Stored Procedures



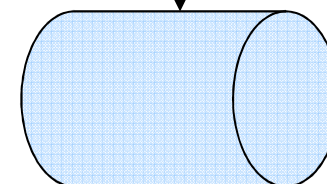
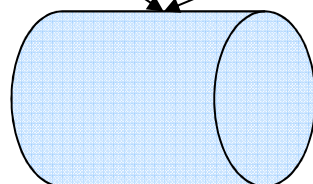
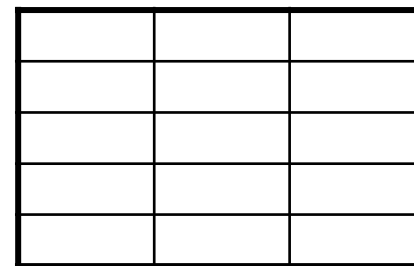
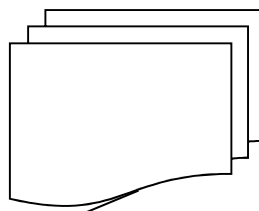
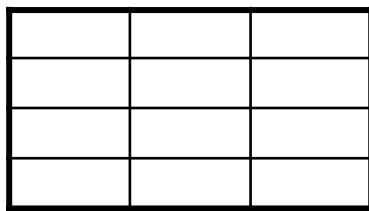
# Tablespace

- logische Struktur einer (Oracle) Datenbank
- Physikalisch durch ein oder mehrere Datenfiles repräsentiert
- innerhalb eines Tablespaces können beliebig viele Datenbankobjekte erzeugt werden
- Trennung von logischer Architektur und physikalischer Speicherung ermöglicht gleichen Aufbau der Datenbank auf unterschiedlichen Plattformen
- Steuerung der Verfügbarkeit der Datenbank (durch Online bzw. Offline-Setzen von Tablespaces)
- Durchführung partieller Backup- und Recovery-Operationen

**Tabelle**

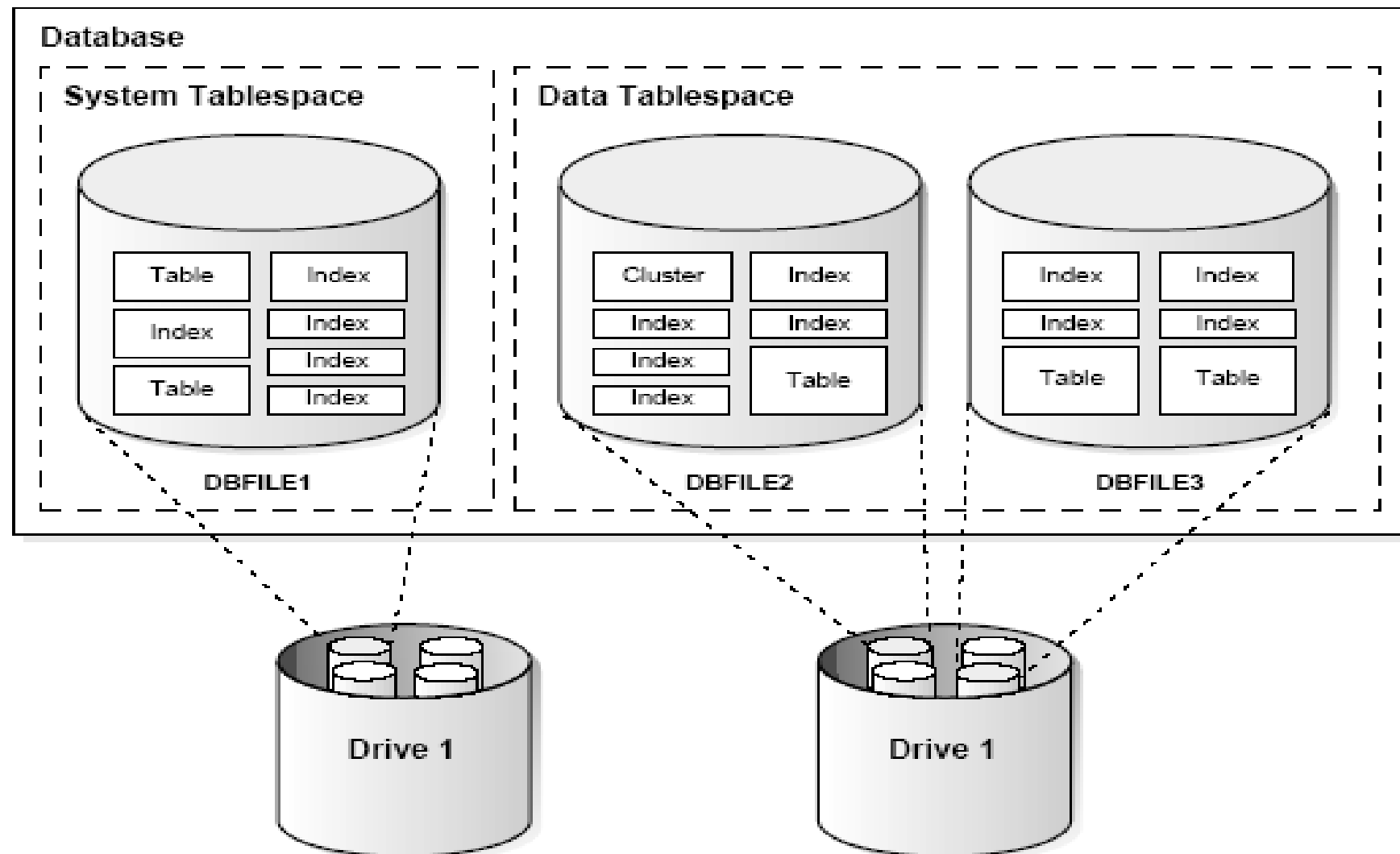
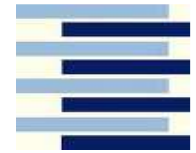
**Index**

