Programmierung von Systemen – 14 – SQL 3

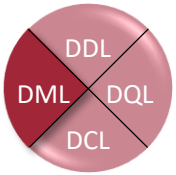
Matthias Tichy & Stefan Götz | SoSe 2020

Ziele

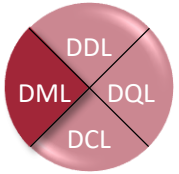
- Daten in einer Datenbank einfügen, verändern können und entfernen können
- Datenbank mit verschiedenen Integritätsbedingungen anlegen können
- Sichten und Schemas verstehen

Wie bekomme ich Daten 'rein?

- Einfügen von Tupeln
- Ändern von Tupeln
- Löschen von Tupeln



Löschen

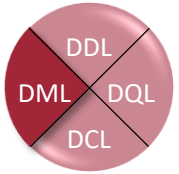



```
DELETE
FROM Relationsname
[ WHERE Bedingung ]
```

Anmerkungen

- Löscht alle Tupel, welche die WHERE-Bedingung erfüllen.
- Fehlt die WHERE-Bedingung, werden alle Tupel gelöscht.
- Die Relation selbst bleibt ggf. als leere Menge bestehen.

Beispiele



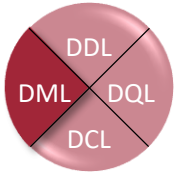
D01: „Lösche alle Tupel in der Relation 'Sonderpreise'“

```
DELETE
FROM   Sonderpreise
```

D02: „Lösche alle Tupel in der Relation 'Sonderpreise', die keine Preisangabe enthalten“

```
DELETE
FROM   Sonderpreise
WHERE Preis IS NULL
```

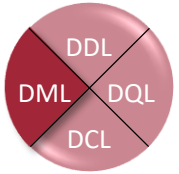
Beispiele



D03: „Lösche alle Lieferanten mit Bewertung = -2 aus der Liefert-Relation“

```
DELETE
FROM    Liefert
WHERE   LiefNr IN
        (  SELECT  LiefNr
           FROM    Lieferanten
           WHERE   Bewertung = -2 )
```

Einfügen



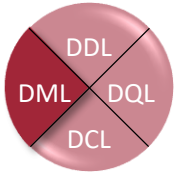
INSERT INTO *Relationsname* [**(** *Attributliste* **)**]

{ **VALUES (** $wert_{11}$, $wert_{12}$, …, $wert_{1n}$ **)** [**(** $wert_{21}$, $wert_{22}$, …, $wert_{2n}$ **) …**] | *SFW-Ausdruck* }

Anmerkungen

- Attributliste bezeichnet Teilmenge der einzufügenden Attribute.
- Rest wird mit Nullwerten aufgefüllt ➔ Restattribute müssen Nullwerte enthalten dürfen.
- Einzufügende Werte als Konstante (VALUES) oder über eine Query
- Einzufügende Werte müssen bzgl. Anzahl und Typ den implizit (keine Attributliste angegeben) oder explizit spezifizierten Attributen passen.

Beispiele



I01: *„Der Klotz 'K18' kann in den Farben 1 und 2 ab sofort auch von Lieferant 528 bezogen werden, die Preise stehen im Moment noch nicht fest“*

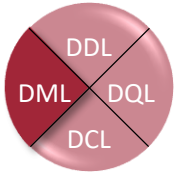
Variante 1: (expliziter Nullwert)

```
INSERT INTO Liefert
  VALUES (528, 'K18', 1, NULL), (528, 'K18', 2, NULL)
```

Variante 2: (impliziter Nullwert)

```
INSERT INTO Liefert(LiefNr, TeileNr, Farbe)
  VALUES (528, 'K18', 1), (528, 'K18', 2)
```

