7. Integritätskontrolle in SQL

Vorlesung "Informationssysteme"

Sommersemester 2017

Technische Universität
Kaiserslautern

Überblick

- Integritätsbedingungen
  - Klassifikation und Eigenschaften
  - Referentielle Integrität und referentielle Aktionen
  - CHECK-Constraints und Assertions
- Trigger
  - Motivation und Grundprinzip
  - Eigenschaften und Ausführung
  - Beispiele
  - Auswertungsmodell

Integritätsbedingungen

- Ziel: Verankerung von semantischen Integritätsbedingungen im DB-Schema
  - Semantik der Mini-Welt möglichst vollständig erfassen
  - Integritätsbedingungen beschreiben akzeptable DB-Zustände
    - Änderungen werden zurückgewiesen, wenn sie Integrität verletzen
  - effiziente Integritätskontrolle durch das DBMS
    - Konsistenzgarantie, auch für interaktive Änderungen
    - vereinfachte Anwendungsentwicklung
    - leichte Änderbarkeit von Integritätsbedingungen
- Überblick
  - Startpunkt: Verfeinerte Abbildung von ER-Schemata
    - PRIMARY KEY, FOREIGN KEY ... REFERENCES, UNIQUE, NOT NULL
  - Prüfzeitpunkt (IMMEDIATE, DEFERRED)
  - Referentielle Constraints und Aktionen
  - CHECK-Constraints und Assertions

Arten von Integritätsbedingungen

- Integritätsbedingungen abhängig vom Relationenmodell
  - Primärschlüsseleigenschaft
  - Referentielle Integrität für Fremdschlüssel
  - Definitionsbereiche (Domains) für Attribute
- Reichweite der Bedingung
  - Attributwert-Bedingungen (z.B. Geburtsjahr > 1900)
  - Satzbedingungen (z.B. Geburtsdatum < Einstellungsdatum)
  - Satztyp-Bedingungen (z.B. Eindeutigkeit von Attributwerten)
  - Satztypübergreifende Bedingungen (z.B. referentielle Integrität zwischen verschiedenen Tabellen)
- Klar, je geringer die Reichweite, desto einfacher lassen sich Bedingungen überprüfen.

Arten von Integritätsbedingungen (2)

- Statische vs. dynamische Bedingungen
  - Statische Bedingungen (Zustandsbedingungen): beschränken zulässige DB-Zustände (z.B. Gehalt < 500000)
  - Dynamische Integritätsbedingungen (Übergangsbedingungen): zulässige Zustandsübergänge (z.B. Gehalt darf nicht kleiner werden)
  - Variante dynamischer Integritätsbedingungen: temporale IBs für längerfristig
- Zeitpunkt der Überprüfbarkeit: unverzögert vs. verzögert
  - Verzögerte Bedingungen lassen sich nur durch eine Folge von Änderungen erfüllen (typisch: mehrere Sätze, mehrere Tabellen) und
  - Benötigen Transaktionsschutz (als zusammengehörige Änderungssequenzen)

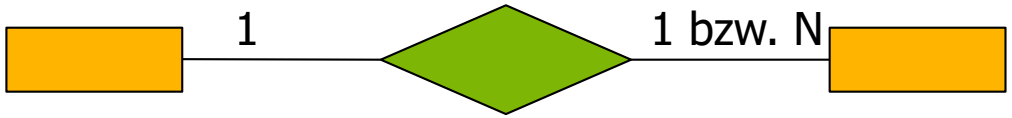
Eindeutigkeit, Fremdschlüssel und Verbot von Nullwerten

- Bekannt aus Kapitel 5: Spezifikation grundlegender Integritätsbedingungen (Constraints)
  - Verbot von Nullwerten (NOT NULL)
  - Schlüsselkandidaten (UNIQUE bzw. PRIMARY KEY)
  - Fremdschlüssel (FOREIGN-KEY ... REFERENCES)
- Beispiel:

```
CREATE TABLE PERS
    (PNR        INT             PRIMARY KEY,
     BERUF      CHAR (30),
     PNAME      CHAR (30)       NOT NULL,
     PALTER     ALTER,          (* siehe Domaindefinition *)
     MGR        INT             REFERENCES PERS,
     ANR        ABTNR           NOT NULL, (* Domaindef. *)
     W_ORT      CHAR (25)       DEFAULT  ' ',
     GEHALT     DEC (9,2)       DEFAULT 0,00,
     FOREIGN KEY (ANR)          REFERENCES ABT)
```

Abbildung von ER-Schemata in SQL

- Abbildung folgt dem in Kapitel 3 (und 4) vorgestellten Verfahren
  - Erzeugen von Tabellen für Entities und (N:M)-Relationships
  - Definition von geeigneten Primärschlüsseln (PRIMARY KEY)
  - Definition von Fremdschlüsseln (FOREIGN-KEY)
    - direkte Abbildung von 1:1, 1:N - Beziehungen
    - FOREIGN KEY ... UNIQUE zur Abbildung von 1:1-Beziehung

