

Datenbanken und SQL

Kapitel 5

Die Datenbankbeschreibungssprache SQL

Die Datenbankbeschreibungssprache SQL

- ▶ Relationen erzeugen, ändern und löschen
- ▶ Temporäre Relationen
- ▶ Sichten erzeugen und löschen
- ▶ Zusicherungen, Gebiete
- ▶ Trigger
- ▶ Sequenzen
- ▶ Zugriffsrechte und Zugriffsschutz
- ▶ Integrität
- ▶ Aufbau einer Datenbank
- ▶ Einrichten einer Datenbank

Befehl CREATE TABLE

```
CREATE TABLE Tabellenname  
  ( Spaltenname gefolgt von Datentyp  
    { Spalte { Datentyp | Gebietsname } [ Spaltenbedingung [ ... ] ]  
    | Tabellenbedingung }  
    [ , ... ] alternativ: Tabellenbedingung  
  )  
  [ Herstellerspezifische Optionen ] Beliebig viele Spalten,  
                                            durch Kommata getrennt
```

- ▶ Es wird eine neue Basisrelation erzeugt

Beispiel zu CREATE TABLE

```
CREATE TABLE Personal
```

```
( Persnr      INT          PRIMARY KEY ,  
  Name        CHAR (25)    NOT NULL ,  
  Ort         CHAR (15) ,  
  Vorgesetzt   INTEGER     REFERENCES Personal  
                      ON DELETE SET NULL  Spaltenbedingung  
                      ON UPDATE CASCADE ,  
  Gehalt      NUMERIC(8,2)  CHECK(Gehalt BETWEEN 800 AND 9000),  
  Beurteilung  CHAR ,  
  CONSTRAINT MinVerdienst  
    CHECK( Gehalt >= Coalesce((6-Beurteilung)*400, 800) )  
);
```

Spaltenname **Datentyp**

Komma

Spaltenbedingung

Spaltenbedingung

Spaltenbedingung

Tabellenbedingung

Wichtige Datentypen in SQL

| | |
|-----------------------------|---|
| INTEGER | Ganzzahl |
| INT | |
| SMALLINT | Ganzzahl |
| NUMERIC(x,y) | x stellige Dezimalzahl mit y Nachkommastellen |
| DECIMAL(x,y) | x stellige Dezimalzahl mit y Nachkommastellen |
| FLOAT(x) | Gleitpunktzahl mit x Nachkommastellen |
| CHARACTER(n) | |
| CHAR(n) | Zeichenkette der festen Länge n |
| CHARACTER VARYING(n) | |
| VARCHAR(n) | Variable Zeichenkette mit bis zu n Zeichen |
| BIT(n) | Bitleiste der festen Länge n |
| DATE | Datum (Jahr, Monat, Tag) |
| TIME | Uhrzeit (Stunde, Minute, Sekunde) |
| DATETIME | Kombination aus DATE und TIME |

Datentyp DATE

► Arithmetik:

Datum1 – Datum2 = Anzahl der Tage dazwischen

Datum + Zahl = Datum addiert um Zahl der Tage

Datum1 + Datum2 FEHLER

► Aktuelles Datum (heute):

► CURRENT_DATE

- ▶ In Oracle auch: SYSDATE
- ▶ In SQL Server: GETDATE()

► Beispiel:

► Gestern: CURRENT_DATE – 1

Umwandlung: String in DATE

- ▶ **CAST-Funktion:**
 - ▶ Beispiel: `CAST ('2012-12-24' AS DATE)`
- ▶ **DATE Operator:**
 - ▶ Format weltweit genormt: `YYYY-MM-TT`
 - ▶ Beispiel: `DATE '2012-12-24'`
 - ▶ In SQL Server nicht implementiert, Operator DATE weglassen!
- ▶ **Herstellerspezifisch:**
 - ▶ In Oracle: `to_date('24.12.2012', 'DD.MM.YYYY')`
 - ▶ In SQL-Server: `convert(date, '24.12.2012')`
 - ▶ In MySQL: `str_to_date('24.12.2012', '%d.%m.%y')`

Zeichenketten CHAR und VARCHAR

▶ Char(n)

- ▶ Zeichenkette der Länge n
- ▶ Es wird ein Speicherplatz von n Byte benötigt
- ▶ Beim Einfügen: Auffüllen mit Leerzeichen

▶ Varchar(n)

- ▶ Variable Zeichen
- ▶ Längenfeld enthält die Länge der Zeichenkette
- ▶ Beim Einfügen: Kein Auffüllen mit Leerzeichen
- ▶ Statische oder dynamische Speichertechnik, abhängig von der Länge n und vom Hersteller

Vergleich: CHAR / VARCHAR

- ▶ Bei kleinen Zeichenketten ($n < 100$):
 - ▶ Meist gleicher statischer Speicherbedarf
 - ▶ Werden wenige Zeichen eingefügt, so muss bei Varchar nicht aufgefüllt werden
 - ▶ Bei Char ist kein Längenfeld erforderlich
- ▶ Bei großen Zeichenketten:
 - ▶ In Relation existiert bei Varchar nur ein Link auf den dynamischen Bereich. Dies reduziert den Speicherverbrauch

Verhalten: CHAR / VARCHAR

```
SELECT * FROM Personal WHERE Name LIKE '%e_';
```

- ▶ Ausgabe, falls Name vom Typ VARCHAR(n) ist:
 - ▶ Alle Namen, deren vorletzter Buchstabe ein e ist
- ▶ Ausgabe, falls Name vom Typ CHAR(n) ist:
 - ▶ Vermutlich nichts, da die auffüllenden Leerzeichen mitzählen
(außer in MySQL, eigenartig in SQL Server)
- ▶ Dringender Rat: TRIM oder RTRIM verwenden:

```
SELECT * FROM Personal WHERE Trim(Name) LIKE '%e_';
```

Spaltenbedingungen

- **NOT NULL** Spalte darf keine Nullwerte enthalten
- { **PRIMARY KEY** | **UNIQUE** } Kennzeichnung des Primärschlüssels und der alternativen Schlüssel
- **REFERENCES** Tabellenname [(Spalte [,...])]
[ON DELETE { NO ACTION | CASCADE | SET NULL }]
[ON UPDATE { NO ACTION | CASCADE | SET NULL }] Kennzeichnung der Fremdschlüssel
- **CHECK** (Bedingung) Zusätzliche Einschränkungen

Vor jeder Spaltenbedingung kann wahlfrei ein Bedingungsname angegeben werden:

- [**CONSTRAINT** Bedingungsname]

Hinweise zu Spaltenbedingungen

- ▶ Spaltenbedingungen beziehen sich auf eine Spalte
 - ▶ Auch die Check-Bedingung bezieht sich nur auf „ihre“ Spalte
- ▶ Eine Relation enthält nur eine Angabe **PRIMARY KEY**
- ▶ Vorgaben sind, falls nicht angegeben:
 - ON DELETE NO ACTION
 - ON UPDATE NO ACTION
- ▶ Problem:
 - ▶ Wie werden Schlüssel definiert, die sich auf mehrere Spalten beziehen?
 - ▶ Wie werden Check-Bedingungen definiert, die sich auf mehrere Spalten beziehen?