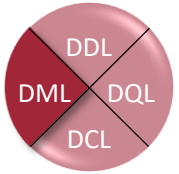
Beispiele



I02: „Angenommen, wir fertigen alle unsere farbigen Artikel ab sofort auch in grüner Farbe (Farbcode = 3). Alle Lieferanten, welche Teile in blauer Farbe liefern, können auch in grüner Farbe liefern. Die Preise für die grünen Teile liegen um 10% über denen der blauen Teile. – Die Liefert-Relation ist entsprechend zu ergänzen.“

```
INSERT INTO Liefert
    SELECT  LiefNr, TeileNr, 3, Preis * 1.1
    FROM    Liefert
    WHERE   Farbe = 2
```

Ändern



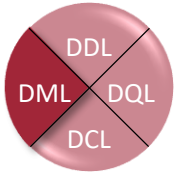
UPDATE *Relationsname* [**AS** Korrelationsvariable]
SET $Attr_1 = Ausdruck_1$, $Attr_2 = Ausdruck_2$, …
[**WHERE** *Bedingung*]

UPDATE *Relationsname* [**AS** Korrelationsvariable]
SET = *(SFW-Ausdruck)*
[**WHERE** *Bedingung*]

Anmerkung

- Die Wertänderung wird für alle Tupel durchgeführt, die die WHERE-Bedingung erfüllen.

Beispiele



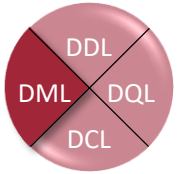
U01: „Erhöhe alle Preise in der Preisliste um 5%.“

```
UPDATE  Preisliste
SET     Preis = Preis * 1.05
```

U02: „Erhöhe die kalkulatorischen Kosten in der Teile-Tabelle bei allen intern bezogenen Teilen um 10%.“

```
UPDATE  Teile
SET     KalkKosten = KalkKosten * 1.1
WHERE   (TeileNr, Farbe) IN
        ( SELECT   TeileNr, Farbe
          FROM     Liefert
          WHERE    LiefNr = 0 )
```

Beispiele



U03: „Ergänze in der Relation 'Sonderpreise' die noch fehlenden Preise. Gewähre für diese Teile einen Rabatt von 15% auf die Normalpreise.“

```
UPDATE Sonderpreise AS s
SET    s.Preis = (
          SELECT Preis * 0.85
          FROM   Preisliste AS p
          WHERE  (s.TeileNr, s.Farbe) = (p.TeileNr, p.Farbe)
       )
WHERE  s.Preis IS NULL
```

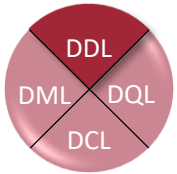
CRUD

- CRUD steht für:
  - Create
  - Read
  - Update
  - Delete

- bekanntes Akronym für grundlegende Datenmanipulationsoperationen

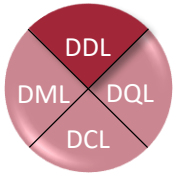
Woher kommt die Datenbank?

- Erzeugen einer Relation
- Ändern einer Relation
- Löschen einer Relation



- Definition von Sichten (views)
- Schemas

Löschen einer Relation



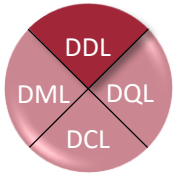

```
DROP TABLE Relationsname
```

Anmerkungen

- Löscht die angegebene Relation samt Inhalt
- Entfernt die Relation mit allen Verweisen und zugehörigen Hilfsdaten (z.B. Indexen) aus dem DB-Katalog

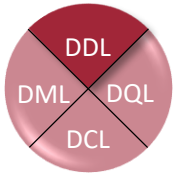
- VORSICHT: Keine Nachfrage, kein Undo!

Erzeugen von Relationen



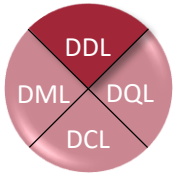

```
CREATE TABLE Relationsname (
        Attribut1   Datentyp1,
        Attribut2   Datentyp2,
        …
        Attributn   Datentypn
)
```