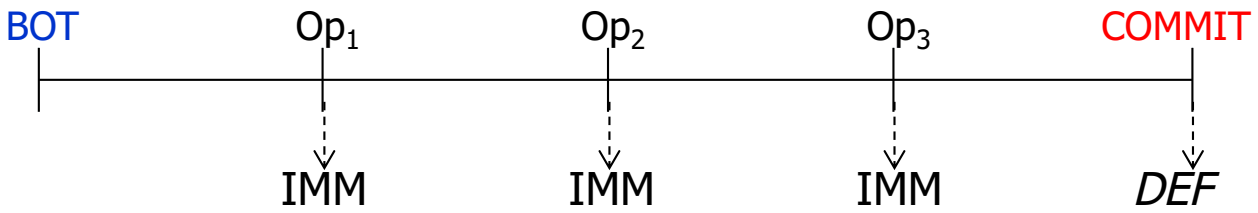


(0,n)	(0,1)	FK
(0,n)	(1,1)	FK ... NOT NULL
(0,1)	(0,1)	FK ... UNIQUE
(0,1)	(1,1)	FK ... UNIQUE NOT NULL

Prüfung von Integritätsbedingungen

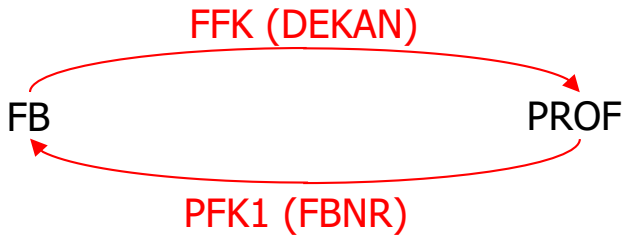
- Oft sollen Integritätsbedingungen (IBen) schon direkt nach Abschluss einer Änderungsoperationen erfüllt sein
  - Prüfzeit IMMEDIATE in SQL (ist auch der Default)
  - Falls IB nach Abschluß einer DML-Operation verletzt, scheitert die DML-Operation vollständig (d.h., hat keine Auswirkungen auf die DB)
- Manchmal (z.B. bei tupelübergreifenden IBen) kann ein konsistenter DB-Zustand erst nach mehreren DML-Befehlen erreicht werden
  - Prüfzeit DEFERRED in SQL
- Transaktionskonzept (→ ACID) fordert Erhaltung der sem. Integrität (Konsistenz) durch jede Transaktion

  ➔spätester Prüfzeitpunkt: Ende der Transaktion (Commit)
  - falls IBen nicht erfüllt, dann scheitert die ganze Transaktion!



IMMEDIATE und DEFERRED

- Beispiel: neuer Fachbereich entsteht
  - INSERT INTO FB (FB13, ..., 1234, ...)
  - INSERT INTO PROF(1234, ..., FB13, ...)
- Bei zyklischen Referenzpfaden
  - wenigstens ein Fremdschlüssel im Zyklus muss „NULL“ erlauben oder
  - Prüfung der referentiellen Integrität muss für mindestens einen FK verzögert (DEFERRED) werden (z. B. bei COMMIT)



- Prüfzeitpunkt (deferrability) kann für jede IB definiert werden
- Im Beispiel:

```
deferrability
::=  [INITIALLY {DEFERRED | IMMEDIATE}]
       [ [NOT] DEFERRABLE ]
```

```
CREATE TABLE FB (
  FBNR      FACHBEREICHSNUMMER    PRIMARY KEY,
  FBNAME    FACHBEREICHSNAME      UNIQUE,
  DEKAN     PERSONALNUMMER        UNIQUE NOT NULL,
     CONSTRAINT FFK FOREIGN KEY (DEKAN)
                REFERENCES PROF (PNR)
                INITIALLY DEFERRED)
```

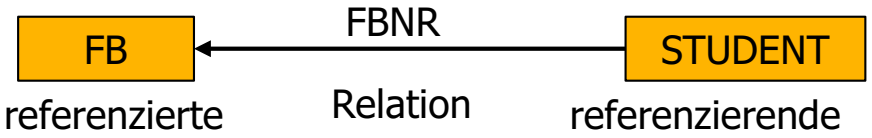
Ändern/Setzen des Prüfzeitpunkts

- SET CONSTRAINTS { constr. ... | ALL } IMMEDIATE | DEFERRED
  - Setzt in der aktuellen Transaktion Prüfzeitpunkt für benannte bzw. alle IBen
    - SET CONSTRAINTS ALL DEFERRED hat nur Auswirkungen auf IBen, die DEFERRABLE sind
  - SET CONSTRAINTS ... IMMEDIATE bewirkt die sofortige Überprüfung der genannten IBen
    - Beispiel:

      INSERT INTO FB (FB13, ..., 1234, ...) //FFK ist INITIALLY DEFERRED!
      INSERT INTO PROF(1234, ..., FB13, ...) //PFK1 ist wird geprüft!
      SET CONSTRAINTS FFK **IMMEDIATE** //FFK wird geprüft!
  - SET CONSTRAINTS schlägt fehl, falls IB verletzt!
    - TA scheitert (noch) nicht, könnte DB-Zustand noch konsistent machen!
- COMMIT
  - impliziert SET CONSTRAINTS ALL IMMEDIATE
  - TA scheitert (wird zurückgesetzt), falls IB verletzt!

Referentielle Integrität (RI)

- Fremdschlüsselbedingung: Zugehöriger PS (SK) muss existieren*



- Welche Operationen führen potentiell zu RI-Verletzungen?
  - Operationen in der referenzierenden Relation (enthält FS)
    - Einfügen eines Tupels
    - Ändern des FS-Wertes in einem Tupel
    - Löschen eines Tupels ist unkritisch (warum?)
  - Operationen in der referenzierten Relation (enthält PS/SK)
    - Löschen eines Tupels
    - Ändern des PS/SK-Wertes
    - Einfügen eine Tupels ist unkritisch (warum?)