Tabellenbedingungen

- { PRIMARY KEY | UNIQUE } (Spalte [, ...]) Spaltenangaben
- FOREIGN KEY (Spalte [, ...])
 REFERENCES Tabellename [(Spalte [, ...])]
 [ON DELETE { NO ACTION | CASCADE | SET NULL }]
 [ON UPDATE { NO ACTION | CASCADE | SET NULL }]
- CHECK (Bedingung) Kann sich auf alle Attribute
der Relation beziehen

Vor jeder Spaltenbedingung kann wahlfrei ein Bedingungsname angegeben werden:

- [CONSTRAINT Bedingungsname]

Beispiel zu CREATE TABLE

```
CREATE TABLE Personal
```

```
( Persnr      INT          PRIMARY KEY ,  
  Name        CHAR (25)    NOT NULL ,  
  Ort         CHAR (15) ,  
  Vorgesetzt   INTEGER     REFERENCES Personal  
                        ON DELETE SET NULL  
                        ON UPDATE CASCADE ,  
  Gehalt      NUMERIC(8,2)  CHECK(Gehalt BETWEEN 800 AND 9000),  
  Beurteilung  CHAR ,  
  CONSTRAINT MinVerdienst  
    CHECK( Gehalt >= Coalesce((6-Beurteilung)*400, 800) )  
);
```

Persnr ist
Primärschlüssel

Name darf nicht
Null sein

Vorgesetzt ist
Fremdschlüssel

Spaltenbedingung:
Gehalt erfüllt Bedingung

Tabellenbedingung! Gehalt und Beurteilung involviert!

Create Table: Oracle, SQL Server, MySQL

- ▶ **Oracle:**
 - ▶ On Delete No Action: Angabe nicht erlaubt, standardmäßig gesetzt
 - ▶ On Update: Angabe grundsätzlich nicht erlaubt. Vorgabe: No Action
- ▶ **SQL Server:**
 - ▶ Kein Operator DATE. Umwandlung geschieht automatisch
- ▶ **MySQL:**
 - ▶ Keine Bedingungsnamen bei Spaltenbedingungen
 - ▶ Fremdschlüsselangaben nur als Tabellenbedingungen (InnoDB)
 - ▶ Spaltennamen müssen bei Fremdschlüsseln angegeben werden

Befehl ALTER TABLE

- ▶ Ändern einer bestehenden Basisrelation
- ▶ Eine Änderung kann pro Befehl vorgenommen werden

ALTER TABLE Tabellenname

{ ADD [COLUMN] Spalte { Datentyp | Gebietsname }
[Spaltenbedingung [...]]

Oder: Spalte löschen

| DROP [COLUMN] Spalte { RESTRICT | CASCADE }

| ADD

| DROP CONSTRAINT Bedingungsname

Oder: Tabellenbedingung
löschen

Entweder:
neue Spalte

Oder: neue
Tabellenbedingung

Alter Table: Oracle, SQL Server, MySQL

- ▶ **Oracle:**
 - ▶ ADD: Bezeichner COLUMN ist nicht erlaubt
 - ▶ DROP: Bezeichner COLUMN zwingend, oder Spalte geklammert
 - ▶ Bezeichner RESTRICT nicht erlaubt, gilt standardmäßig
 - ▶ CASCADE CONSTRAINT statt CASCADE
- ▶ **SQL Server:**
 - ▶ Bezeichner COLUMN, RESTRICT, CASCADE nicht erlaubt.
Standardmäßig gilt RESTRICT
- ▶ **MySQL:**
 - ▶ DROP CONSTRAINT: Eigene Syntax (DROP INDEX, DROP KEY)

Befehl DROP TABLE

- ▶ Entfernen einer Relation

DROP TABLE Tabellenname { RESTRICT | CASCADE }

- ▶ Beispiel:

DROP TABLE Personal CASCADE ;

- ▶ Oracle, SQL Server, MySQL:

- ▶ RESTRICT, CASCADE nicht erlaubt bzw. ignoriert
- ▶ Standard: RESTRICT
- ▶ In Oracle CASCADE CONSTRAINT möglich

Temporäre Relationen

```
CREATE { LOCAL | GLOBAL } TEMPORARY TABLE Tabellename  
( { Spalte { Datentyp | Gebietsname } [ Spaltenbedingung [ ... ] ]  
| Tabellenbedingung }  
[ , ... ]  
) [ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

Analog zu
Create Table

Inhalt löschen bei Transaktionsende

- ▶ Anlegen von Spalten und Tabellenbedingungen wie bisher
 - ▶ Sichtbarkeit lokal oder auch für andere Benutzer (global)
 - ▶ Automatisches Löschen dieser Relation am Sessionende
 - ▶ Falls gewünscht: Löschen des Inhalts bei Transaktionsende
- ▶ 19

Temporäre Tabelle: Oracle, SQL Server, MySQL

- ▶ **Oracle:**
 - ▶ Angabe LOCAL nicht möglich
- ▶ **SQL Server:**
 - ▶ CREATE TEMPORARY TABLE nicht implementiert. Stattdessen:
 - ▶ Lokal temporär: #...
 - ▶ Global temporär: ##...
 - ▶ ON COMMIT nicht implementiert, kein Löschen bei TA-Ende
- ▶ **MySQL:**
 - ▶ LOCAL und GLOBAL nicht möglich. LOCAL ist Standard
 - ▶ ON COMMIT nicht implementiert, kein Löschen bei TA-Ende

Sichten (Views)

- ▶ Eine Sicht ist eine virtuelle Tabelle, die über einen Select-Befehl implementiert ist

```
CREATEVIEW Sichtname [ ( Spalte [ ,... ] ) ]  
AS Select-Befehl  
[WITH CHECK OPTION ]
```

