Integrität in Datenbanken

Unterschied Konsistenz - Integrität

- Konsistenz beschreibt die Korrektheit der DB-internen Speicherungsstrukturen, Zugriffspfade und sonstigen Verwaltungsinformation
- Integrität beschreibt die Korrektheit der Abbildung der Miniwelt in die in der DB gespeicherten Daten
  - Integrität kann verletzt sein, obwohl die Konsistenz der DB gewahrt bleibt
  - Ein DBMS kann nur die Konsistenz der Daten sichern
- Trotzdem spricht man in der DB-Welt von Integritätssicherung (z.B. referentielle Integrität)
  - Integritätsbedingungen (*Constraints*) spezifizieren akzeptable DB-Zustände
  - Änderungen werden nur zurückgewiesen, wenn sie entsprechend der Integritätsbedingungen als falsch erkannt werden

Klassifikation von semantischen Integritätsbedingungen

- Reichweite
  a) Attributbedingungen — GEB-JAHR ist numerisch, 4-stellig
  b) Bedingungen bezüglich Satzausprägung — ABT.GEHALTSSUMME < ABT.JAHRESETAT
  c) Satztypbedingungen — PNR ist eindeutig
  d) Satztypübergreifende Beding. — ABT.GEHALTSSUMME ist Summe aller Angestelltengehälter
- Zeitpunkt der Überprüfbarkeit
  - Unverzögert (*immediate*) sofort bei Änderungsoperation, z.B. a)
  - Verzögert (*deferred*) am Transaktionsende, z.B. d)
- Art der Überprüfbarkeit
  - Zustandsbedingungen
  - Zustandsübergangsbedingungen
  - Beispiele:
    - Übergang von FAM-STAND von ‘ledig‘ nach ‘geschieden‘ unzulässig
    - Gehalt darf nicht kleiner werden

Beschreibung von Integritätsbedingungen in SQL

- Constraints werden im Rahmen der Datendefinition angelegt
  - Deskriptiv: Definition der Bedingung (WAS?), aber nicht der Methode der Überwachung (WIE?)
  - Unterscheide spalten- vs. tabellenbezogene Constraints
- Eindeutigkeit von Attributwerten
  - UNIQUE: Werte in der Spalte/den Spalten sind eindeutig
- Verbot von Nullwerten
  - NOT NULL: Spalte muß einen Wert enthalten
- Schlüsselintegrität
  - PRIMARY KEY: bedeutet NOT NULL und UNIQUE
  - Definiert für eine Spalte oder für eine Menge von Spalten auf Tabellenebene
- Referentielle Integrität
  - FOREIGN KEY-REFERENCES Constraint
- Allgemeine benutzerdefinierte Integritätsbedingung
  - CHECK-Constraint (Regel für Spaltenwerte)

Beispiel: Referentielle Integrität

Nur Studenten, die in der Tabelle *Students* erfaßt sind, dürfen sich in Kurse einschreiben.

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
    PRIMARY KEY  (sid,cid),
    FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Einhaltung der referentiellen Integrität

- Betrachte Students und Enrolled; *sid* in Enrolled ist ein Fremdschlüssel, der Students referenziert
- Verhalten beim Einfügen
  Was ist zu tun, wenn ein Tupel in Enrolled eingefügt werden soll mit einer Student-ID, die nicht existiert ? (*Zurückweisen!*)
- Verhalten beim Löschen
  Was ist zu tun, wenn ein Tupel in Students gelöscht werden soll?
  - Lösche auch alle Tupel in Enrolled, die diesen Studenten referenzieren
  - Verbiete das Löschen eines Tupels in Students, das noch referenziert wird
  - Setze *sid* in Tupeln in Enrolled, die auf den Studenten verweisen, auf eine *Default sid* (Standardwert)
  - (In SQL auch möglich: Setze sid in Enrolled, die auf den Studenten verweisen, auf einen speziellen Wert *null,* bedeutet *`unbekannt* oder *`nicht anwendbar'*.)
- Ähnlich bei einer Änderung des Primärschlüssels in Students

