Auswertungen und „Programmieren“
Views und DB-Trigger mit der SQL

- Mithilfe von Trigger - Anweisungen lassen sich komplexe semantische Integritätsbedingungen definieren, die zu automatischen Veränderungen der Datenbank führen können

- Mit **CREATE TRIGGER** können sogenannte Trigger-Befehle hinzugefügt werden.

- Trigger-Befehle werden automatisch ausgeführt, wenn die entsprechenden Ereignisse eintreten

- Das „Feuern“ eines Triggers kann definiert werden für **DELETE**, **INSERT**, oder **UPDATE** einer DB-Tabelle,

  oder wenn ein **UPDATE** für eine oder mehrere Spalten einer Tabelle geschieht

Auswertungen und „Programmieren“
Views und DB-Trigger mit der SQL

Beispiel:

Es sollen Bestellungen von Produkten in einer Tabelle erfasst werden und dabei jeweils geprüft werden, ob die benötigten Rohstoffe vorhanden sind.

Im weiteren werden auf der Basis von SQL unterschiedliche Möglichkeiten der Programmierung gezeigt.

Wir verzichten dabei auf die Produktnamen und arbeiten mit der Produktnummer, um die notwendigen Join-Anweisungen klein zu halten.

Erzeugen einer Tabelle Bestellung

```
DROP TABLE IF EXISTS Bestellung;

CREATE TABLE Bestellung (     Kunde char(20), Pnr INT, Anz INT,
                              FOREIGN KEY (Pnr) REFERENCES Produkt,
                              PRIMARY KEY (Kunde, Pnr )
                              );

INSERT INTO Bestellung (Kunde, PNR, Anz) Values ('Best', 1,70);
```

Ergebnis:

```
----------
Kunde     Pnr   Anz
Best      1     70
```

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

Bedarf an Rohstoffen für eine Bestellung:

```
SELECT pr.rnr,pr.menge FROM bestellung,produkt,pr
where bestellung.pnr=produkt.pnr
and produkt.pnr=pr.pnr;
```

Ergebnis:

```
---------
RNR  MENGE
1    3
2    1
```

Bedarf an Rohstoffen für eine Bestellung mit Berücksichtigung der Auftragsmenge:

```
SELECT pr.rnr,SUM(pr.menge*Anz) FROM bestellung,produkt,pr
where bestellung.pnr=produkt.pnr
and produkt.pnr=pr.pnr
GROUP BY rnr
```

Ergebnis:

```
RNR  SUM(pr.menge*Anz)
1    210
2    70
```


Fachhochschule Dortmund
University of Applied Sciences and Arts

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

Bestände in den Lagern:

```
SELECT Rnr,SUM(Menge) FROM lr  GROUP BY Rnr
```

Ergebnis:

```
---------
RNR  SUM(Menge)
1    1100
2    300
3    800
4    200
5    1300
6    400
```

Verbleibende Bestände in allen Lagern nach Abzug der Bestellung
(!!! Falsches Ergebnis - aus PR werden alle PNR berücksichtigt):

```
SELECT pr.rnr,Rname,SUM(lr.menge-pr.menge*Anz) AS wert FROM bestellung,produkt,pr,lr,rohstoff
where bestellung.pnr=produkt.pnr   and   pr.rnr = rohstoff.rnr
and produkt.pnr=pr.pnr   and   pr.rnr = lr.rnr
GROUP BY pr.rnr
```

Ergebnis:

```
---------
RNR  RNAME     wert
1    Glutin    470
2    Olefin    160
```

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

Besseres Vorgehen über sogenannte VIEWS:

1) VIEW R_Bestand

```
DROP View IF EXISTS R_Bestand;

CREATE VIEW R_Bestand AS
SELECT Rnr,SUM(Menge) AS vorhanden FROM lr  GROUP BY Rnr;

SELECT * FROM R_Bestand
```

Ergebnis:

```
----------
RNR  vorhanden
1       1100
2       300
3       800
4       200
5       1300
6       400
```

- **Vorteile von Views:**
  - **Vereinfachung von Anfragen für den Benutzer durch Realsierung von Teilanfragen als View**

Fachhochschule Dortmund
University of Applied Sciences and Arts

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

2) VIEW R_Bedarf:

```
DROP View IF EXISTS R_Bedarf;

CREATE VIEW R_Bedarf AS
SELECT pr.rnr,SUM(pr.menge*Anz) AS angefragt FROM bestellung,produkt,pr
WHERE bestellung.pnr=produkt.pnr and produkt.pnr=pr.pnr
GROUP BY rnr ;

SELECT * FROM R_Bedarf
```

Ergebnis:

```
----------
RNR  angefragt
1       210
2       70
```

3) Differenztabelle:

```
SELECT R_Bestand.rnr, (vorhanden-angefragt) AS wert FROM R_Bestand,R_Bedarf
WHERE R_Bestand.rnr=R_Bedarf.rnr
GROUP BY R_Bestand.rnr
```