```
DROPVIEW Sichtname { RESTRICT | CASCADE }
```

- ▶ Oracle, SQL Server, MySQL:
 - ▶ RESTRICT, CASCADE wie bei DROPTABLE

Sichten und Zugriffsschutz

► Szenario:

- ▶ Benutzer darf nur auf einige Attribute von Personal zugreifen
- ▶ Also: Zugriff auf Sicht Personall erlauben, nicht auf Personal!

CREATE VIEW Personall

Personall ist
eine Relation!

AS SELECT Persnr, Name, Ort AS Wohnort, Vorgesetzt AS Chef
FROM Personal ;

Personall besitzt 4 Attribute:
Persnr, Name, Wohnort, Chef

► Beispielhafter Zugriff:

SELECT * FROM Personall ;

Relation Personall

▶ `SELECT * FROM Personall ;`

| Persnr | Name | Wohnort | Chef |
|--------|------------------|------------|------|
| 1 | Maria Forster | Regensburg | NULL |
| 2 | Anna Kraus | Regensburg | 1 |
| 3 | Ursula Rank | Frankfurt | 6 |
| 4 | Heinz Rolle | Nürnberg | 1 |
| 5 | Johanna Köster | Nürnberg | 1 |
| 6 | Marianne Lambert | Landshut | NULL |
| 7 | Thomas Noster | Regensburg | 6 |
| 8 | Renate Wolters | Augsburg | 1 |
| 9 | Ernst Pach | Stuttgart | 6 |

Sichten und Lesbarkeit

- ▶ **Szenario:**
 - ▶ Auftrag soll „lesbar“ werden, z.B. Kundename statt Kundnr

CREATEVIEW

Auftragssumme
(Gruppierung)

```
VAuftrag (AuftrNr, Datum, Kundname, Persname, Summe) AS
SELECT AuftrNr, Datum, Kunde.Name,
       Personal.Name, SUM(Gesamtpreis)
FROM   Auftrag JOIN Kunde ON Kunde.Nr = Auftrag.Kundnr
               JOIN Personal USING (Persnr)
               JOIN Auftragsposten USING (Auftrnr)
GROUP BY Auftrnr, Datum, Kunde.Name, Personal.Name ;
```

Join über 4
Tabellen

Sicht VAuftrag

| AuftrNr | Datum | Kundname | Persname | Summe |
|---------|------------|-------------------|----------------|-------|
| 1 | 04.01.2013 | Fahrrad Shop | Anna Kraus | 800 |
| 2 | 06.01.2013 | Maier Ingrid | Johanna Köster | 2350 |
| 3 | 07.01.2013 | Rafa – Seger KG | Anna Kraus | 800 |
| 4 | 18.01.2013 | Fahrräder Hammerl | Johanna Köster | 824 |
| 5 | 06.02.2013 | Fahrrad Shop | Anna Kraus | 70 |

► Beispielhafter Zugriff:

```
SELECT * FROM VAuftrag  
WHERE Persname LIKE '%Köster%' ;
```

Zugriff: Intern generierter Befehl

```
SELECT * FROM  
  ( SELECT AuftrNr, Datum, Kunde.Name AS Kundname,  
          Personal.Name AS Persname, SUM(Gesamtpreis) AS Summe  
    FROM Auftrag JOIN Kunde ON Kunde.Nr = Auftrag.Kundnr  
              JOIN Personal USING (Persnr)  
              JOIN Auftragsposten USING (Auftrnr)  
   GROUP BY AuftrNr, Datum, Kunde.Name, Personal.Name  
 )  
 WHERE Persname LIKE '%Köster%' ;
```

VAuftrag:
Select-Befehl übernehmen

Änderbare Sichten

- ▶ Eine Sicht ist änderbar, wenn gilt:
 - ▶ Die From-Klausel enthält nur eine Relation
 - ▶ Eine Group-By-Klausel ist nicht vorhanden
 - ▶ Die Select-Klausel enthält keine Distinct-Angabe
 - ▶ Die Spaltenliste besteht nur aus einzelnen Spaltennamen
 - ▶ Die Operatoren Union, Intersect, Except kommen nicht vor
- ▶ Sicht Personal I ist änderbar
- ▶ Sicht VAuftrag ist nicht änderbar

Beispiel einer änderbaren Sicht

UPDATE Personal

SET Wohnort = 'Hannover'

WHERE Persnr = 2 ;

▶ Ergebnis (Auszug aus Relation Personal):

| Persnr | Name | Strasse | PLZ | Ort |
|--------|---------------|-----------------|-------|------------|
| 1 | Maria Forster | Ebertstr. 28 | 93051 | Regensburg |
| 2 | Anna Kraus | Kramgasse 5 | 93047 | Hannover |
| 3 | Ursula Rank | Dreieichstr. 12 | 60594 | Frankfurt |
| 4 | Heinz Rolle | In der Au 5 | 90455 | Nürnberg |
| ... | ... | ... | ... | ... |

Beispiel: Sicht Jugend

- ▶ **Gegeben: Relation Vereinsmitglieder und Sicht Jugend:**

```
CREATE VIEW Jugend AS
```

```
SELECT * FROM Vereinsmitglieder WHERE Alter < 21 ;
```

- ▶ **Szenario: Mitglied 227 wird 21 Jahre alt:**

```
UPDATE Jugend
```

```
SET Alter = Alter + 1
```

```
WHERE Mitgliednr = 227 ;
```

- ▶ **Problem: Mitglied 227 ist verschwunden !?**

Problem der Sicht Jugend

- ▶ Ein Update wirkt sich aus wie ein Delete!
 - ▶ Ein Mitglied verschwindet mit 21 Jahren aus der Sicht
 - ▶ Aber: Mitglied existiert noch (in Relation Vereinsmitglieder)
- ▶ Lösung: Wir verbieten solche Änderungen!
- ▶ Option: WITH CHECK OPTION

CREATE VIEW Jugend AS

SELECT * FROM Vereinsmitglieder WHERE Alter < 21

WITH CHECK OPTION;

Jetzt muss das
Alter kleiner
21 bleiben

Ansonsten:
Fehlermeldung

Zusicherungen (Assertions)

- ▶ Zusicherung: Datenbankweite Bedingung
- ▶ Zusicherung erzeugen:

CREATE ASSERTION Bedingungsname

CHECK (Bedingung)

- ▶ Zusicherung entfernen:

DROP ASSERTION Bedingungsname

Beispiel zu Zusicherung

- ▶ Angebotspreis soll nicht über Listenpreis liegen
- ▶ Gelöst mit CHECK-Bedingung:

```
ALTER TABLE Auftragsposten ADD  
CONSTRAINT Auftragspreis  
CHECK ( Gesamtpreis <= (SELECT Anzahl*Preis  
                           FROM Artikel  
                          WHERE ANr = Artnr) ) ;
```

In fast allen Datenbanken:
Select-Befehl ist hier
nicht erlaubt!

- ▶ Aber:
 - ▶ Preisänderung in Relation Artikel bleibt unberücksichtigt!

Lösung mit ASSERTION