**Tabelle**

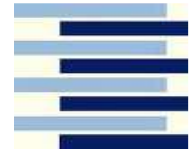


**Tablespaces**  
Modellierung

# Tablespace



# Überlegungen zu Tablespaces



1. Tabellen mit ähnlichen Eigenschaften in einem Tablespace zusammenfassen. Zum Beispiel Read-only Tabellen in einem Read-only Tablespace.
2. Mindestens ein Tablespace für Temp-Dateien.
3. Tablespaces überschaubar lassen. Empfehlung Oracle: Tablespace < 10 GB. Hinweis: Man kann eine Tabelle auf mehrere Tablespaces verteilen.
4. System-Tabellen in einem eigenen SYSTEM-Tablespace.
5. Rollback-Segmente in einem eigenen UNDO Tablespace.

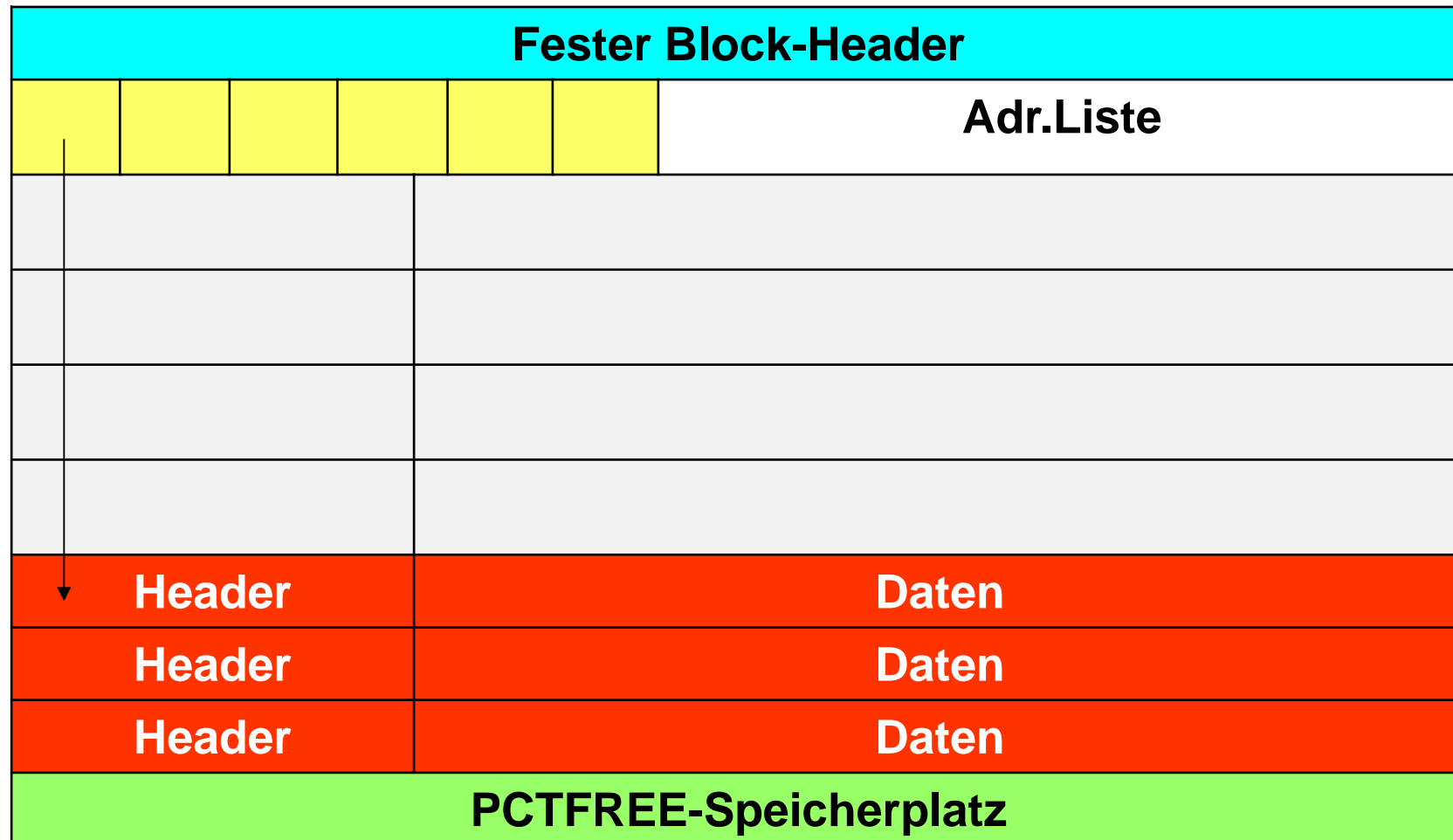
**Hinweis: Datenbankdateien, RedoLog-Dateien und Online-Archive nie auf dem selben Laufwerk.**

Beispiel: Create Tablespace Spielwiese

Datafile 'swt05/oradata/prod/spielwiese.dbf' size 10M

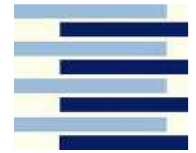
Default storage (initial 50K next 100K minextents 1 maxextents 4  
pctincrease 0)

# Aufbau eines Blocks



Länge von Attributen:    DATE: 7 Byte, CHAR(n): n  
 VARCHAR2(n), NUMBER: var.