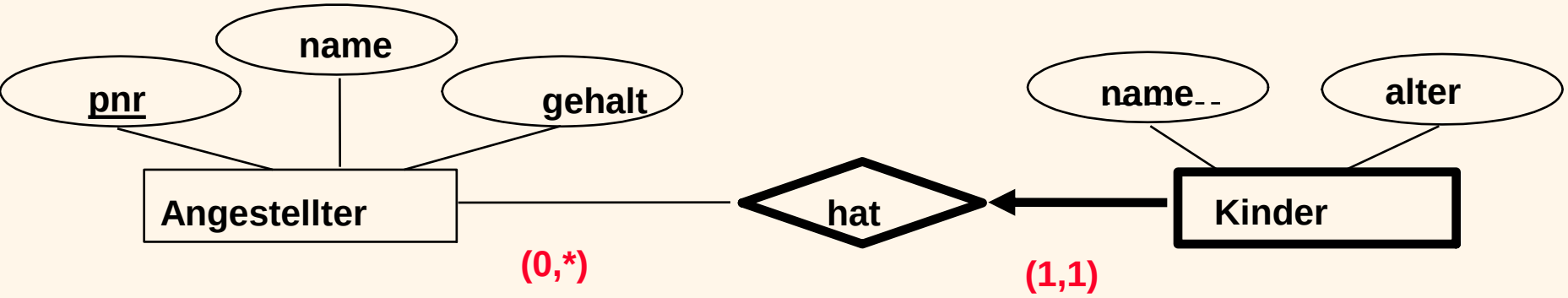
Referentielle Integrität in SQL/92

- SQL/92 unterstützt alle 4 Optionen für Deletes und Updates:
  - Default ist NO ACTION (*delete/update wird zurückgewiesen*)
  - CASCADE (propagiere die Löschung, d.h. entferne alle Tupel, die das gelöschte Tupel referenzieren)
  - SET NULL / SET DEFAULT (setze Fremdschlüsselwert des referenzierenden Tupels)

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2),
    PRIMARY KEY (sid,cid),
    FOREIGN KEY (sid)
      REFERENCES Students
            ON DELETE CASCADE
            ON UPDATE SET DEFAULT )
```

Rückblick: Schwache Entities

- Schwaches Entity (weak entity) kann eindeutig identifiziert werden nur über den Primärschlüssel einer anderen (Owner) Entity.
- Owner Entity und Weak Entity müssen in einer 1:n-Beziehung stehen (ein Owner, mehrere Weak Entities)



Jedes Entity aus Kinder **muß** an der Beziehung teilnehmen (*total Participation Constraint*)

Übersetzung schwacher Entity-Menge

- Schwache Entity-Menge und identifizierende Beziehung werden in eine einzige Tabelle übersetzt
  - Wenn das Owner-Entity (z.B. der Angestellte) gelöscht wird, müssen auch alle davon abhängigen schwachen Entities gelöscht werden (Existenzabhängigkeit).