```
CREATE ASSERTION AssertionPreis
```

```
  CHECK ( NOT EXISTS
```

```
    ( SELECT *
```

```
      FROM Auftragsposten INNER JOIN Artikel ON ANr = Artnr
```

```
      WHERE Gesamtpreis > Anzahl * Preis
```

```
    )    );
```

- ▶ Jetzt Überprüfung in Auftragsposten und Artikel!
- ▶ Aber:
 - ▶ In Oracle, SQL Server und MySQL nicht implementiert

Lösung mit WITH CHECK OPTION

- ▶ Besser als nichts: **WITH CHECK OPTION** verwenden

```
CREATE VIEW VAuftragsposten AS
```

```
    SELECT * FROM Auftragsposten
```

```
    WHERE Gesamtpreis <= ( SELECT Anzahl * Preis
```

```
        FROM Artikel
```

```
        WHERE Artnr = Anr )
```

```
    WITH CHECK OPTION ;
```

Lösung funktioniert,
wenn Benutzer nur über
diese Sicht zugreift

- ▶ Noch besser: **TRIGGER** (siehe weiter unten)

Gebiete (Domains)

- ▶ Verfeinerung von Datentypen
 - ▶ Beispiel Aufzählungen: Städte, Familienstand
- ▶ Gebiet erzeugen:

CREATE DOMAIN Gebietsname [AS] Datentyp

[[CONSTRAINT Bedingungsname] CHECK (Bedingung)] [...]

- ▶ Gebiet entfernen:

DROP DOMAIN Gebietsname { RESTRICT | CASCADE }

Beispiel zu Gebieten

▶ Definition einiger wichtiger EURO-Hauptstädte:

```
CREATE DOMAIN EURO_Hauptstadt AS CHARACTER (15)
CHECK (VALUE IN ( 'Berlin', 'Paris', 'Rom', 'Madrid', 'Lissabon',
                  'Amsterdam', 'Dublin', 'Brüssel', 'Athen',
                  'Luxemburg', 'Wien', 'Helsinki' ) );
```

▶ Aber:

- ▶ In Oracle, SQL Server und MySQL nicht implementiert

▶ Alternative:

- ▶ CHECK-Bedingungen oder Trigger; aber nicht so elegant!

Trigger

- ▶ **Trigger sind ereignisgesteuerte Aktionen**
- ▶ **Einsatz von Triggern:**
 - ▶ Überprüfen, ob Eingaben im erlaubten Bereich
 - ▶ Zusätzliches Ablegen von Informationen (z.B. Protokollierung)
 - ▶ Ausgaben von Infos und Warnungen
- ▶ **Mögliche auslösende Ereignisse:**
 - ▶ Vor oder nach dem Einfügen, Löschen, Ändern von Relationen
- ▶ **Aktionen:**
 - ▶ **SQL-Befehle, (kleinere) Prozeduren**

Trigger (Erzeugen und Entfernen)

CREATE TRIGGER Triggername

vor oder nach ...

... Änderungen ...

{ BEFORE | AFTER } { DELETE | INSERT | UPDATE } [OR ...]

ON Tabellenname

... in Relation

[REFERENCING [OLD AS NameAlt] [NEW AS NameNeu]]

[FOR EACH STATEMENT | FOR EACH ROW]

[WHEN Bedingung]

Anweisungen

Ganze Relation
bearbeiten (Vorgabe)
oder jedes Tupel einzeln

Setzen von
Aliasnamen für
neue/alte Daten

DROP TRIGGER Triggername

Anweisungen
ausführen, wenn
Bedingung wahr

Beispiel 1 zu Trigger

- ▶ Wunsch: Bei jedem neuen Auftrag wird automatisch das heutige Datum (`CURRENT_DATE`) gesetzt
- ▶ Lösungsversuch:

```
CREATE TRIGGER AuftragsdatumI_Trigger
```

```
  AFTER INSERT ON Auftrag
```

```
  BEGIN
```

```
    UPDATE AUFTAG
```

```
      SET Datum = CURRENT_DATE ;
```

```
  END ;
```

Nach dem Einfügen in Auftrag ...

In Standard-SQL:
`BEGIN ATOMIC`

... erfolgt dieser Update

- ▶ Aber:

- ▶ Es werden alle Aufträge auf das heutige Datum gesetzt!

Beispiel 2: Korrektur von Beispiel 1

- ▶ Jetzt: Tupelweises Ausführen des Triggers
- ▶ Lösung für Oracle

CREATE TRIGGER Auftragsdatum2_Trigger

BEFORE INSERT ON Auftrag

Vor dem Einfügen in Auftrag ...

REFERENCING NEW AS neu FOR EACH ROW

BEGIN

In Standard-SQL:
BEGIN ATOMIC

... wird zeilenweise überprüft ...

:neu.Datum := CURRENT_DATE ;

END;

In Standard-SQL:
keine Doppelpunkte

... und zu änderndes Datum wird vor
dem Einfügen durch neuen (:neu)
Eintrag überschrieben

Beispiel 3

- ▶ Auftragspreis ist nicht größer als Listenpreis
- ▶ Lösung mit Trigger:
 - ▶ Wenn doch, dann Listenpreis setzen und Meldung ausgeben
 - ▶ Gelöst mit Oracle PL/SQL

Vor Einfügen oder Ändern in
Auftrag zeilenweise ausführen

```
CREATE TRIGGER Auftragspreis_Trigger
  BEFORE INSERT OR UPDATE ON Auftragsposten
  REFERENCING NEW AS neu FOR EACH ROW
```

DECLARE

listenpreis NUMERIC(8,2) ;

Deklarationsteil in
PL/SQL

Beispiel 3: Anweisungen

BEGIN

```
SELECT :neu.Anzahl*Preis          -- Berechnung des Listenpreises
```

```
INTO listenpreis
```

Speichert Ergebnis in
Variable listenpreis

```
FROM Artikel
```

```
WHERE Anr = :neu.Artnr ;
```

Abfrage, ob neuer
Gesamtpreis zu hoch

```
IF (:neu.Gesamtpreis > listenpreis)
```

THEN

```
:neu.Gesamtpreis := listenpreis ;
```

Wenn ja, dann
ersetzen und ausgeben

```
dbms_output.put_line ('Preis in Posnr ' || :neu.posnr || ' geändert');
```

END IF;

END ;

Ausgabefunktion
in Oracle

Konkatenierung in
Oracle

/ in Oracle wichtig!

Sequenzen

- ▶ Sequenzen sind Generatoren zum Erzeugen von automatischen Nummerierungen
- ▶ Sequenz erzeugen:

CREATE SEQUENCE Sequenzname [AS Datentyp]
[START WITH Konstante] [INCREMENT BY Konstante]

- ▶ Sequenz entfernen:

DROP SEQUENCE Sequenzname

Beispiel

- ▶ Aufträge mit automatischer Auftragsnummer:
 - ▶ Nummern fortlaufend ab 1000