Datentypen



Datentyp	Spezifikation	Erläuterungen
Integer	**INTEGER**	32bit
Real	**FLOAT**, **DOUBLE**, **REAL**	64bit-Precision
Dezimal	**DECIMAL(**g[,k]**)**	g Stellen, davon k Nachkommastellen
Zeichenkette	**CHAR(***Länge***)**	feste Länge
Zeichenkette	**VARCHAR(***Länge***)**	variable Länge (Anfangslänge)
Zeichenkette	**CLOB**	variable Länge, extra groß
Bytestring	**BLOB**	für Binärdaten (z.B. Bilder)
Datum	**DATE**	ISO 8601-Format
Zeit	**TIME**	ISO 8601-Format
Boolean	**BOOLEAN**	**TRUE**, **FALSE**, **NULL**

Beispiel

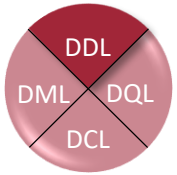



```
CREATE TABLE Preisliste (
    TeileNr     VARCHAR(8),
    Farbe       INTEGER,
    Preis       DECIMAL(8,2)
)
```

- neu angelegte Relation hat drei Attribute: TeileNr, Farbe, Preis
- Für alle drei Attribute sind Nullwerte zugelassen (Default)
- explizit:

```
CREATE TABLE Preisliste (
    TeileNr  VARCHAR(8)     NULL,
    Farbe    INTEGER        NULL,
    Preis    DECIMAL(8,2)   NULL )
```

Primärschlüssel

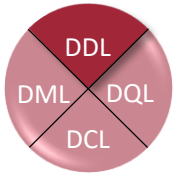



```
CREATE TABLE Relationsname (
        Attribut1       Datentyp1       NOT NULL PRIMARY KEY,
        …
)
```

Alternativ z.B. bei zusammengesetzten Primärschlüsseln:

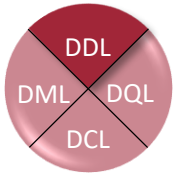
```
CREATE TABLE Relationsname (
        Attribut1       Datentyp1       NOT NULL,
        Attribut2       Datentyp2       NOT NULL,
        … …             ,
        PRIMARY KEY (Attribut1, Attribut2)
)
```

Beispiele



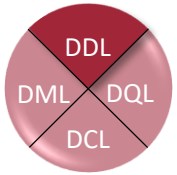

```
CREATE  TABLE Lieferanten (
        LiefNr          INTEGER NOT NULL PRIMARY KEY,
        LiefName        VARCHAR(30) NOT NULL,
        LiefStadt       VARCHAR(30) NOT NULL,
        Bewertung       INTEGER
  )

CREATE TABLE Teile (
        TeileNr         VARCHAR(8) NOT NULL,
        Farbe           INTEGER NOT NULL,
        KalkKosten      DECIMAL(6,2),
        Bestand         INTEGER,
        MinBestand      INTEGER,
        PRIMARY KEY (TeileNr, Farbe)
  )
```

Integritätsbedingungen



- Als Primärschlüssel sind nur solche Attribute oder Attributkombinationen erlaubt, für die **`NOT NULL`** (oder ein anderer Defaultwert) spezifiziert wurde.
- Spezifikation eines DEFAULT-Wertes: … *Datentyp* **`DEFAULT`** *Ausdruck*
  (i. W. nur Konstanten oder Datums- und Zeitfunktionen erlaubt)
- Das DBMS gewährleistet bei Einfüge-, Lösch- und Änderungsoperationen, dass der Wert des Primärschlüssels innerhalb dieser Tabelle stets eindeutig ist. Die Änderungsoperation wird ansonsten zurückgewiesen.
- Je Relation kann maximal ein Primärschlüssel deklariert werden.
- Deklaration (beliebig vieler) weiterer Schlüssel mittels **`UNIQUE`**-Klausel

Fremdschlüssel

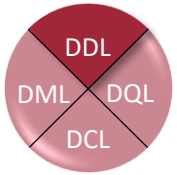


CREATE TABLE *Relationsname*$_1$ **(**
    … … ,
    Attribut$_x$ … **REFERENCES** *Relationsname*$_2$ [(*Attrname*)] ,
    …
)

alternativ (bzw. bei zusammengesetztem Fremdschlüssel zwingend):