# Create table



```
CREATE TABLE [schema.]table [table_constraint]
(column datatype [DEFAULT expr][column_constraint]
| table_constraint), ...)
[TABLESPACE tablespacename
[PCTFREE n1] [PCTUSED n2]
[STORAGE (INITIAL m1 NEXT m2 ...) ] ]
[CLUSTER cluster (cluster, ...)]
[PARTITION BY ... (Partition ...) ]
```

**PCTFREE:** reservierter Platz für Updates mit Satzverlängerungen

**PCTUSED:** neue Datensätze können nur eingefügt werden, wenn freier Platz > n2

**INITIAL und NEXT:** Anfangsextend: Größe m1, neue Extents: Größe m2.

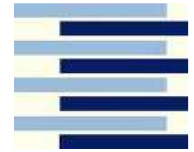
# Platzbedarf einer Tabelle (für Oracle)



Variable	Bedeutung	Beispiel
blksize	Blockgröße	2048 B
pctfree	Freiplatz im Block	20 %
initrans	Transaktionen pro Block muss bei Isolation level Serializable $\geq 3$ sein	1
avglen	durchschnittl. Satzlänge	25 Byte
colno1	Anzahl Spalten Länge $\leq 255$	3
colno2	Anzahl Spalten Länge $> 255$	0
Variable	Formel	im Bsp.
header	$3 + \text{colno1} + 3 * \text{colno2}$	6
rowlen	$\text{avglen} + \text{header}$	31
fixheader	$57 + 23 * \text{initrans}$	80
blkvar	$\text{Blksize} - \text{fixheader}$	1968
R	$(\text{blkvar} * (1 - \text{pctfree}) - 4) / (\text{rowlen} + 2)$	$R = 47$
blkanz	Zeilen / R	$20000/47 = 426$
initial	$\text{blkanz} * \text{blksize}$	852 K



# Automatische Schlüsselgenerierung



## RDBMS Oracle:

Create sequence Zaehler increment by 1;

Create table test (id numeric(5), ... );

Insert into test values (Zaehler.nextval, ...);

## RDBMS MySQL

Create table Kunden

(id number auto\_increment primary key,

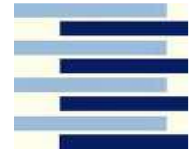
name varchar(30),

telefon char(10));

Insert into kunden (name, telefon)

values ('Asterix', '003712345');

## Automatische Schlüsselgenerierung (2)



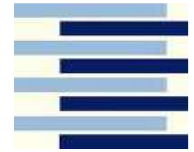
### RDBMS Oracle ab Version 12c:

```
CREATE TABLE test  
( id    NUMBER GENERATED BY DEFAULT ON NULL AS  
  IDENTITY,  
  text  VARCHAR2(30) );
```

```
INSERT INTO test (text) VALUES ('Nur Text');  
INSERT INTO test (id, text) VALUES (999, 'ID und Text');  
INSERT INTO test (id, text) VALUES (NULL, 'NULL und Text');
```

### Alternativ

```
CREATE TABLE test  
( id    NUMBER default zaehler.nextval primary key,  
  text  VARCHAR2(30) );
```



**PL/SQL ist eine prozedurale Oracle-spezifische Erweiterung von SQL für Trigger, Stored Procedures und Objekt-Methoden**

**Deklarations-  
block**

```
declare
    Anzahl      number(5);
    ...
```

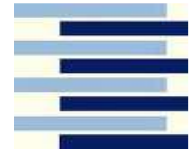
**Ausführungs-  
block**

```
begin
    select count(*) into Anzahl from Kunde;
    if Anzahl = 0 then dbms_output.put_line('Tabelle leer');
    else dbms_output.put_line(Anzahl || ' Tupel');
    end if;
```

**Fehler-  
behandlungs-  
block**

```
exception
    when no_data_found then
        dbms_output.put_line('Tabelle nicht da');
    when others then ... ;
end;
```

# Der Deklarationsteil



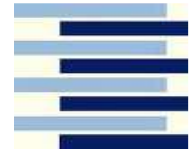
PL/SQL ist eine typenstrenge Programmiersprache. Deklarationen für:

- PL/SQL-lokale Variable,
- explizite SQL-Cursor für mengenorientierte Select-Befehle,
- benutzerdefinierte Fehlersituationen.

```
Variable_x      Varchar2(50);
Variable_y      Tabelle1.SpalteX%TYPE;
Variable_z      Variable_x%TYPE ;
TYPE eigener_Typ1 IS RECORD (   Nr      number(5);
                                Name varchar2(30) );

Struktur_x      eigener_Typ1;
Struktur_y      Tabelle1%ROWTYPE ;
Cursor c1       is select * from Student;
Var_c1          c1%ROWTYPE;
Tabelle_leer    EXCEPTION;
```