```
CREATE SEQUENCE Auftragssequenz  
START WITH 1000;
```

- ▶ Verwendung:

```
INSERT INTO Auftrag(Auftrnr, Datum, Kundnr, Persnr) VALUES  
(NEXT VALUE FOR Auftragssequenz, CURRENT_DATE, 3, 5);
```

- ▶ In Oracle:

```
... VALUES (Auftragssequenz.NEXTVAL, CURRENT_DATE, 3, 5);
```

Zugriffsrechte

- ▶ **Zugriffsrechte auf Relationen**
 - ▶ Beliebiger Zugriff für den Eigentümer
 - ▶ Eigentümer: Benutzer, der die Relation erzeugte
 - ▶ Zugriff für andere nur, wenn Rechte eingeräumt werden
- ▶ **Gewähren und Entziehen von Zugriffsrechten**
 - ▶ Eigentümer kann Zugriffsrechte an Dritte vergeben
 - ▶ Diese Rechte können jederzeit widerrufen werden
 - ▶ Eigentümer behält immer alle Rechte

Zugriffsrechte gewähren (Grant)

GRANT Zugriffsrecht [, ...] ev. mehrere Rechte
ON [TABLE] { Tabellenname | Sichtname }
TO Benutzer [, ...] nur eine Relation!
[WITH GRANT OPTION] ev. mehrere Benutzer

▶ **Grant-Befehl** gibt an,

- ▶ welche(r) Benutzer
- ▶ auf welche Relation
- ▶ welche(s) Zugriffsrecht(e)

Vererben eines Zugriffsrechts,
siehe weiter unten

erhält / erhalten

Zugriffsrechte

| Zugriffsrecht | erlaubt ... |
|-----------------------------|--|
| Select | den lesenden Zugriff auf eine Relation |
| Update | das Ändern von Inhalten einer Relation |
| Update (x_1, \dots) | das Ändern der Attributwerte x_1, \dots einer Relation |
| Delete | das Löschen von Tupeln einer Relation |
| Insert | das Einfügen neuer Tupel in eine Relation |
| Insert (x_1, \dots) | das Einfügen der Attributwerte x_1, \dots in eine Relation |
| References (x_1, \dots) | das Referenzieren der Attribute x_1, \dots einer Relation |
| Usage | das Verwenden eines Gebietes |

Besonderheiten im Grant Befehl

- ▶ Alle Rechte vergeben:
 - ▶ ALL [PRIVILEGES]
- ▶ Rechte an alle Benutzer vergeben:
 - ▶ Benutzername PUBLIC verwenden
- ▶ OPTION: WITH GRANT OPTION
 - ▶ Benutzer erhält mit dem Zugriffsrecht auch das Recht der Weitergabe dieses Rechts an Dritte
- ▶ Oracle, SQL Server, MySQL:
 - ▶ Komplett implementiert, mit Erweiterungen (z.B. mehrere Relationen, Wildcart-Syntax); teilweise: Füllwort TABLE verboten

Zugriffsrechte entziehen (Revoke)

REVOKE [GRANT OPTION FOR]

{ Zugriffsrecht [,...] | ALL PRIVILEGES }

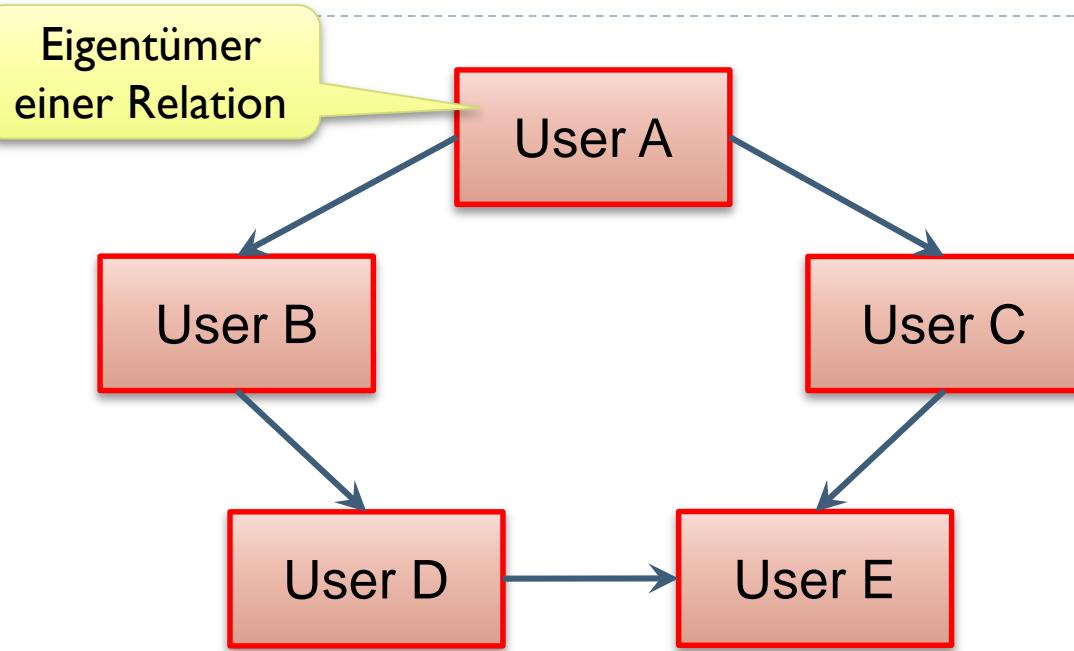
ON [TABLE] { Tabellenname | Sichtname }

FROM Benutzer [,...]

{ RESTRICT | CASCADE }

- ▶ Oracle, SQL Server, MySQL:
 - ▶ GRANT OPTION FOR, RESTRICT, CASCADE, TABLE teilweise nicht unterstützt
 - ▶ In MySQL gibt es kein Reference-Recht

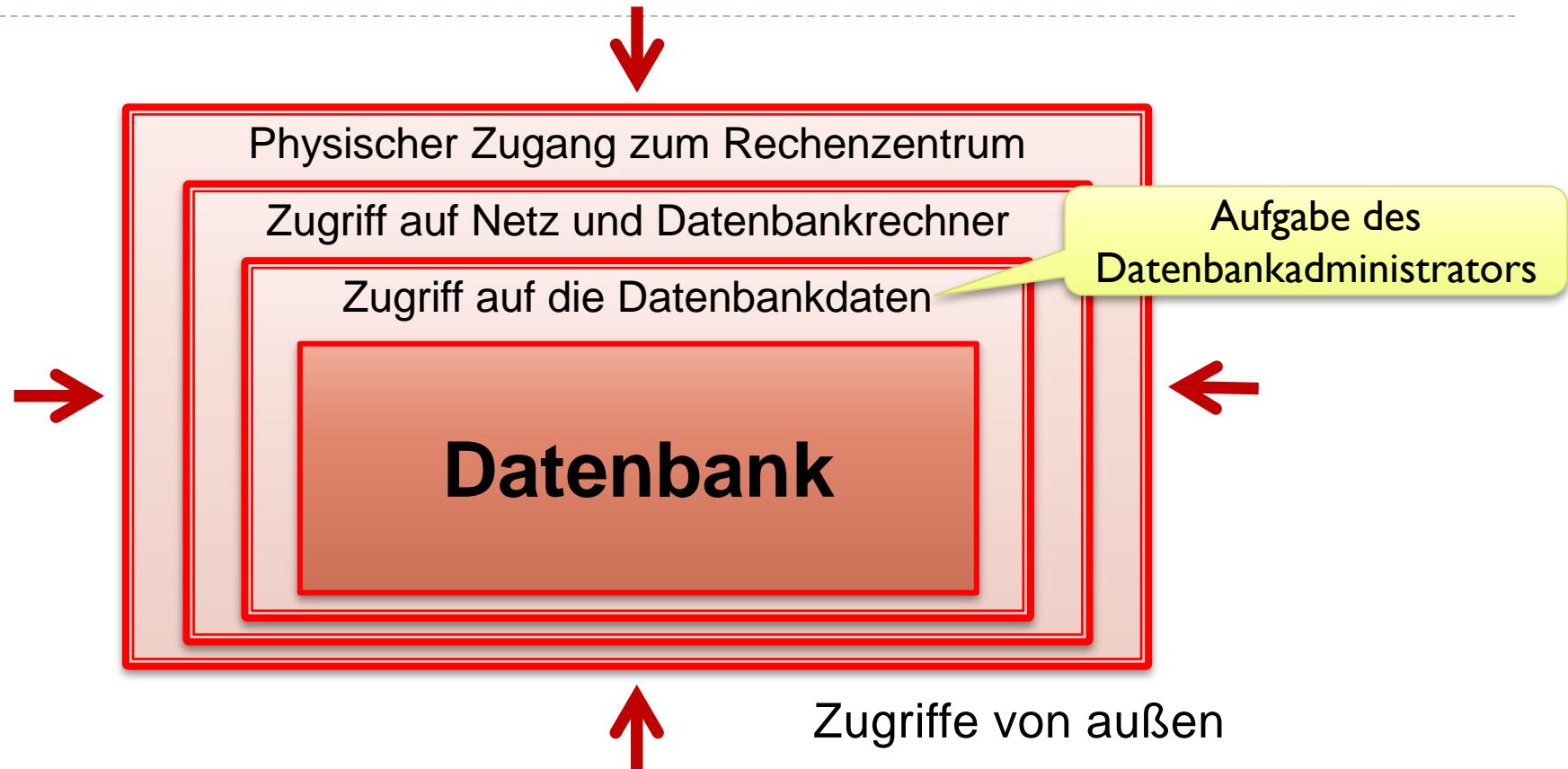
Kaskadierendes Entziehen von Rechten



Beim kaskadierenden Revoke wird die ganze betroffene Rechtekette gelöscht

- ▶ Rechtevergabe mit **Grant Option**:
 - ▶ A → B
 - ▶ A → C
 - ▶ B → D
 - ▶ D → E
 - ▶ C → E
- ▶ Rechteentzug kaskadierend:
 - ▶ A entzieht B
 - ▶ Automatisch auch:
 - ▶ B entzieht D
 - ▶ D entzieht E

Zugriffsschutz



Zugriffsschutz auf Relationen (1)

- ▶ In SQL: Select-Recht gibt es nur auf gesamte Relation
- ▶ Lösung: Sichten!
- ▶ Beispiel: Relation Personal; Erzeugen einer Sicht VPersonal
 - ▶ Projektion: Zugriff auf Persnr, Name, Ort, Vorgesetzt, Aufgabe
 - ▶ Restriktion: Kein Zugriff auf Daten von Vorgesetzten