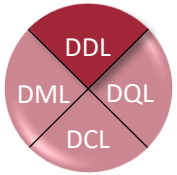
CREATE TABLE *Relationsname*$_1$ **(**
    … … ,
    Attribut$_x$ … ,
    Attribut$_y$ … ,
    …
    FOREIGN KEY (*Attrib*$_x$, *Attrib*$_y$) **REFERENCES** *Relname*$_2$ [(*Attrib*$_1$, *Attrib*$_2$)]
)

Erläuterungen



- Wenn für Fremdschlüssel keine Nullwerte (NOT NULL) zugelassen wurden, dann muss der referenzierte Wert in der referenzierten Relation existieren.
- Bei Verwendung von **… REFERENCES** *Relationsname (Attributname)* muss '*Attributname*' (bzw. die Attribut-kombination) in der referenzierten Relation entweder als **PRIMARY KEY** oder als **UNIQUE** deklariert sein.
- Bei Verwendung von **… REFERENCES** *Relationsname* wird der Primärschlüssel der referenzierten Relation verwendet.

Auswirkungen



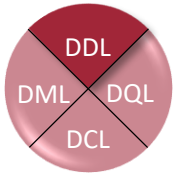
- Auswirkungen auf abhängige Relation:

  Jegliche Einfüge- oder Änderungsoperation in Bezug auf die „Kind-Relation“, die hiergegen verstößt, wird vom DBMS zurückgewiesen.

- Auswirkungen auf die referenzierte Relation
  (wenn Bedingung so spezifiziert (NOT NULL)):

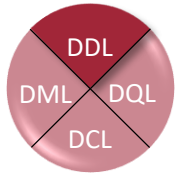
  Jegliche Änderungs- oder Löschoperation auf der referenzierten Relation, die zu fehlerhaften Referenzen („Waisenkindern“) in der referenzierenden Relation führen würde, wird vom DBMS zurückgewiesen.

Beispiel




```
CREATE TABLE TeileTypen (
   TeileNr          VARCHAR(8) NOT NULL PRIMARY KEY,
   TeileName        VARCHAR(50) NOT NULL UNIQUE )

CREATE TABLE Farbcodes (
   Farbcode         INTEGER NOT NULL PRIMARY KEY,
   FarbeText        VARCHAR(10) NOT NULL UNIQUE  )


CREATE TABLE Teile (
   TeileNr          VARCHAR(8) NOT NULL REFERENCES TeileTypen,
   Farbe            INTEGER NOT NULL REFERENCES  Farbcodes(Farbcode),
   KalkKosten       DECIMAL(6,2),
   Bestand          INTEGER,
   MinBestand       INTEGER,
   PRIMARY KEY (TeileNr, Farbe)  )
```