(*) *Achtung: falls FS zusammengesetzt ist, kann in SQL zusätzlich definiert werden, wie Nullwerte für Teile des Schlüssels interpretiert werden (MATCH-Klausel). Darauf wird hier nicht weiter eingegangen.*

Wartung der referentiellen Integrität (RI)

- Welche Maßnahmen sind möglich/sinnvoll?
  - beim Einfügen/Ändern in der referenzierenden Tabelle
    - Prüfung, ob in der referenzierten Tabelle ein Tupel mit einem PS/SK-Wert gleich dem FS-Wert des einzufügenden/zu ändernden Tupels existiert.
    - Operation wird abgewiesen, falls ein solches Tupel nicht existiert
  - beim Löschen/Ändern in der referenzierten Tabelle
    1. Operation verbieten, falls es noch referenzierende Tupel gibt
    2. Löschen bzw. Ändern der FS in allen referenzierenden Tupeln
    3. Erhalten der referenzierenden Tupel durch Setzen von Default- bzw. NULL-Werten für FS (falls das erlaubt ist)
- SQL unterstützt referentielle Aktionen, um bei Löschen/Ändern in der referenzierten Tabelle die gewünschte Maßnahme festzulegen
  - Erlaubt Standardmaßnahmen zum Vermeiden von RI-Verletzungen durch das DBMS! (Aktives Verhalten)

Referentielle Aktionen

```
references-def ::=
        REFERENCES base-table [(column-commalist)]
        [ON DELETE  referential-action]
        [ON UPDATE  referential-action]

referential-action
::=  NO ACTION | CASCADE | SET DEFAULT | SET NULL | RESTRICT
```

- Referentielle Aktionen (*referential actions*)
  - für jeden Fremdschlüssel (FS) separat festzulegen
  - Angabe der gewünschten Aktionen bei Löschen/Ändern von Tupeln in der referenzierten Relation
    - Löschregel: ON DELETE ...
    - Änderungsregel: ON UPDATE ...
  - unterschiedliche Maßnahmen für DELETE und UPDATE möglich
- Durchführung von referentiellen Aktionen
  - immer sofort bei der Ausführung der Änderungsoperation
  - vor der Prüfung der RI-Bedingung
  - unabhängig vom Prüfzeitpunkt (IMMEDIATE/DEFERRED)!
  - verursacht ggf. weitere referentielle Aktionen

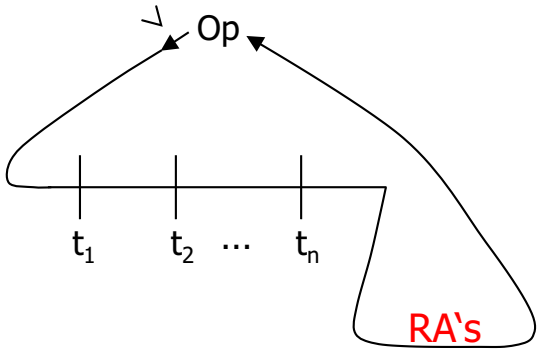
Referentielle Aktionen (2)

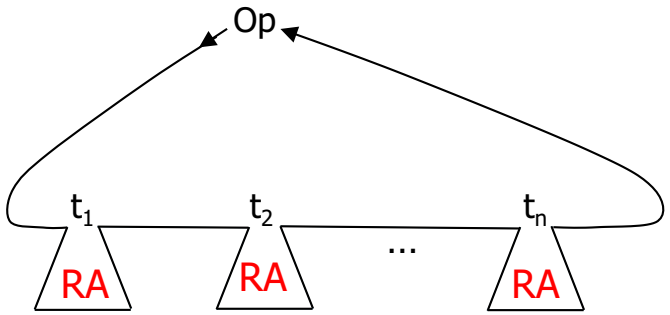
- Bedeutung der einzelnen Aktionen
  - NO ACTION (Defaulteinstellung) - keine referentielle Aktion
    - Prüfung der RI erfolgt zum definierten Zeitpunkt (evtl. DEFERRED), nachdem die referentiellen Aktionen aller IBen ausgeführt wurden
  - CASCADE - Operation „kaskadiert“ zu allen zugehörigen Sätzen

    ➔ Existenzabhängigkeit (z.B. für schwache Entities)
    - DELETE CASCADE: referenzierende Tupel werden gelöscht
    - UPDATE CASCADE: FS in referenzierenden Tupeln wird geändert
  - SET NULL - FS wird in zugehörigen Sätzen auf „NULL“ gesetzt
  - SET DEFAULT - FS wird in den zugehörigen Sätzen auf den (benutzerdefinierten) Default-Wert gesetzt
  - RESTRICT – die Operation wird nur ausgeführt, wenn keine zugehörigen Sätze (FS-Werte) vorhanden sind
    - ist restriktiver als NO ACTION, da Operation sofort zurückgewiesen wird
- Referentielle Aktion ersetzt nicht generell die Prüfung der RI!
  - Prüfung bei SET DEFAULT und NO ACTION erforderlich