Ergebnis:

```
----------
RNR  wert
1       890
2       230
```

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

Bedarfsliste für den Einkauf der Rohstoffe erzeugen:

```
CREATE TABLE Einkauf (
        Rnr INT,
        Anz INT,
        FOREIGN KEY (Rnr) REFERENCES Rohstoff,
        PRIMARY KEY (Rnr)
);

INSERT INTO Einkauf (Rnr, Anz) Values (1, 0);
INSERT INTO Einkauf (Rnr, Anz) Values (2, 0);
INSERT INTO Einkauf (Rnr, Anz) Values (3, 0);
INSERT INTO Einkauf (Rnr, Anz) Values (4, 0);
INSERT INTO Einkauf (Rnr, Anz) Values (5, 0);


SELECT * FROM Einkauf
```

Ergebnis:

Rnr	Anz
1	0
2	0
3	0
4	0
5	0

Fachhochschule Dortmund
University of Applied Sciences and Arts

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

Erster einfacher DB-Trigger:

```
DROP TRIGGER If EXISTS trig1;

CREATE TRIGGER trig1 AFTER INSERT ON Bestellung
BEGIN
        UPDATE einkauf SET Anz= -1
        WHERE einkauf.rnr = NEW.pnr;
END;
```

NEW meint den INSERT-Tuple,
… . rnr = … .pnr dient hier nur als BSP !

```
SELECT * FROM Einkauf
```

Ergebnis:

Rnr	Anz
1	0
2	0
3	0
4	0
5	0

```
INSERT INTO Bestellung (Kunde, PNR, Anz) Values ('Best2', 1,80);

SELECT * FROM Einkauf
```

Ergebnis:

Rnr	Anz
1	-1
2	0
3	0
4	0
5	0

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

DB-Trigger der ein RAISE erzeugt und den Tuple-Eintrag verhindert:

```
CREATE TRIGGER trig2 AFTER INSERT ON Bestellung
BEGIN
 SELECT
   CASE
    WHEN NEW.Anz > 200 THEN RAISE(ABORT, 'zu gross')
   END;
 SELECT
   CASE WHEN NEW.anz <=100 THEN RAISE(ABORT, 'zu klein')
 END;
END;
```

```
INSERT INTO Bestellung (Kunde, PNR, Anz) Values ('Best3', 2,80);
```

Ergebnis:
---
java.sql.SQLException: zu klein

Rnr	ANZ
1	-1
2	0
3	0
4	0
5	0

```
INSERT INTO Bestellung (Kunde, PNR, Anz) Values ('Best4', 2, 120);
```

Ergebnis:

Rnr	ANZ
1	-1
2	-1 ← DB-Trigger 1 wirkt nun
3	0
4	0
5	0

Fachhochschule Dortmund
University of Applied Sciences and Arts

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

Update multiple rows

```
UPDATE Einkauf SET anz = -2  WHERE anz < 0;
```

Ergebnis:

Rnr	ANZ
1	-2
2	-2
3	0
4	0
5	0

Auswertungen und „Programmieren“
Views und DB-Trigger mit der SQL

Einkaufsliste für fehlende Rohstoffe mittels Update für multiple Zeilen erstellen:

1. Bedarf als View ermitteln:

```
DROP View IF EXISTS R_Bedarf;

CREATE VIEW R_Bedarf AS
SELECT pr.rnr,SUM(pr.menge*Anz)
AS angefragt
FROM bestellung,produkt,pr
WHERE bestellung.pnr=produkt.pnr
and produkt.pnr=pr.pnr
GROUP BY rnr ;

SELECT * FROM R_Bedarf
Ergebnis:
--------
RNR angefragt
1       690
2       150
4       240
5       120
```

Bestand als View ermitteln:

2. Bestand als View ermitteln

```
drop View if exists R_Bestand;

CREATE VIEW R_Bestand AS
SELECT Rnr,SUM(Menge)as vorhanden
FROM lr
Group by Rnr;

SELECT * FROM R_Bestand
Ergebnis:
--------
RNR vorhanden
1       1100
2       300
3       800
4       200
5       1300
6       400
```


Fachhochschule Dortmund
University of Applied Sciences and Arts

Auswertungen und „Programmieren“
Views und DB-Trigger mit der SQL

Einkaufsliste für fehlende Rohstoffe mittels Update für multiple Zeilen erstellen:

3. Aktuelle Differenztabelle

```
SELECT R_Bestand.rnr, (vorhanden-angefragt) AS wert FROM R_Bestand,R_Bedarf
WHERE R_Bestand.rnr=R_Bedarf.rnr
GROUP BY R_Bestand.rnr

Ergebnis:
----------
RNR wert
1       410
2       150
4       -40
5       1180
```