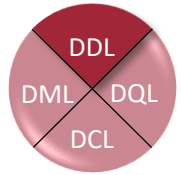
Fremdschlüsselbedingungen

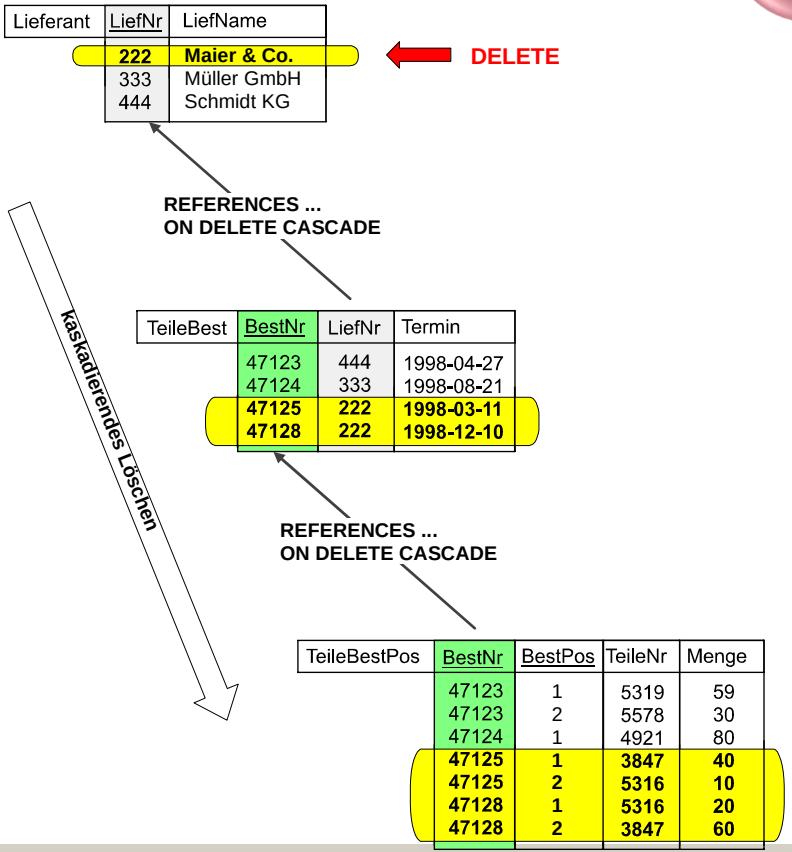



```
CREATE TABLE Relationsname (
    ...
    ... REFERENCES ... [ ON DELETE { RESTRICT | CASCADE | SET NULL } ] [ ON UPDATE { RESTRICT | CASCADE } ]
    ...
)
```

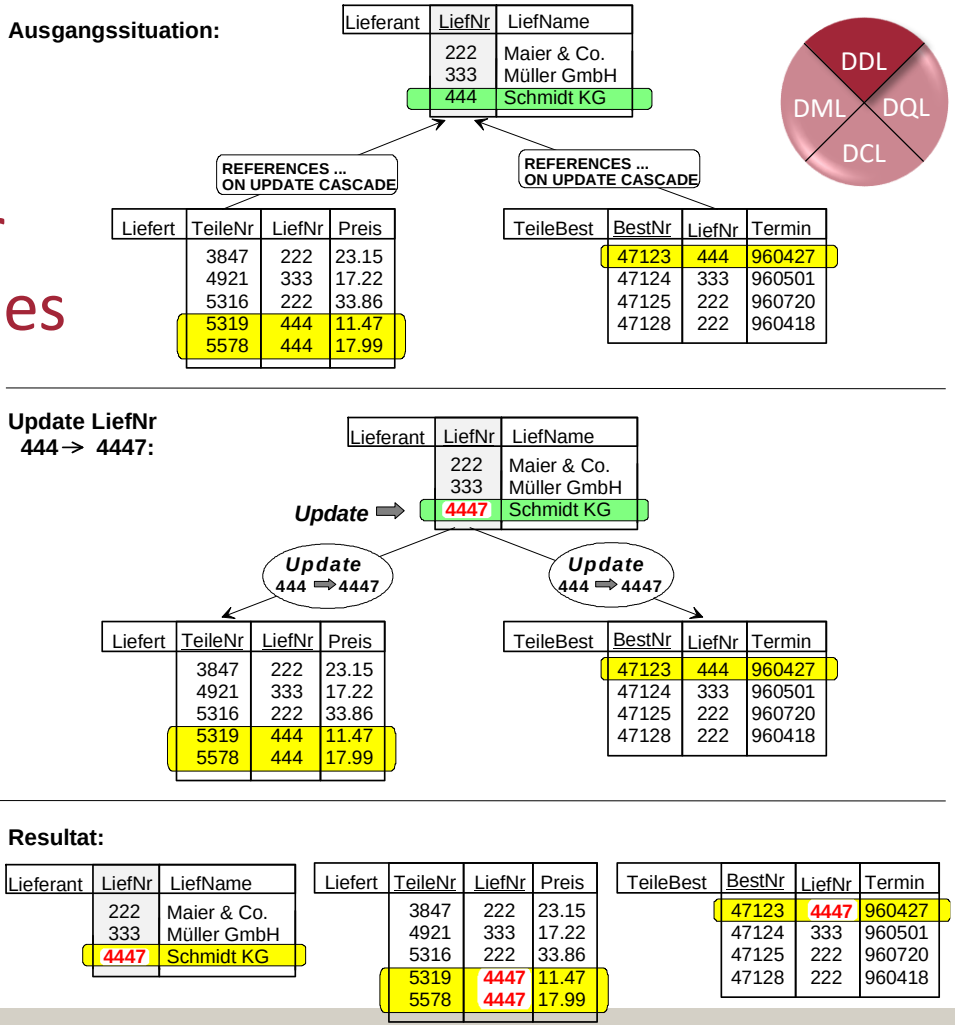
- CASCADE propagiert Löschen von Tupeln oder Update des Primärschlüssels in der referenzierten Relation auf diese Relation, sofern diese Operation hier zulässig ist.
- Kaskadierend bedeutet, dass sich diese Aktion über mehrere Relationen hinweg fortsetzen kann.
- Ausführung komplett oder gar nicht.
- DELETE … SET NULL setzt die Attribute in der abhängigen Relation auf NULL, falls erlaubt!