Durchführung der Änderungsoperationen

- Durchführung der referentiellen Aktionen (RA)
  - Benutzeroperationen (Op) sind in SQL immer atomar
  - mengenorientiertes oder satzorientiertes (in-flight) Verarbeitungsmodell

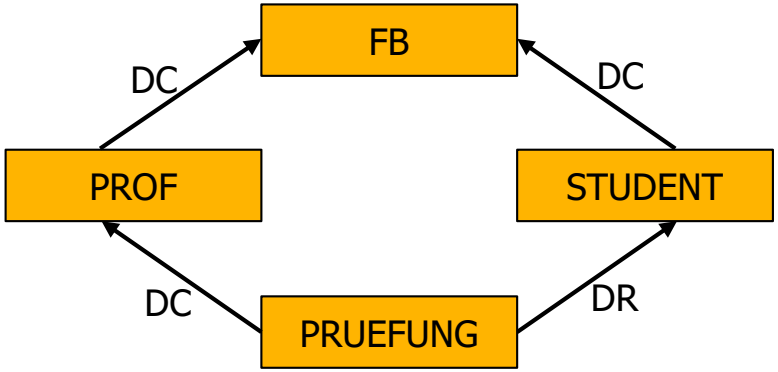




  - IMMEDIATE-Bedingungen müssen erfüllt sein an Anweisungsgrenzen (➥ mengenorientierte Änderung)
  - Satzorientiertes Modell darf nur genutzt werden, wenn Äquivalenz zum mengenorientierten Modell garantiert ist
    - Beipiel: PERS.MGR → PERS.PNR (RESTRICT)
      Lösche alle Angestellten aus Abteilung K55, inklusive Manager

Auswirkungen referentieller Aktionen

- Operation: Lösche FB (FBNR=FB9)



erst links

- Löschen in FB
- Löschen in PROF
- Löschen in PRUEFUNG
- Löschen in STUDENT
- Überprüfen in PRUEFUNG

Wenn ein Student bei einem FB-fremden Professor geprüft wurde
→ Rücksetzen

erst rechts

- Löschen in FB
- Löschen in STUDENT
- Überprüfen in PRUEFUNG

Wenn ein gerade gelöschter Student eine Prüfung abgelegt hatte
→ Rücksetzen

sonst:

- Löschen in PROF
- Löschen in PRUEFUNG

  - Schema ist nicht sicher
  - Es können reihenfolgenabhängige Ergebnisse auftreten!
    - In SQL sind solche Konflikte verboten, wird zur Laufzeit überwacht!

- Verwendung von NO ACTION anstelle RESTRICT
  - Bei der NA-Option wird der explizite Test der referenzierenden Relation ans Ende der Operation verschoben. Eine Verletzung der referentiellen Beziehung führt zum Rücksetzen.
  - Schema ist immer sicher

Spezifikation des relationalen DB-Schemas

- Wertebereiche

```
CREATE DOMAIN FACHBEREICHSNUMMER AS CHAR (4)
CREATE DOMAIN FACHBEREICHSNAME
                              AS VARCHAR (20)
CREATE DOMAIN FACHBEZEICHNUNG AS VARCHAR (20)
CREATE DOMAIN NAMEN   AS VARCHAR (30)
CREATE DOMAIN PERSONALNUMMER     AS CHAR (4)
CREATE DOMAIN MATRIKELNUMMER     AS INT
CREATE DOMAIN NOTEN              AS SMALLINT
CREATE DOMAIN DATUM              AS DATE
```

- Relationen

```
CREATE TABLE FB (
 FBNR        FACHBEREICHSNUMMER    PRIMARY KEY,
 FBNAME      FACHBEREICHSNAME      UNIQUE,
 DEKAN   PERSONALNUMMER    UNIQUE NOT NULL,
       CONSTRAINT FFK     FOREIGN KEY (DEKAN)
                          REFERENCES PROF (PNR)
                          ON UPDATE CASCADE
                          ON DELETE NO ACTION
                          INITIALLY DEFERRED)
CREATE TABLE PROF (
 PNR         PERSONALNUMMER        PRIMARY KEY,
 PNAME       NAMEN                 NOT NULL,
 FBNR        FACHBEREICHSNUMMER    NOT NULL,
 FACHGEBIET  FACHBEZEICHNUNG,
       CONSTRAINT PFK1   FOREIGN KEY (FBNR)
                         REFERENCES FB (FBNR)
                         ON UPDATE CASCADE
                         ON DELETE SET DEFAULT)
```

```
CREATE TABLE STUDENT (
 MATNR      MATRIKELNUMMER          PRIMARY KEY,
 SNAME      NAMEN                   NOT NULL,
 FBNR       FACHBEREICHSNUMMER      NOT NULL,
 STUDBEG    DATUM,
       CONSTRAINT SFK     FOREIGN KEY (FBNR)
                          REFERENCES FB (FBNR)
                    ON UPDATE CASCADE
                 ON DELETE NO ACTION)

CREATE TABLE PRUEFUNG (
 PNR        PERSONALNUMMER,
 MATNR      MATRIKELNUMMER,
 FACH       FACHBEZEICHNUNG,
 PDATUM     DATUM                   NOT NULL,
 NOTE       NOTEN                   NOT NULL,
       PRIMARY KEY (PNR, MATNR),
       CONSTRAINT PR1FK FOREIGN KEY (PNR)
                        REFERENCES PROF (PNR)
                        ON UPDATE CASCADE
                        ON DELETE CASCADE,
       CONSTRAINT PR2FK FOREIGN KEY (MATNR)
                        REFERENCES STUDENT
(MATNR)
                        ON UPDATE CASCADE
                        ON DELETE CASCADE)
```

// Es wird hier darauf verzichtet, die Rückwärtsrichtung der „ist-Dekan-von"-Beziehung explizit als Fremdschlüsselbeziehung zu spezifizieren. Damit fällt auch die mögliche Spezifikation von referentiellen Aktionen weg.