3. Einkaufsliste für fehlende Rohstoffe

```
SELECT R_Bestand.rnr, (angefragt-vorhanden) as wert
FROM R_Bestand,R_Bedarf
WHERE R_Bestand.rnr = R_Bedarf.rnr
and wert > 0

Ergebnis:
----------
RNR wert
4       40
```

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

DB-Trigger zur Erstellung der Einkaufsliste:

```
CREATE TRIGGER trig4 AFTER INSERT ON Bestellung
BEGIN
  UPDATE Einkauf SET anz=
    (SELECT (angefragt-vorhanden) AS wert
     FROM R_Bestand,R_Bedarf
     WHERE R_Bestand.rnr = R_Bedarf.rnr
     and R_Bestand.rnr=Einkauf.rnr
     and wert > 0);
END;
```

```
INSERT INTO Bestellung (Kunde, PNR, Anz) Values ('Midle2', 2,140);
INSERT INTO Bestellung (Kunde, PNR, Anz) Values ('Midle3', 2,140);
```

Einkaufsliste an fehlenden Rohstoffen

```
SELECT R_Bestand.rnr, (angefragt-vorhanden) as wert
FROM R_Bestand,R_Bedarf
WHERE R_Bestand.rnr = R_Bedarf.rnr
and wert > 0
```

Ergebnis:

```
---------
RNR wert
1   150
4   600
```

```
SELECT * FROM Einkauf
```

Ergebnis:

```
---------
Rnr  ANZ
1    150
2    -1
3    null
4    600
5    null
```

DB-Trigger 4 wirkt

DB-Trigger 1 wirkt

Fachhochschule Dortmund
University of Applied Sciences and Arts

Auswertungen und „Programmieren"
Views und DB-Trigger mit der SQL

Internes:

```
select * from sqlite_master
```

Ergebnis:

```
type    name    tbl_name        rootpage        sql
table   ROHSTOFF        ROHSTOFF        4       CREATE TABLE ROHSTOFF (RNR INT, RNAME CHAR(20), RCODE
CHAR(1), GEBINDE CHAR(20), PRIMARY KEY(RNR))
index   sqlite_autoindex_ROHSTOFF_1     ROHSTOFF        5       null
table   PRODUKT         PRODUKT         6       CREATE TABLE PRODUKT (PNR INT, PNAME CHAR(20), ORT
CHAR(20), PREIS FLOAT, PRIMARY KEY(PNR))
index   sqlite_autoindex_PRODUKT_1      PRODUKT         7       null
table   LR      LR      8       CREATE TABLE LR (LNR INT, RNR INT, MENGE INT, BWERT FLOAT, PRIMARY
KEY(LNR,RNR), FOREIGN KEY (LNR) REFERENCES LAGER, FOREIGN KEY (RNR) REFERENCES ROHSTOFF)
index   sqlite_autoindex_LR_1   LR      9       null
table   PR      PR      10      CREATE TABLE PR (PNR INT, RNR INT, MENGE INT, PRIMARY KEY(PNR,RNR), FOREIGN
KEY (PNR) REFERENCES PRODUKT, FOREIGN KEY (RNR) REFERENCES ROHSTOFF)
index   sqlite_autoindex_PR_1   PR      11      null
table   LAGER   LAGER   2       CREATE TABLE LAGER (LNR INT, ORT CHAR(20), LCODE CHAR(1), MENGE INT,
PRIMARY KEY(LNR))
index   sqlite_autoindex_LAGER_1        LAGER   3       null
table   Bestellung      Bestellung      13      CREATE TABLE Bestellung (Kunde char(20), Pnr INT, Anz INT,
FOREIGN KEY (Pnr) REFERENCES Produkt,
PRIMARY KEY (Kunde, Pnr )
)
index   sqlite_autoindex_Bestellung_1   Bestellung      14      null

                                        view    R_Bestand       R_Bestand       0       CREATE VIEW R_Bestand AS
                                        SELECT Rnr,SUM(Menge)as vorhanden FROM lr  Group by Rnr
                                        view    R_Bedarf        R_Bedarf        0       CREATE VIEW R_Bedarf AS
                                        SELECT pr.rnr,SUM(pr.menge*Anz)as angefragt FROM bestellung,produkt,pr
                                        where bestellung.pnr=produkt.pnr
                                        and produkt.pnr=pr.pnr
                                        Group by rnr
                                        table   Einkauf Einkauf 16      CREATE TABLE Einkauf (
                                                Rnr INT,
                                                ANZ INT,
                                                FOREIGN KEY (Rnr) REFERENCES Rohstoff,
                                                PRIMARY KEY (RNr)
                                        )
                                        index   sqlite_autoindex_Einkauf_1      Einkauf 17      null
```