```
CREATE VIEW VPersonal AS
```

```
    SELECT Persnr, Name, Ort, Vorgesetzt, Aufgabe
```

```
        FROM Personal
```

```
        WHERE Vorgesetzt IS NOT NULL ;
```

Projektion

Restriktion

Zugriffsschutz auf Relationen (2)

► Einstellen des Zugriffs für Benutzer Mitarbeiter:

```
REVOKE ALL PRIVILEGES  
ON Personal  
FROM Mitarbeiter ;
```

Entziehen der
Rechte auf Personal

```
GRANT SELECT  
ON VPersonal  
TO Mitarbeiter ;
```

Gewähren der
Rechte auf VPersonal

Zugriffsschutz auf Relationen (3)

▶ Zugriffe, abhängig von Benutzergruppen

| Benutzer | Rechte |
|-------------------|---|
| Alle Benutzer | Select-Recht auf VPersonal |
| Abteilungsleiter | Select- und Update-Recht auf VPersonal Select-Recht auf Personal |
| Personalabteilung | Select-Recht auf Personal Änderungsrechte auf einzelne Attribute |
| Personalchef | Alle Rechte auf Personal |

Semantische Integrität (Maßnahmen)

- Benutzer greift über Masken und Eingabefelder zu GUI
- Eingabefelder sind eingabespezifisch:
 - Beispiel: Bei Zahleneingaben nur Ziffern zulassen.
- Bei Eingabefeldern möglichst Auswahlfelder verwenden
- Nummern automatisch generieren GUI+Programmierung
 - Beispiel: Auftragsnummer, Personalnummer, Kundennummer
- Überprüfen auf Plausibilität und Korrektheit Sequenzen
 - Check-Bedingung, Assertion, Domain,
With Check Option, Trigger

Beispiel: Auswahlfeld

► Auswahlfeld, programmiert mit PHP:

Bitte wählen Sie einen Kunden aus:

Bitte wählen Sie einen Artikel aus:

- Herren-City-Rad
- Herren-City-Rad
- Damen-City-Rad
- Herren-City-Rahmen lackiert
- Damen-City-Rahmen lackiert
- Herren-City-Rahmen geschweisst
- Damen-City-Rahmen geschweisst
- Rad**
- Rohr 25CrMo4 9mm
- Sattel
- Gruppe Deore LX
- Gruppe Deore XT
- Gruppe XC-LTD
- Felgensatz
- Bereifung Schwalbe

Vorteil:

Nur erlaubte Eingaben möglich

Nachteil:

Zusätzlicher Zugriff auf Datenbank

Vorteil überwiegt Nachteil!

Unterstützende Werkzeuge, Beispiele

| Befehl | Info |
|------------------|---|
| Create Table | Check-Bedingung |
| Create View | With-Check-Option |
| Create Assertion | Datenbankweite Zusicherung |
| Create Domain | Datenbankeinheitliche Gebiete |
| Create Trigger | Eingabetrigger (Insert, Update, Delete) |