Beispiel für kaskadierendes Löschen

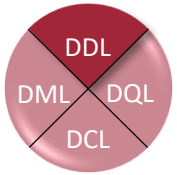




Beispiel für kaskadierendes Ändern



Check-Klauseln

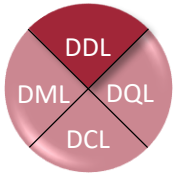



```
CREATE TABLE Lieferanten (
   LiefNr        INTEGER NOT NULL PRIMARY KEY,
   LiefName      VARCHAR(30) NOT NULL,
   LiefStadt     VARCHAR(30) NOT NULL,
   Bewertung     INTEGER
   CHECK (Bewertung IS NULL OR Bewertung BETWEEN -2 AND 2)
   )
```

Anmerkungen/Erläuterungen:

- CHECK-Bedingung muss für jedes einzufügende oder zu ändernde Tupel erfüllt sein.
- Seit SQL92 auch „freistehende“ CHECK-Klauseln möglich
  ➔ Assertions

komplexeres Beispiel

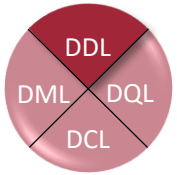



```
CREATE TABLE  Liefert (
  TeileNr INTEGER NOT NULL,
  Farbe   INTEGER NOT NULL,
  LiefNr  INTEGER NOT NULL,
  Preis   DECIMAL(8,2),
  UNIQUE (TeileNr, LiefNr),
  CHECK
  (   EXISTS
      ( SELECT *
        FROM   Teile
        WHERE  (Teile.TeileNr, Teile.Farbe) = (Liefert.TeileNr , Liefert.Farbe))  AND
  EXISTS
      ( SELECT *
        FROM   Lieferant
        WHERE  Lieferant.LiefNr = Liefert.LiefNr ) )
  )
```

existiert diese (TeileNr, Farbe)-Kombination in der Teile-Relation?

existiert LiefNr in der Lieferanten-Relation?

Probleme mit Integrationsbedingungen

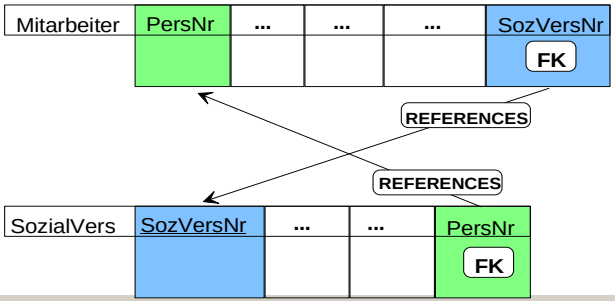


- Massenänderungen

  Ständige Überprüfung der Integritätsbedingungen bei Massen-Inserts, -Updates oder -Deletes u.U. sehr ineffizient.

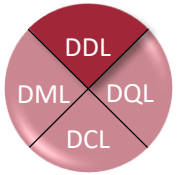
  Besser wäre in solchen Fällen eine Überprüfung en bloc.