Auswertungen und „Programmieren“ (DB-Trigger) mit der SQL

Fachhochschule Dortmund
University of Applied Sciences and Arts

Internes:

select * from sqlite_master

```
trigger  trig1    Bestellung      0       CREATE TRIGGER trig1 AFTER INSERT ON Bestellung
BEGIN
        UPDATE einkauf SET Anz= -1
    WHERE einkauf.rnr = NEW.pnr;
END

trigger  trig2    Bestellung      0       CREATE TRIGGER trig2 AFTER INSERT ON Bestellung
BEGIN
 SELECT
   CASE
    WHEN NEW.Anz > 200 THEN RAISE(ABORT, 'zu gross')
   END;
 SELECT
   CASE WHEN NEW.anz <=100 THEN RAISE(ABORT, 'zu klein')
 END;
END

trigger  trig4    Bestellung      0       CREATE TRIGGER trig4 AFTER INSERT ON Bestellung
BEGIN
  UPDATE Einkauf SET anz=
    (SELECT (angefragt-vorhanden) AS wert
     FROM R_Bestand,R_Bedarf
     WHERE R_Bestand.rnr = R_Bedarf.rnr
     and R_Bestand.rnr=Einkauf.rnr
     and wert > 0);

END

----------
```


Fachhochschule Dortmund
University of Applied Sciences and Arts

Auswertungen und „Programmieren“ Views und DB-Trigger mit der SQL

- Mithilfe von View - Anweisungen lassen sich aber auch komplexe Abfragen zu Inhalten der Datenbank realisieren
- Mithilfe von Trigger - Anweisungen lassen sich aber auch komplexe Seiteneffekte ausführen, die zu Veränderungen der Datenbank führen
  - Dies haben wir gerade bei multiplen Triggern für eine Aktion gesehen
  - Dies ist kritisch in Bezug auf mögliche Fehler in der Programmierung
  - Es sollten deshalb besser keine multiplen Trigger für eine **`DELETE -`**, **`INSERT -`** oder **`UPDATE`** - Funktionalität auf eine DB-Tabelle definiert werden

Fachhochschule Dortmund
University of Applied Sciences and Arts

Zusammenfassung Grundlegender Datenbank-Entwurf

- Ziel: Erstellung eines „konzeptuellen Schemas“
- **1. Problemanalyse**
  - Die „Miniwelt“ wird durch Elementaraussagen in **Tabellenform** detailliert beschrieben.
- **2. Strukturanalyse**
  - Aus den Elementaraussagen werden die Strukturbestandteile - Objekttypen,Attribute und Beziehungstypen - der Datenbank ermittelt.
  - Einsatz des **ER-Modells**
- **3. Festlegen der Elementarstruktur**
  - Definition der **Elementarrelationen**

Fachhochschule Dortmund
University of Applied Sciences and Arts

Zusammenfassung Grundlegender Datenbank-Entwurf

- 4. Normalisierte Relationen
  - Die Relationen aus Schritt 3 sind auf **Normalisierungskriterien** zu prüfen und unter Umständen aufzuspalten
- 5. Aggregierte Relationen
  - Relationen mit gleichem Primärschlüssel werden zu **einer Relation** zusammengefasst

> **Iteration der Schritte 4 und 5**
>
> Die Schritte 4 und 5 sind ggf. iterativ durchzuführen, bis alle Relationen normalisiert sind und unterschiedliche Primärschlüssel aufweisen.

Zusammenfassung Grundlegender Datenbank-Entwurf

Fachhochschule Dortmund
University of Applied Sciences and Arts

- 6. Abfragen und Änderungen an eine Datenbank werden formuliert
  - **SQL-Befehle** z.B. `SELECT`, `INSERT`, `UPDATE` ,u.a
- 7. Integritätsbedingungen
  - referentielle und semantische Integritätsbedingungen als **Regeln** an die DB werden definiert
- 8. Definition von Views
  - Realisieren von komplexeren Abfragen an die DB mit **Zwischenergebnissen**
- 9. Definition von Triggern
  - Programmieren **automatischer Reaktion** der Datenbank bei Veränderungen

Qualitätsanforderungen

Fachhochschule Dortmund
University of Applied Sciences and Arts

- Schnelligkeit
  - Jede neue Tabelle erfordert einen neuen indirekten Zugriff und eventuelle Join-Befehle
  - Dies trägt zur Erhöhung der Laufzeit von Anfragen an die Datenbank bei
- Konsistenz und Redunzfreiheit
  - Durch die Aufspaltung von Relationen auf verschiedene einzelne Relationen, um wie gezeigt die Einhaltung der Normalformen zu gewährleisten, wird
    - Redundanzfreiheit gewährleistet und
    - die Erhaltung einer konsistenten Datenbank erleichtert.
- Hinweise zu SQL
  - https://www.w3schools.com/sql/