```
ALTER TABLE Personal ADD Arbeitszeit INTEGER NOT NULL  
    CHECK( Arbeitszeit BETWEEN 15 AND 40 ) ;
```

```
ALTER TABLE Artikel ADD CONSTRAINT Preischeck  
    CHECK( Preis = Netto + Steuer ) ;
```

Gefährlich: Rundungsfehler!

Schema

- ▶ Datenbank besteht aus mehreren Schemata
- ▶ Schema enthält Relationen, Zugriffsrechte, ev. Trigger
- ▶ Schema erzeugen:

CREATE SCHEMA Schemaname

[AUTHORIZATION Benutzername] [Schemaelement [...]]

- ▶ Schema entfernen:

DROP SCHEMA Schemaname { CASCADE | RESTRICT }

Beispiel zu Schema: Schema Bike

- ▶ Datenbank Bike als eigenes Schema definieren

CREATE SCHEMA **Bike**

 CREATE TABLE Personal (...)

 CREATE TABLE Kunde (...)

 CREATE TABLE Auftrag (...)

 CREATE VIEW VAuftrag (...)

 GRANT ...

... ;

- ▶ Externer Zugriff mittels: **Bike.Personal, Bike.Kunde usw.**

Information Schema

| Relation | enthält |
|--------------------------------|--|
| SCHEMATA | alle Schemata |
| DOMAINS | alle Gebiete |
| TABLES | alle Basisrelationen |
| VIEWS | alle Sichten |
| VIEW_TABLE_USAGE | alle Abhängigkeiten der Sichten von Relationen |
| VIEW_COLUMN_USAGE | alle Abhängigkeiten der Sichten von Spalten |
| COLUMNS | alle Spaltennamen aller Basisrelationen |
| TABLE_PRIVILEGES | alle Zugriffsrechte auf Relationen |
| COLUMN_PRIVILEGES | alle Zugriffsrechte auf Spalten aller Relationen |
| DOMAIN_CONSTRAINTS | alle Gebietsbedingungen für alle Gebiete |
| TABLE_CONSTRAINTS | alle Tabellenbedingungen aller Relationen |
| REFERENTIAL_CONSTRAINTS | alle referentiellen Bedingungen |
| CHECK_CONSTRAINTS | alle Check-Bedingungen aller Relationen |
| TRIGGERS | alle Trigger |
| TRIGGER_TABLE_USAGE | alle Abhängigkeiten der Trigger von Relationen |
| TRIGGER_COLUMN_USAGE | alle Abhängigkeiten der Trigger von Spalten |
| ASSERTIONS | alle Zusicherungen |
| DOMAINS | alle Gebiete |

Beispielhafter Zugriff:

Select *

From Information_Schema.Tables;

Datenbanken und Oracle

▶ SCHEMA = USER

- ▶ Zu jedem Benutzer wird ein Schema gleichen Namens angelegt
- ▶ Dies geschieht automatisch mit CREATE USER

▶ Rechte auf ein Schema können vergeben werden: CREATE SCHEMA AUTHORIZATION Benutzername ...

▶ Es gibt kein INFORMATION_SCHEMA

Systemtabellen in Oracle (Auszug)

| Relation | Enthält |
|--------------------------|---|
| DICTIONARY | Zusammenfassung zu allen Systemtabellen |
| USER_TABLES | alle Relationen des Benutzers |
| USER_TAB_COLUMNS | alle Attribute aller Relationen des Benutzers |
| USER_VIEWS | alle Sichten des Benutzers |
| USER_CONSTRAINTS | alle Spalten- und Tabellenbedingungen |
| USER_CONS_COLUMNS | alle Attribute mit Spalten- und Tabellenbedingungen |
| USER_INDEXES | alle Indexe in Relationen des Benutzers |
| USER_IND_COLUMNS | alle Attribute, die Indexe besitzen |
| USER_TAB_PRIVS | alle Privilegien in Bezug auf Relationen |
| USER_COL_PRIVS | alle Privilegien in Bezug auf Attribute |
| USER_TRIGGERS | alle Trigger des Benutzers |
| USER_TRIGGER_COLS | alle Attribute, auf die sich Trigger beziehen |
| USER_TABLESPACES | alle Tablespaces des Benutzers |

Datenbanken und SQL Server

▶ CREATE USER:

- ▶ Legt neuen Benutzer und sein Schema fest
- ▶ Standardmäßig Schema dbo,
- ▶ Oder explizit festlegen (**WITH DEFAULT_SCHEMA =**)