# Der Ausführungsteil



Der Ausführungsteil enthält:

- Zuweisungen, arithmetische Ausdrücke, Behandlung von Collections
- Bedingte Verarbeitung
- Schleifenkonstrukte
- SQL-Anweisungen
- Funktions- und Prozeduraufrufe

Var\_1:=0; Var\_2:=Var\_1;

**if** *bedingung* **then** *ja\_Befehle* **else** *nein\_Befehle* **end if**;

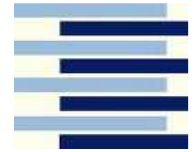
**loop** *Befehle* **exit when** *Bedingung* *Befehle* **end loop**;

**for** *i* **in** *Startwert .. Endwert* **loop** *Befehle* **end loop**;

**while** *Bedingung* **loop** *Befehle* **end loop**;

proc1(par1, par2); var\_Euro :=Euro(var\_DM) ;

# Beispiele



Declare

```
StGang      char(2):='AI';  
Anzahl      number(5);  
zu_viele    Exception;
```

Begin

```
select count(*) into Anzahl from Student where  
       Studiengang=StGang;  
if Anzahl > 400 then raise zu_viele; End if;
```

...

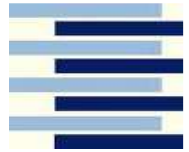
Exception

```
when zu_viele  
then insert into Logtabelle values(sysdate, StGang, Anzahl);
```

End;

```
For i in 1 .. 4 loop delete from Student where Semester = i;  
End loop;
```

# Cursor-Verarbeitung



## Verarbeiten von SQL-Ergebnismengen

1. Öffnen des Cursors
2. Lesen der Daten
3. Schließen des Cursors

Open c1;

Loop

    fetch c1 into var\_c1;

    Exit when c1%notfound;

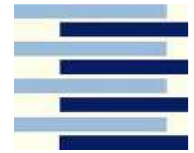
    Dbms\_output.put\_line(var\_c1.MatrNr || var\_c1.Name);

End loop;

Dbms\_output.put\_line(c1%rowcount);

Close c1;

# Fehlerbehandlung



## Mit Exception-Behandlung ...

Exception

```
when Ausnahme_1 then ... ;  
when too_many_rows then ... ;  
when others then Dbms_output.put_line(SQLCODE, SQLERRM) ;  
End;
```

standardmäßig  
definiert

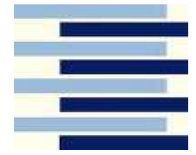
## ... oder direkt im Anweisungsteil

```
raise_application_error(-20001, 'Fehlertext') ; Programmabbruch !
```

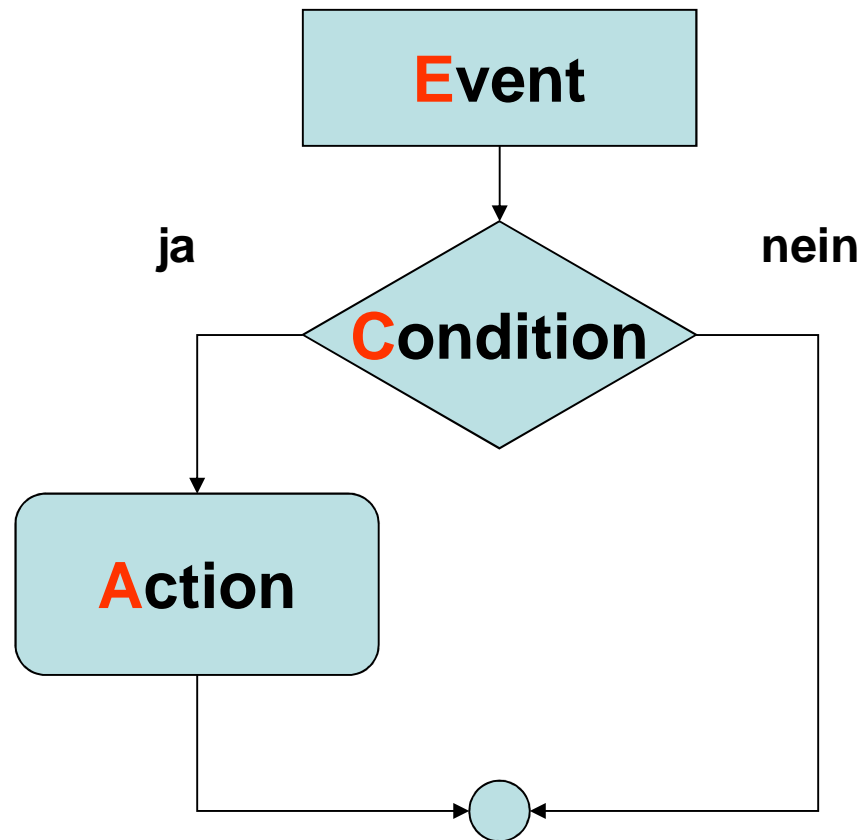
**Nach der Behandlung einer Exception wird der aktuelle Block verlassen und zum umgebenden Block zurückgekehrt. Die Verarbeitung wird nicht mit der der auslösenden Anweisung folgenden Anweisung fortgesetzt. D.h. kein goback.**