- Wechselseitige Referenzierung



Falls bei den FKs keine Nullwerte erlaubt sind, sind keine Einfügungen möglich

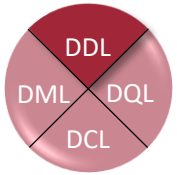
benamte Integritätsbedingungen



- Alle bisher dargestellten Integritätsbedingungen waren unbenamte Integritätsbedingungen.
- Gezieltes Löschen bzw. (temporäres) Außerkraftsetzen nur möglich mit benamten Integritätsbedingungen
- Beispiel:

```
CREATE TABLE Stueckliste (
   Teil            VARCHAR(8) NOT NULL,
   TeilFarbe       INTEGER NOT NULL,
   UTeil           VARCHAR(8) NOT NULL,
   UTeilFarbe      INTEGER NOT NULL,
   Anzahl          INTEGER NOT NULL,
   CONSTRAINT fk1 FOREIGN KEY (Teil, TeilFarbe) REFERENCES Teile,
   CONSTRAINT fk2 FOREIGN KEY (UTeil, UTeilFarbe) REFERENCES Teile)
```

Lösung der Probleme

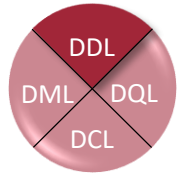


1. Mittels der ALTER TABLE-Anweisung (nächste Folie)
   Das „große“ Geschütz: bewirkt eine Schema-Änderung (inkl. Katalog-Update)

2. Durch gezieltes Ein-/Ausschalten innerhalb einer Transaktion
   ➔ Deferred Constraint Checking

```
SET CONSTRAINTS { ALL | constraint-name(n) } DEFERRED
    SQL-Anweisung1
    ...
    SQL-Anweisungn
SET CONSTRAINTS { ALL | constraint-name(n) } IMMEDIATE
```

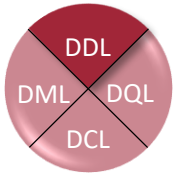
Ändern einer Relation



Auszug

```
ALTER TABLE Relationsname
{
  ADD { [ COLUMN ] column-definition
      | { unique-constraint
        | referential-constraint
        | check-constraint } }
| ALTER COLUMN column-name column-alteration
| DROP { PRIMARY KEY
       | { FOREIGN KEY constraint-name
         | UNIQUE constraint-name
         | CHECK constraint-name
         | CONSTRAINT constraint-name }
       | COLUMN columnname [ { RESTRICT
                             | CASCADE } ] }
}
```

Beispiel

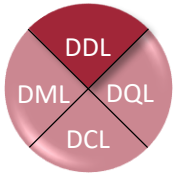



```
ALTER TABLE Lieferanten
    ADD Plz DECIMAL(5)
    ADD Strasse VARCHAR(30)
```

Anmerkungen

- **ADD COLUMN**
  - Neue Spalten werden stets hinten angehängt.
  - Sofern die Relation bereits Tupel enthält, werden die neuen Attribute mit Nullwerten gefüllt ➔ NOT NULL-Klausel nur bei leeren Relationen
- **DROP COLUMN**
  - Per Default werden alle von diesem Attribut abhängigen Indexe, Views, Trigger etc. ebenfalls gelöscht (➔ CASCADE). Bei Angabe der RESTRICT-Klausel wird die Anweisung zurückgewiesen, wenn (noch) eine solche Abhängigkeit existiert

Sichten



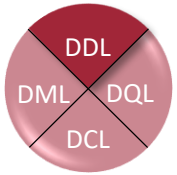
- Sichtenkonzept ein sehr wichtiges Mittel zur Erhöhung der Datenunabhängigkeit von Anwendungsprogrammen
- Realisierung „virtueller“ (abgeleiteter) Relationen
  - „Ausblenden“ von Attributen (Datenschutz-, Vertraulichkeitsaspekte)
  - (Versteckte) Vorformulierung von Anfragen
  - Verstecken von physischen Details vor den Anwendungsprogrammen, wie z.B. Aufspaltung einer Relation in zwei Relationen
- **Syntax:**