Statische Integritätsbedingungen – CHECK-Constraints

- CHECK-Klausel

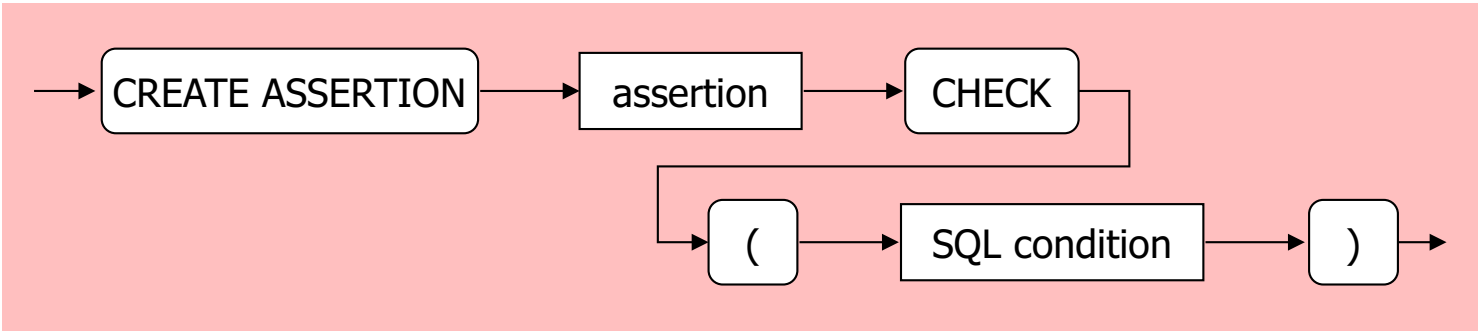
  CHECK (cond-exp)

  - cond-exp ist eine beliebige SQL-Suchbedingung
- Verwendung als
  - Wertebereichsbedingung (VALUE bezeichnet zul. Wert)
    - **CREATE DOMAIN** ALTER **AS INT CHECK** (VALUE > 18 **AND** VALUE < 70)
  - Attributbedingung (kann Attribut referenzieren)
    - **CREATE TABLE** PERS
      (...
      GEHALT **DEC** (9,2) **CHECK** (GEHALT < 120.000,00))
  - Tabellenbedingung (kann alles Attribute der Tabelle referenzieren)
    - **CREATE TABLE** PERS
      (...
      CHECK (GEHALT < (120.000,00 * (ALTER / 18)))
- Bedeutung: Es darf kein Tupel in der Tabelle geben, für das die CHECK-Bedingung den Wahrheitswert *false* ergibt
  - *unknown* durch Nullwerte führt nicht zur Integritätsverletzung!
  - leere Tabelle erfüllt "jede" Integritätsbedingung

Tabellenübergreifende CHECK-Constraints

- CHECK-Constraints können beliebige Tabellen der DB referenzieren
- Beispiele
  - **CREATE TABLE** PRODUKT
    (...
    Verkaufs_Preis **DECIMAL** (9, 2)
    CHECK (Verkaufs_Preis <=
    (**SELECT MIN** (Preis) **FROM** Konkurrenz_Preise)))
  - **CREATE TABLE** ABT
    (ANR ABTNR **PRIMARY KEY**,
    ANZAHL-ANGEST **INT** **NOT NULL**
    CHECK (ANZAHL-ANGEST =
    (**SELECT COUNT**(*) **FROM** PERS P **WHERE** P.ANR = ANR))
    . . .)
  - **CREATE TABLE** PERS
    (...
    CHECK (GEHALT <
    (**SELECT** GEHALT **FROM** PERS M **WHERE** MNR=M.MNR))

SQL-Zusicherungen (*Assertions*)



- Zusicherungen (assertions) sind allgemeine Integritätsbedingungen
  - beziehen sich oft auf mehrere Relationen
    - natürlichere Formulierung im Vergleich zu Tabellenbedingungen
  - lassen sich als eigenständige DB-Objekte definieren
- Eine Zusicherung gilt als erfüllt, wenn ihre SQL-Bedingung nicht zu *false* evaluiert wird
- Beispiele
  - **CREATE ASSERTION** ANZ-ANG
    CHECK (**NOT EXISTS**
    (**SELECT** * **FROM** ABT A **WHERE** A. ANZAHL-ANGEST <>
    (**SELECT COUNT** (*) **FROM** PERS P **WHERE** P.ANR = A.ANR)))
    INITIALLY DEFERRED
  - **CREATE ASSERTION** EXIST-ABT
    CHECK ((**SELECT COUNT** (*) **FROM** ABT) > 0)