# Trigger

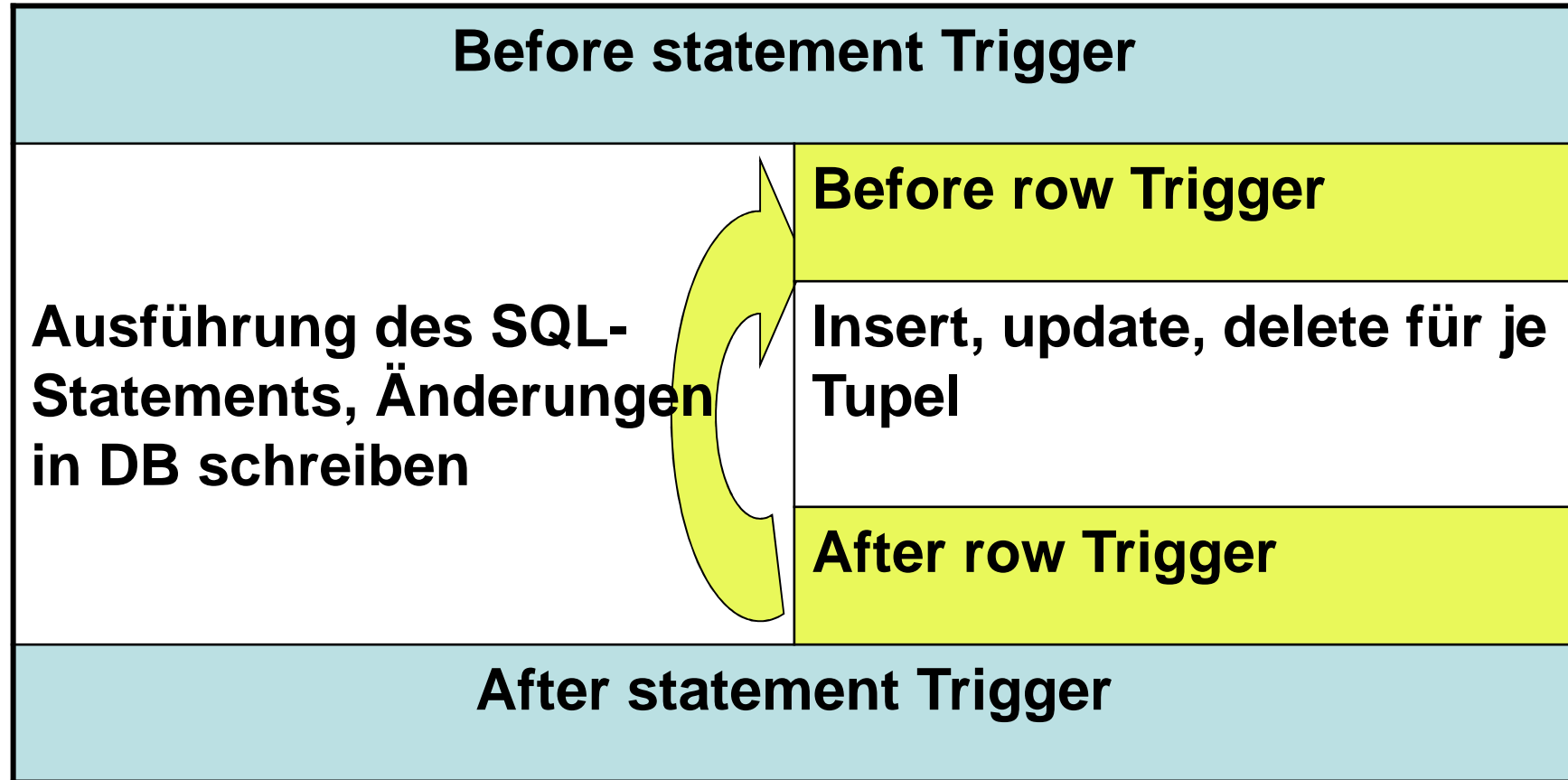
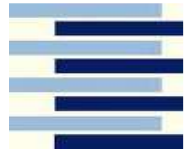


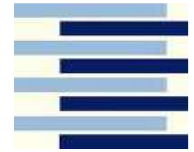
Ein Trigger ist eine Datenbankprozedur, die einer Tabelle zugeordnet, in der Datenbank gespeichert und bei Eintreten eines bestimmten DML-Ereignisses gestartet wird.