▶ ALTER USER:

- ▶ Ändern der Zuordnung zu Schema

▶ CREATE SCHEMA:

- ▶ Zuordnung zu Benutzer möglich

▶ INFORMATION_SCHEMA wird voll unterstützt

Datenbanken und MySQL

- ▶ **SCHEMA = DATENBANK**
 - ▶ **CREATE SCHEMA = CREATE DATABASE**
- ▶ **Schema ist keinem Benutzer zugeordnet**
 - ▶ Kein Parameter: **AUTHORIZATION Benutzername**
- ▶ **USE Schemaname:**
 - ▶ Zuordnung eines Benutzers während der Laufzeit zu Schema
- ▶ **INFORMATION_SCHEMA wird voll unterstützt**

Einloggen in Datenbank

- ▶ Einloggen, Start der Session und der ersten Transaktion:

CONNECT TO {DEFAULT|Servername} [AS Verbindungsname]
[USER Benutzername]

- ▶ Beenden der Session:

DISCONNECT { DEFAULT | CURRENT | SQL-Servername }

- ▶ In Standard-SQL:

- ▶ Kein CREATE DATABASE, kein CREATE USER

Datenbank verwalten in Oracle

- ▶ Je Server mehrere Datenbanken möglich
 - ▶ CREATE DATABASE
 - ▶ ALTER DATABASE
 - ▶ DROP DATABASE
- ▶ Benutzer einrichten:
 - ▶ CREATE USER Benutzername IDENTIFIED BY Kennwort
 - ▶ Gewähren von Verbindungsrechten mit GRANT
- ▶ Beispiel:

CREATE USER gast IDENTIFIED BY neu ;

GRANT Connect, Resource TO gast ;

Connect: Erlaubt Verbinden mit DB
Resource: Erlaubt CREATE-Befehle

Verbinden mit Oracle, Befehle

▶ **CONNECT gast/neu@xe ;**

Verbindung mit Kennung **gast** und
Passwort **neu** an Datenbank **xe**

| | |
|---|---|
| CREATE DATABASE Datenbankname ... | Legt eine neue Datenbank inklusive Logdateien an |
| ALTER DATABASE Datenbankname ... | Ändert Datenbankeinstellungen |
| CREATE CLUSTER Clustername ... | Erzeugt einen neuen Cluster |
| CREATE USER Benutzername | Legt einen neuen Benutzer an |
| ALTER USER Benutzername | Ändert Benutzereinstellungen |
| CONNECT Benutzer/Passwort@DB | Einloggen in Datenbank |
| CREATE TABLESPACE Tablespacename ... | Erzeugt einen physischen Bereich zum Speichern der Basisrelationen und Indexe |

Datenbank verwalten mit SQL Server

- ▶ Pro Server kann eine Datenbank erstellt werden
 - ▶ CREATE DATABASE
 - ▶ ALTER DATABASE
 - ▶ DROP DATABASE
- ▶ Benutzer einrichten:
 - ▶ CREATE USER Benutzername WITH DEFAULT_SCHEMA =
- ▶ Es gibt keinen Connect-Befehl
 - ▶ Einloggen mittels SSMS oder Programmierschnittstelle
- ▶ Beispiel:
CREATE USER guest WITH DEFAULT_SCHEMA = Bike ;

Datenbank verwalten mit MySQL

- ▶ Vorzugsweise: Arbeiten mit MySQL Workbench
- ▶ Mit der Konsole:
 - ▶ mysql -u root
 - ▶ use bike
- ▶ CREATE USER (Unterscheidung zwischen global / lokal):

CREATE USER guest IDENTIFIED BY 'neu' ;

CREATE USER guest@localhost IDENTIFIED BY 'neu' ;

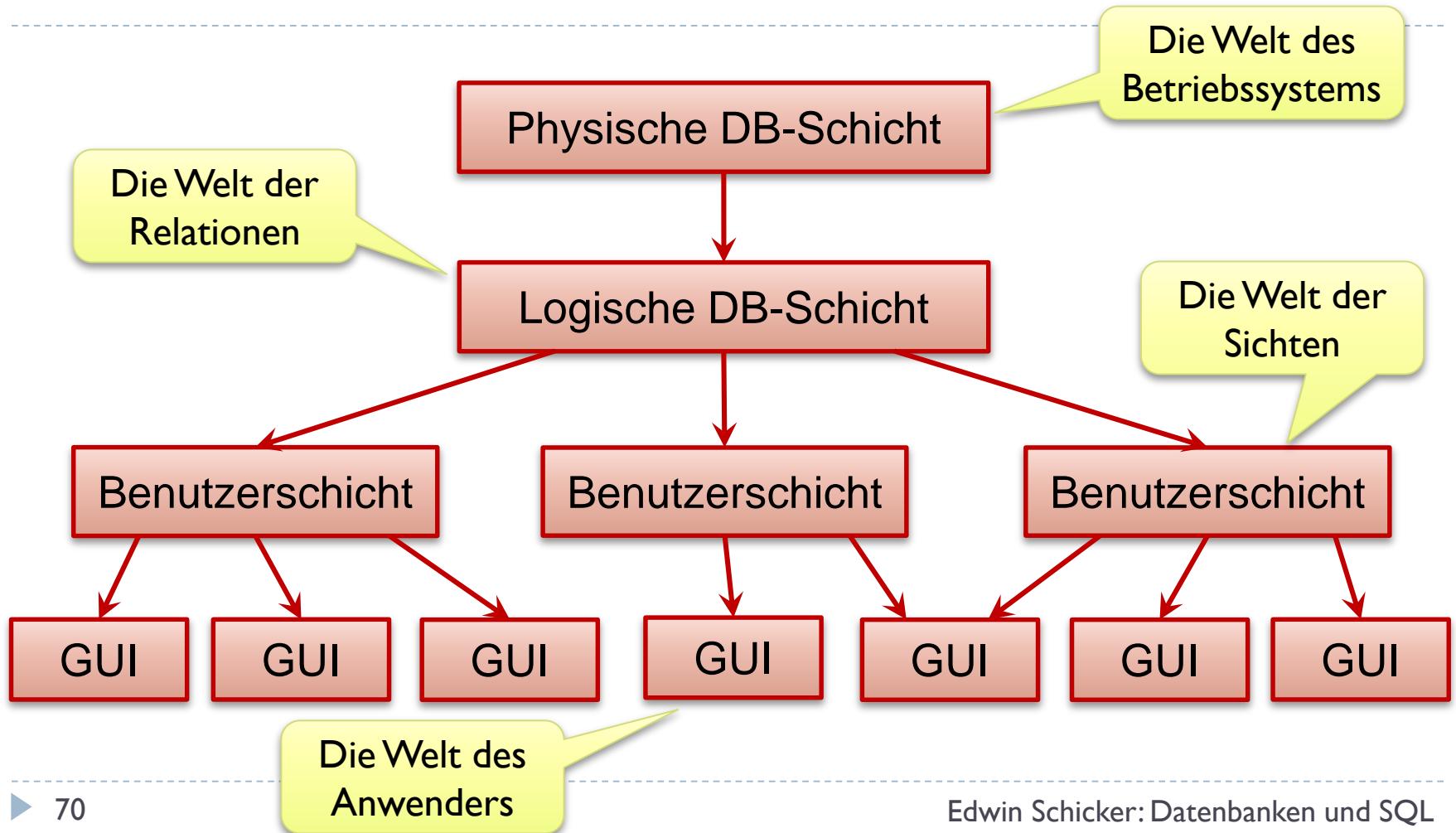
GRANT SELECT, INSERT, DELETE, UPDATE ON Bike.* TO guest ;

GRANT SELECT, INSERT, DELETE, UPDATE ON Bike.*

TO guest@localhost ;

In MySQL
Wildcards erlaubt

Aufbau einer Datenbank



Zusammenfassung

- ▶ DDL hat viele Möglichkeiten zur Gestaltung einer DB:
 - ▶ CREATE TABLE, ALTER TABLE, DROP TABLE
 - ▶ CREATE VIEW, DROP VIEW
 - ▶ CREATE ASSERTION, DROP ASSERTION
 - ▶ CREATE DOMAIN, ALTER DOMAIN
 - ▶ CREATE TRIGGER, ALTER TRIGGER, DROP TRIGGER
 - ▶ CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE
 - ▶ CREATE SCHEMA, ALTER SCHEMA, DROP SCHEMA
 - ▶ GRANT, REVOKE
 - ▶ CREATE DATABASE, CREATE USER, CREATE TABLE_SPACE, ...