CREATE VIEW *Viewname* [($Attrib_1$, $Attrib_2$, …, $Attrib_n$)]
AS *Query* **[WITH CHECK OPTION]**

… und Löschen:

DROP VIEW *Viewname*

Beispiele



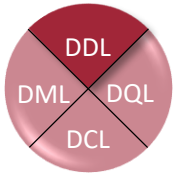
V01: *„Erzeuge eine virtuelle Relation TeileNrFarbe (TeileNr, Farbe, Farbcode)“*

```
CREATE  VIEW TeileNrFarbe AS
        SELECT  t.TeileNr, f.FarbeText AS Farbe, t.Farbe AS FarbCode
        FROM    Teile AS t
                JOIN  Farbcodes AS f ON t.Farbe = f.FarbCode
```

V02: *„Erzeuge eine virtuelle Relation BestellWerte(Bestellung, Lieferant, BestellPos), mit Bestellung = BestNr, Lieferant = LiefNr und BestellPos = Anzahl jeweiliger Bestellpositionen“*

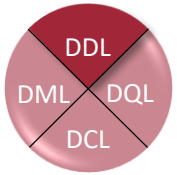
```
CREATE  VIEW BestellWerte (Bestellung, Lieferant, BestellPos) AS
        SELECT  b.BestNr, b.LiefNr, COUNT(*)
        FROM    Bestellungen AS b
                JOIN BestellPos AS p ON b.BestNr = p.BestNr
        GROUP BY  b.BestNr, b.LiefNr
```

Anmerkungen zu Views



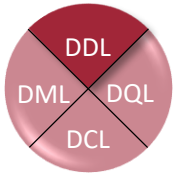
- Bei „guten“ DBMS keine eigenen Tabellen für View, sondern Transformation der Anfrage auf Anfrage gegen Basis-Relationen
- daher typischerweise keine Performanzeinbußen gegenüber Anfragen auf Basisrelationen
- **UPDATE** und **DELETE** auch über Views möglich, wenn
  - sich die Operation eindeutig zu einer (Basis-)Relation zuordnen lässt
  - die View-Definition keine DISTINCT-Klausel enthält
  - die View-Definition keine GROUP-Functions (AVG, SUM, ...) enthält (normale „Expressions“ sind erlaubt!)
- **INSERT** über Views sind möglich, wenn UPDATE-/DELETE-Restriktionen erfüllt sind, keine „Expressions“ vorkommen und keine NOT-NULL-Attribute durch die View ausgeblendet werden.

Schemas



- Schemas dienen zur Kategorisierung der Relationen einer (Gesamt-) Datenbank, z.B.
  - alle Relationen zur Teileverwaltung: Teile_Schema
  - alle Relationen zur Personalverwaltung: Personal_Schema
  - alle Relationen zur Buchhaltung: Buchhaltung_Schema
- wird beim CREATE TABLE nichts explizit angegeben, so wird eine Relation dem Default-Schema zugeordnet
- **CREATE TABLE** [*schema*.]*relname* bewirkt Eintrag der Relation in den DB-Katalog und Zuordnung zum (explizit oder implizit) angegebenen Schema.
- Relationsnamen innerhalb eines Schemas müssen eindeutig sein.

Schemas



- Der DB-Katalog ist selbst wieder als Sammlung von Relationen realisiert ➔ Katalog-Relationen können wie normale Relationen abgefragt werden.
- Evtl. spezielle Zugriffs-Privilegien erforderlich
- In HyperSQL: INFORMATION_SCHEMA
  - INFORMATION_SCHEMA.TABLES
  - INFORMATION_SCHEMA.COLUMNS
  - INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS
  - INFORMATION_SCHEMA.SYSTEM_USERS
- Bei manchen DBMS (z.B. mySQL) Wechsel zwischen Schemas mit '**USE** *Schema*' möglich

Ziele

- Daten in einer Datenbank einfügen, verändern können und entfernen können
- Datenbank mit verschiedenen Integritätsbedingungen anlegen können
- Sichten und Schemas verstehen