Für Trigger relevante Events sind:  
Insert,  
Update,  
Delete von Tupeln

# Arten von Triggern

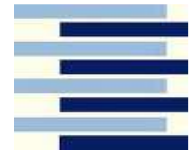




## Einige wichtige Hinweise

- Benutze before-update row-level Trigger für komplexe Geschäftsregeln, Security Checks und komplexe arithmetische Berechnungen.
- Benutze after-update row-level Trigger für Datenreplikationen und Logging von Updates.
- Benutze keine before Trigger für Replikationen und Logging.
- Benutze keine Trigger für deklarativ definierbare referenzielle Integrität.
- Benutze statement-level before Trigger für Security Regeln, die nicht von den Werten der einzelnen Tupel abhängen.
- Statement-level Trigger für alles, was nicht tupelabhängig ist.
- Wenn bei mehreren Triggern desselben Typs die Ausführungsreihenfolge wichtig ist, fasse sie als ein Trigger zusammen.
- Auch Trigger unterliegen dem Transaktionskonzept (Commit, Rollback)

# Syntax



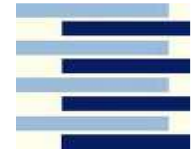
```
CREATE [OR REPLACE] TRIGGER trigger_name
    {BEFORE | AFTER } {INSERT | UPDATE | DELETE
    [OF column_list ] } [ OR ...] ON table_name
    [FOR EACH ROW [ WHEN (condition) ] ]
DECLARE
    Declarations
BEGIN
    PL/SQL-Code
END;
```

Auf ein altes bzw. neues Tupel kann, **allerdings nur bei ROW-Triggern**, mit :OLD und :NEW zugegriffen werden.

Bsp. :OLD.Name

Bei Insert sind die Attribute von :OLD null,  
bei Delete sind die von :NEW null

# Beispiel



Student(MatrNr, Name, DNr, ...)      Department(DNr, Bezeichnung, Anzahl, ...)  
*Trigger soll das Attribut Anzahl konsistent halten.*

```
CREATE TRIGGER student_upd
  AFTER UPDATE OR DELETE OR INSERT ON Student
  FOR EACH ROW
BEGIN
  IF :OLD.DNr IS NOT NULL THEN
    UPDATE Department SET Anzahl=Anzahl - 1
    WHERE Department.DNr=:OLD.DNr;
  END IF;
  IF :NEW.DNr IS NOT NULL THEN
    UPDATE Department SET Anzahl=Anzahl + 1
    WHERE Department.DNr=:NEW.DNr;
  END IF;
END;
```

**Abfrage möglich auf**  
If inserting  
If deleting  
If updating [<Spalte>]

# Wirkungsweise von before- und after-row Triggern

```
create table Konto(Kontonr number(5) primary key,  
                  Saldo number(9,2),  
                  Tagessumme number(7,2),  
                  Tageslimit number(7,2),  
                  letzterTag date);
```

Enthält die Auszahlungssumme an diesem Tag

```
insert into Konto values(1, 2000, 1000, 2000, '14.01.2013');
```

```
create or replace trigger Vor_Auszahlung
```

```
before update on konto for each row
```

```
begin   if :old.letzterTag < current_date then :new.Tagessumme := 0;  
        end if;
```

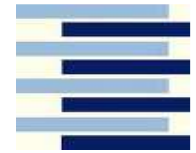
```
end Vor_Auszahlung;
```

Z. B. Current\_date =  
15.1.2013

```
update konto set saldo=saldo -1000,  
            tagessumme=tagessumme+1000,  
            letzterTag=current_date
```

```
where kontoNr=1;
```

# Und das Ergebnis



PL/SQL Developer - gerken@SWT02

File Project Edit Session Debug Tools Macro Documents Reports Window Help

My objects

- EUL5\_OBJS
- EUL5\_PLAN\_TABLE
- EUL5\_QPP\_STATS
- EUL5\_SEGMENTS
- EUL5\_SEQUENCES
- EUL5\_SQ\_CRLTNS
- EUL5\_SUB\_QUERIES
- EUL5\_SUM\_BITMAPS
- EUL5\_SUMMARY\_OBJS
- EUL5\_SUMO\_EXP\_USGS
- EUL5\_SUM\_RFSH\_SETS
- EUL5\_VERSIONS
- FB
- FRIENDS
- KONTAKTE
- KONTO
  - Columns
    - KONTONR
    - SALDO
    - TAGESSUMME
    - TAGESLIMIT
    - LETZTERTAG
  - Primary key
  - Unique keys
  - Foreign keys
  - Check constraints
  - Triggers
  - Indexes
  - Foreign key references

Templates

- Constants
- Default
- DML statements
- Error handling

SQL Window - update\_Konto.sql

SQL Output Statistics

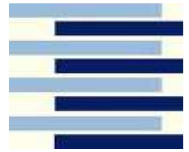
```
select * from konto;
```

	KONTONR	SALDO	TAGESSUMME	TAGESLIMIT	LETZTERTAG
▶ 1	1	1000,00	0,00	2000,00	15.01.2003 10:44:44

1:21 1 row selected in 0,03 seconds

Start PL/SQL Developer - g... Dokument2 - Microsoft ... 11:00

## Und warum ist das so?



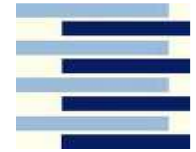
### Abfolge der Ausführung

1. :OLD und :NEW durch Befehl berechnen
2. Before-Row Trigger ausführen, evtl. :NEW ändern
3. Änderung gemäß :NEW in Datenbank speichern
4. After-Row Trigger ausführen

### Zu beachten

- Auf :OLD und :NEW kann ein Statement-level Trigger nicht zugreifen.
- :OLD kann ein Before Row-level Trigger nicht ändern.
- :NEW kann ein After Row-level Trigger nicht ändern.
- Row-level Trigger können andere Zeilen der Tabelle nicht ändern.