Trigger-Konzept

- Assertions/Constraints legen fest was erlaubt ist.
- Trigger ermöglichen, auf bestimmte Ereignisse zu reagieren!
  - Z.B wenn neue Zeilen in eine Tabelle bzw. Sicht eingefügt wurden bzw. werden sollen
  - Tritt ein bestimmtes Ereignis auf so wird ein Trigger "gefeuert", d.h. eine vorher festgelegte Aktion (Prozedur bzw. Funktion) ausgeführt.
- Somit können z.B. komplexe Constraints überprüft werden oder Tabellen automatisch angepasst werden
- Allgemeines Prinzip: ECA - Event, Condition, Action

Beispiel für Einsatz von Triggern

- Automatisches Nachbestellen, wenn Lagerbestände leerlaufen
- Automatisches Anschreiben von Studenten, wenn creditPoints < vordefinierte Schranke
- Automatische Mahnung für Rechnungen/ablaufende Abos/etc.
- Bibliothek: Anschreiben von Nutzern wenn Ausleihfrist überschritten
- Überwachung von Kontobeständen
- Obergrenzen für Überweisungen
- Untergrenzen für Kontostände

Ein Teil der Anwendungslogik ist dann in den Triggern definiert.

Beispiel Trigger

- Idee: Jeder neue Student wird automatisch durch einen Eintrag in der Tabelle *hören* für die Vorlesung Logik registriert.
  - Also: **AFTER INSERT ON** studenten ...
- Und für jede neue Zeile einzeln, also **FOR EACH ROW**
- **CREATE TRIGGER** pflichtvorlesungLogikTrigger
  AFTER INSERT ON studenten **REFERENCING NEW AS** stud
  FOR EACH ROW
  INSERT INTO hoeren **VALUES** (stud.matrnr, 4052);

Wann soll ein Trigger ausgelöst werden?

- Auslösende Operation
  - INSERT / DELETE / UPDATE auf einer Tabelle (Triggertabelle)
  - Bei UPDATE können auch einzelne Spalten angegeben werden
- Zeitpunkt: BEFORE / AFTER / INSTEAD OF
  - BEFORE: direkt vor der auslösenden Operation
    - erweiterte Constraintüberprüfung (z.B. dynamische Constraints)
  - AFTER : direkt nach der auslösenden Operation
    - automatische Wartung von Constraints durch Folgeänderungen
    - Audit trail logging, "externe" Aktionen (mit Hilfe von benutzerdef. Funktionen/Prozeduren, siehe nächstes Kapitel)
    - ausführen von Anwendungslogik in der Datenbank
  - INSTEAD OF: anstelle der auslösenden Operation
    - erlaubt benutzerdefinierte Abbildung von Änderungsoperationen auf Sichten

Trigger – Granularität und Ausführungsbedinungen

- Auslösende Ereignisse/Operationen sind mengenorientiert. Wie oft soll die Trigger-Aktion ausgeführt werden?
  - FOR EACH ROW: einmal pro Tupel, das in der Operation geändert/eingfügt/gelöscht wurde
  - FOR EACH STATEMENT: einmal pro Ausführung der vollständigen Operation
- Ausführung der Trigger-Aktion kann optional an eine logische Bedingung (WHEN-Bedingung) geknüpft werden
  - logisches Prädikat, vergleichbar mit WHERE-Klausel
    - Aktion wird nur ausgeführt, wenn die Bedingung erfüllt ist
  - kann sich auf Änderungsinformation beziehen
  - kann auf beliebige Tabellen der DB zugreifen
    - Trigger-Ausführung vom Zustand der DB abhängig

Wie spezifiziert man Trigger-Aktionen?

- Welche Aktionen kann ein Trigger ausführen?
  - (Sequenz von) DML-Operation (SELECT, INSERT, UPDATE, DELETE)
  - Aufruf von benutzerdefinierten Prozeduren (siehe Kapitel 8)
  - BEFORE-Trigger dürfen auch die Werte des neuen/geänderten Tupels verändern (mit SET-Klausel)!
- Bezug auf verschiedene DB-Zustände erforderlich
  - OLD/NEW erlaubt Referenz von alten/neuen Werten in Aktionen und in der WHEN-Bedingung
  - Übergangstabellen (Transition Tables) OLD/NEW TABLE
    - enthalten für alle Tupel, die von der auslösenden DML-Operation betroffen sind, den Zustand vor bzw. nach der Änderung
  - Übergangsvariablen (Transition Variables) OLD/NEW ROW
    - ermöglichen Referenz auf alte und neue Werte für einzelne Tupel
    - nur bei "FOR EACH ROW"-Triggern erlaubt
  - Namen von Übergangstabellen/-variablen werden in einer Referenzklausel definiert (z.B., REFERENCING OLD AS …)

Trigger-Syntax (SQL Standard)

```
CREATE TRIGGER <trigger name>
    { BEFORE | AFTER | INSTEAD OF }
    { INSERT | DELETE | UPDATE [ OF <trigger column list> ] }
    ON <table name> [ REFERENCING <transition table or variable> ... ]
    [ FOR EACH { ROW | STATEMENT } ]
    [WHEN ( <search condition> ) ]
    [ BEGIN ATOMIC ]
       <SQL procedure statement> ; [<SQL procedure statement> ; ... ]
    [ END ]

<transition table or variable> ::=
    OLD [ ROW ] [ AS ] <old transition variable name>
    | NEW [ ROW ] [ AS ] <new transition variable name>
    | OLD TABLE [ AS ] <old transition table name>
    | NEW TABLE [ AS ] <new transition table name>
```