# Instead-of Trigger

## Instead-of Trigger ersetzen DML-Operationen auf Views

```
Create table tab1 ( t1Nr number(10) primary key, t1Text varchar2(30) );
```

```
Create table tab2 ( t2Nr number(3) primary key, t2Text char(1),  
                  t1FK number(10) not null references tab1 );
```

```
Create View V12 as
```

```
Select t1Nr, t1Text, t2Text from tab1, tab2 where t1Nr=t1FK
```

```
insert into V12 values(1,'Text für Tabelle 1', 'X');
```

**Fehler: Kann keine Spalte, die einer Basistabelle zugeordnet wird, verändern**

```
Create sequence SeqTab2 increment by 1;
```

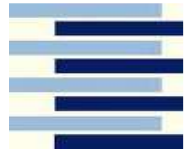
```
Create or replace trigger tr_V12 instead of insert on V12
```

```
Begin insert into tab1 values(:new.t1Nr, :new.t1Text);
```

```
        Insert into tab2 values(SeqTab2.nextval, :new.t2Text, :new.t1Nr);
```

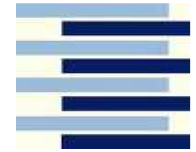
```
End;
```

# Trigger für Primary-Keys



```
create or replace TRIGGER Teile_PK
BEFORE INSERT ON Teile FOR EACH ROW
declare maxnr number;
BEGIN
    select max(teilenr) into maxnr from teile;
    if maxnr is null then maxnr := 0;
    end if;
    maxnr := maxnr +1;
    :new.teilenr := maxnr;
END;
```

# Trigger und Transaktionen



```
create table Kunden(KdNr number(5) primary key, name varchar2(10));  
create table KdNummern(KdNr number(5) primary key);
```

```
create or replace trigger KdNrTrigger after insert on Kunden for each row  
Begin    insert into KdNummern values(:new.KdNr);  
         dbms_output.put_line('neue Kundennummer ' || :new.KdNr);  
end;
```

```
insert into kunden values(1,'Otto');  
insert into kunden values(2,'Ottilie');  
commit;
```

neue Kundennummer 1  
neue Kundennummer 2  
Transaktion mit COMMIT abgeschlossen.

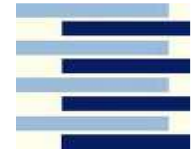
```
insert into kunden values(3,'Donald');  
insert into kunden values(4,'Daisy');  
rollback;
```

neue Kundennummer 3  
neue Kundennummer 4  
Transaktion mit ROLLBACK rückgängig gemacht

```
select * from KdNummern;
```

<u>KDNR</u>
1
2

# Trigger und Transaktionen ...



Gegeben seien wieder Kunden(KdNr, Name) und KdNummern(KdNr)


create or replace trigger KdNrTrigger after insert on Kunden for each row

```
Begin  insert into KdNummern values(:new.KdNr);
      dbms_output.put_line('neue Kundennummer ' || :new.KdNr);
      if :new.KdNr > 20 then Raise_application_error(-20001,'KdNr zu gross');
      end if;
end;
```

insert into kunden values (13,'Plisch');

1 Zeile wurde erstellt.

insert into kunden values (33,'Plumm')



KDNR	NAME
1	Otto
2	Otilie
13	Plisch

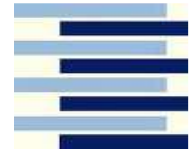
FEHLER in Zeile 1:

ORA-20001: KdNr zu gross

ORA-06512: in "SCOTT.KDNRTRIGGER", Zeile 3

ORA-04088: Fehler bei der Ausführung von Trigger 'SCOTT.KDNRTRIGGER'

# Trigger und Select „auf sich selbst“ (1)



```
create table Studenten  
(MatrNr number(5) primary key, Name VarChar2(20), StudGang Char(2));
```

## ***Nicht mehr als 30 Studierende pro Studiengang***

Create or replace trigger Anzahl\_Stud **before insert** on Studenten  
for each row

```
Declare Anzahl number;
```

```
Begin  select count(*) into Anzahl from Studenten  
       where StudGang=:NEW.StudGang;
```

```
If Anzahl > 29 then
```

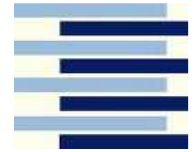
```
raise_application_error(-20001, 'zu viele Studierende') ;
```

```
End if;
```

```
End;      /* Statement processed */
```

```
INSERT into Bestellung values(200, 'Petermann', 'WI');
```

## Trigger und Select „auf sich selbst“ (2)



create table Bestellung

(BestellNr number(5) primary key, KdNr number(5), datum date);

Column Name	Null?	Type
BESTELLNR		NUMBER(5)
KDNR		NUMBER(5)
DATUM		DATE

desc  
Bestellung;

Create or replace trigger Anzahl\_tr **after insert** on Bestellung for each row

Declare Anzahl number;

Begin select count(\*) into Anzahl from Bestellung where KdNr = :new.KdNr;

Dbms\_output.put\_line('Anzahl Bestellungen von ' || :new.KdNr || ': ' || Anzahl);

End; /\* Statement processed \*/

**INSERT into Bestellung values(200, 101, '12.04.2012');**

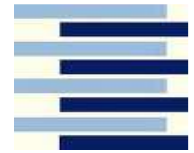
ORA-04091: Tabelle BESTELLUNG wird gerade geändert,