Erlaubte Kombinationen

Granularität	Aktivierungszeit	Ereignis	Erlaubte Übergangsvariab.	Erlaubte Übergangstabellen
ROW	BEFORE	INSERT	NEW	*keine*
		UPDATE	OLD, NEW	
		DELETE	OLD	
	AFTER *oder* INSTEAD OF	INSERT	NEW	NEW TABLE
		UPDATE	OLD, NEW	OLD / NEW TABLE
		DELETE	OLD	OLD TABLE
STATEMENT	BEFORE	INSERT	*keine*	*keine*
		UPDATE		
		DELETE		
	AFTER *oder* INSTEAD OF	INSERT	*keine*	NEW TABLE
		UPDATE		OLD / NEW TABLE
		DELETE		OLD TABLE

Beispiel

- Annahme: ABT enthält Attribut "Gehaltssumme", das automatisch gewartet werden soll.

```
CREATE TRIGGER GehSumUpd
AFTER UPDATE OF GEHALT ON PERS
REFERENCING OLD AS oldpers, NEW AS newpers
FOR EACH ROW
UPDATE ABT
 SET GEHALTSSUMME =
       GEHALTSSUMME + newpers.GEHALT – oldpers.GEHALT
 WHERE ANR = newpers.ANR;
```

- Weitere Trigger müssen analog für INSERT und DELETE erzeugt werden.

Beispiel (2)

- Idee: Das GEHALT von Mitarbeitern darf nicht fallen. Falls das neue Gehalt kleiner ist als das alte, wird das alte übernommen.

```
CREATE TRIGGER GehaltFaelltNicht
BEFORE UPDATE OF GEHALT ON PERS
REFERENCING OLD AS oldpers, NEW AS newpers
FOR EACH ROW
WHEN (newpers.GEHALT < oldpers.GEHALT)
     SET newpers.GEHALT = oldpers.GEHALT;
```

- Die Trigger-Aktion "überschreibt" das Gehalt, das im auslösenden Update angegeben war! Alternativ könnte man auch einen Fehler melden und damit das auslösende UPDATE abbrechen:

```
… WHEN (newpers.GEHALT < oldpers.GEHALT)
   SIGNAL SQLSTATE '70001'
                  SET MESSAGE_TEXT = 'Salary out of range');
```

INSTEAD OF - Trigger

- INSTEAD OF kann nur zur Definition der Semantik von Änderungen auf Sichten verwendet werden
  - Prinzip: Die Aktionen definieren die "inverse" Abbildungslogik der Sichtdefinition
  - Nur ein einziger Instead-Of-Trigger pro Sicht erlaubt
- Beispiel

```
CREATE VIEW PERSV (PNR, NAME, GEHALT, ALTER, ABTNAME)
AS (SELECT PNR, NAME, GEHALT, ALTER, ABT.ANAME
        FROM PERS, ABT
        WHERE PERS.ANR = ABT.ANR);
```

PERSV ist nicht änderbar

PERSV wird änderbar

- Update-Trigger ändert nur Werte in PERS

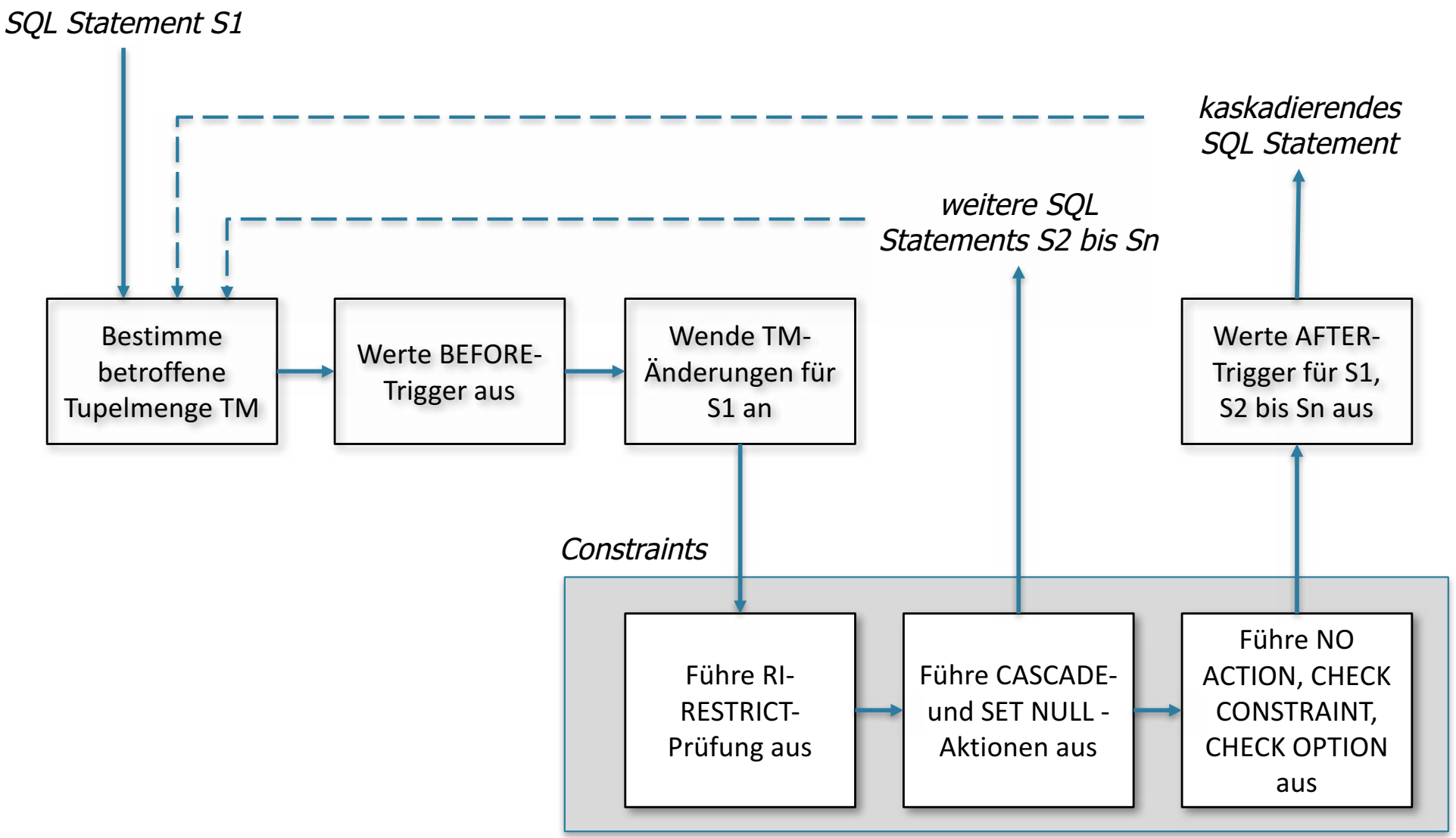
```
CREATE TRIGGER PERSV_UPDATE
INSTEAD OF UPDATE ON PERSV
REFERENCING NEW AS NP OLD AS OP
FOR EACH ROW
UPDATE PERS AS P
    SET (PNR, NAME, GEHALT, ALTER)
            = (NP.PNR, NP.NAME, NP.GEHALT, NP.ALTER),
        ANR = (SELECT ANR FROM ABT A
                WHERE A.ANAME = NP.ANAME)
WHERE OP.PNR = P.PNR
```