Trigger/Funktion darf es nicht sehen

ORA-04088: Fehler bei der Ausführung von Trigger 'Anzahl-tr'

engl.  
mutating table

# Was kann man da machen?



```
create table temp(KNr number(5));
```

```
create trigger t1 before insert on Bestellung
begin          delete from temp;
end;
```

```
create trigger t2 after insert on Bestellung for each row
begin  insert into temp values(:New.KdNr);
end;
```

```
create or replace trigger t3 after insert on Bestellung
declare Cursor c1 is select * from temp;
          Var_c1 c1%ROWTYPE; anz number(5);
```

```
Begin
```

```
Open c1;
```

```
Loop  fetch c1 into var_c1;
```

```
      Exit when c1%notfound;
```

```
      select count(*) into anz from Bestellung where KdNr=var_c1.KNR;
```

```
      Dbms_output.put_line(var_c1.KNR || ': ' || ANZ);
```

```
End loop;
```

```
end;
```



oder  
Compound  
Trigger

## (Noch ein) Beispiel für einen fehlerhaften Trigger



Relation Kunden (KdNr number(5) primary key,  
Name varchar2(20) not null,  
Typ char(2) null);

Create or replace trigger Kunden\_Ins before insert on Kunden for each row  
Begin  
if Typ = null then insert into Kunden values (:new.KdNr, :new.name, '99');  
end if;  
End;

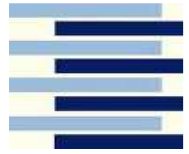
*Was ist da falsch?*  
1. ? 2.?

Create procedure Kunden\_Ins (PKdNr number, PName varchar2, PTyp char)  
As  
Begin if PTyp is null then PTyp = '99'; end if;  
insert into Kunden values (PKdNr, PName, PTyp);  
End;

**Statt insert-Befehl SP, dann insert sperren**



# Stored Procedure



**Stored Procedures sind im DB-Server gespeicherte und auszuführende PL/SQL-Programme. Sie werden, im Gegensatz zu Triggern, explizit gestartet.**

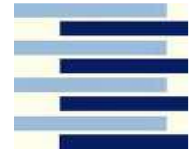
```
create or replace procedure mySP(Name in out type, , ...) is  
    Definition lokaler Variablen;  
begin  
    PL/SQL-Code;  
end mySP;
```

Vor der Definition von lokalen Variablen kein DECLARE (im Ggs. zu Triggern)

## Parameter:

1. Ohne Längenangaben bei Number, Varchar2, ...
2. Ein- und Ausgabeparameter, IN, OUT, INOUT
3. Standardmodus ist IN

# Beispiele

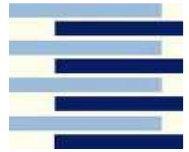


```
create table test1(a number(5), b varchar2(300));  
create or replace procedure insproc(p1 in number, p2 in varchar2) as  
begin  
    insert into test1 values(p1, p2);  
end;  
  
create or replace procedure anzproc(p out number) as  
begin  
    select count(*) into p from Student;  
end;
```

Aufruf mit      `Begin mySP; end;`      oder  
                 `execute mySP;`

Aufrufe in einem anderen PL/SQL-Programm mit  
                 `Test1(akt1, akt2);`  
                 `Anzproc(anzahl);`

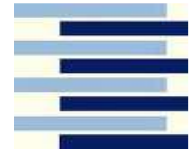
# Ausgabe in Stored Procedures



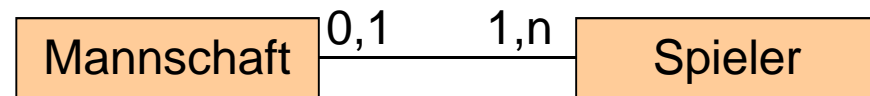
Stored Procedures ermöglichen keine direkte Ausgabe von SQL-Select. Die Ausgabe muss mit einem Cursor und DBMS\_Output entsprechend dem folgenden Beispiel durchgeführt werden.

```
create or replace procedure alltabs as
    ax      test1.a%type;
    bx      test1.b%type;
    cursor csr is select a, b from test1;
begin
    open csr;
    loop
        fetch csr into ax, bx;
        exit when csr%notfound;
        dbms_output.put_line(ax || ' ' || bx);
    end loop;
    close csr;
end;
```

# Anwendung von Stored Procedures



Create table Mannschaft (  
MannNr number(5) primary key,  
Bezeichnung varchar2(30) );



Create table Spieler ( SpielerNr number(5) primary key,  
Name varchar2(30),  
Mannschaft number(5) references Mannschaft on delete set null);

Not null bei (1,1)

```
Create Procedure NeueMannschaft
(MNr in number, MBez in varchar2, EinSpieler in number)
As      Anzahl number;
Begin
Select count(*) into Anzahl from Spieler where SpielerNr=EinSpieler
      and Mannschaft is null;
If Anzahl = 0 then
      Raise_application_error(-20001,'Kein freier Spieler vorhanden');
else insert into Mannschaft values(MNr, MBez);
      update Spieler set Mannschaft=MNR where SpielerNr=EinSpieler;
end if;
End;
```