- Insert-Trigger folgt der gleichen Idee, fügt in PERS ein
- Delete-Trigger löscht nur aus PERS

Trigger – Überlegungen zur Auswertung

- Existiert das Problem der Terminierung und der Auswertungsreihenfolge?
  - Mehrere Trigger-Definitionen pro Tabelle erlaubt
  - Mehrere Trigger-Auslösungen pro Ereignis möglich
  - BEFORE vor AFTER, ältere Trigger zuerst
- Trigger-Aktionen können weitere Trigger auslösen
  - geschachtelte, mengenorientierte Auswertung
  - mögliche rekursive Aktivierung → Terminierung!
- Zusammenspiel mit Integritätskontrolle:
  Trigger-Auswertung muss mit Prüfung von Integritätsbedingungen und Ausführung von referentiellen Aktionen koordiniert werden.

Trigger-Auswertungsmodell



Trigger in Postgres

- Nicht komplett kompatibel zum SQL Standard
  - Z.B. kann man in PG keinen Alias OLD bzw. NEW einführen
  - und generell für STATEMENT Trigger nicht auf OLD TABLE bzw. NEW TABLE zugreifen
  - Trigger-Aktionen werden in Postgres immer mit speziellen Prozeduren implementiert
- SQL Standard sagt Trigger sollten in Reihenfolge der Erzeugung ausgeführt werden, Postgresql aber sortiert nach Namen
- CREATE CONSTRAINT TRIGGER ist eine Erweiterung in Postgres
  - In Zusammenhang mit create table verwendbar
  - Schließt den Kreis zu komplexeren check constraints

Weitere Details: siehe Postgresql-Manuals!

Zusammenfassung

- Modellinhärente Integritätsbedingungen in SQL
  - PRIMARY KEY, UNIQUE, NOT NULL, FOREIGN KEY
  - ermöglichen verfeinerte Abbildung von ER-Schemata
- Prüfzeitpunkt von Integritätsbedingungen
  - Verzögerung der Überprüfung (DEFERRED)
  - explizite Überprüfung mit SET CONSTRAINTS ... IMMEDIATE
- Referentielle Aktionen
  - ermöglichen automatisierte DB-Änderungen als Reaktion auf Änderung/Löschen von referenzierten Tupeln
  - Eindeutigkeit von referentiellen Aktionen anstreben!
- Beliebige statische Integritätsbedingungen
  - CHECK-Constraints, assoziiert mit Wertebereichen, Tabellen
  - Zusicherungen (Assertions) als unabhängige Integritätsbedingungen

Zusammenfassung (2)

- Trigger-Konzept
  - Unterstützung von aktiven Datenbanken mit Event-Condition-Action-Regeln (ECA)
  - Ermöglicht weitergehende Integritätskontrolle und –wartung durch automatische Ausführung von Triggeraktionen bei INSERT, UPDATE und DELETE
  - Erlaubt Definition von Abbildungssemantik bei Änderung von Sichten
- Wichtige Eigenschaften
  - Granularität, Aktivierungszeitpunkt, Ereignis, Aktivierungsbedingung, Aktion
  - Zugriff auf Übergangstabellen und –variablen in der Bedingung und den Aktionen
- Komplexes Auswertungsmodell
  - Wechselwirkungen mit Constraints
  - Kaskadierende Trigger