```
CREATE TABLE Abhängig (
   name CHAR(20),
   alter INTEGER,
   pnr CHAR(11) NOT NULL,
   PRIMARY KEY (name, pnr),
   FOREIGN KEY (pnr) REFERENCES Angestellter,
      ON DELETE CASCADE)
```

Weiteres zu Constraints

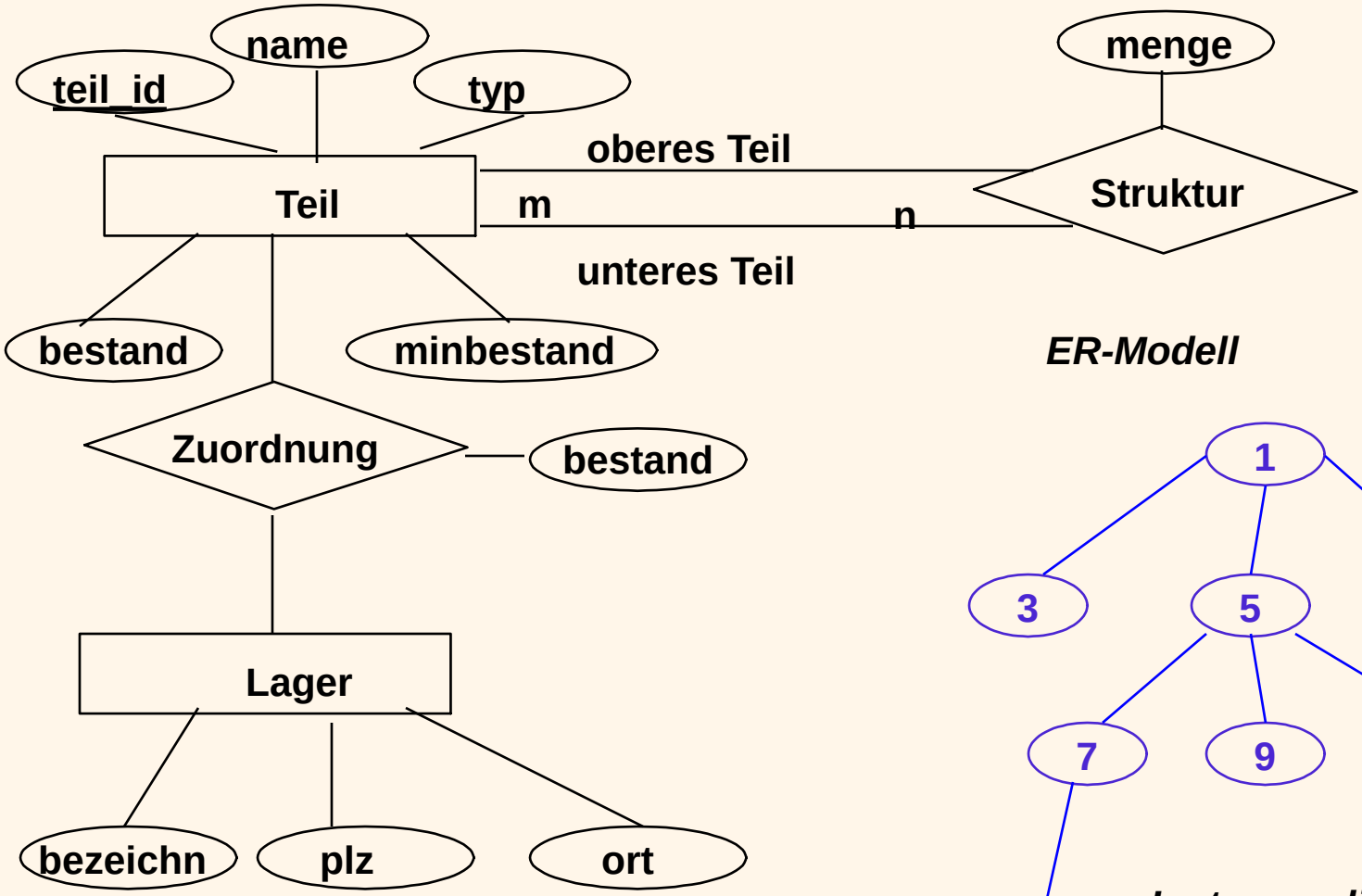
- Constraints können bei der Definition durch den Benutzer benannt werden
- Beispiel:

```
CREATE TABLE Teil (
   pk_teil_id   INTEGER
            CONSTRAINT pk_teil PRIMARY KEY (pk_teil_id),
   tname CHAR(30),
   ttyp CHAR(10)
)
```

- Constraints können
  - standardmässig unmittelbar (IMMEDIATE) ausgeführt werden oder
  - auch verzögert am Ende der aufrufenden Transaktion (DEFERRED), global einstellbar durch SET CONSTRAINTS DEFERRED

Beispiel für Constraint-Definitionen

Definition einer Stückliste



ER-Modell

Instanzendiagramm (Teile-Struktur)

Beispiel-Definition

```
CREATE TABLE  Teil (
   pk_teil_id      INTEGER PRIMARY KEY,
   tname           CHAR(30),
   ttyp            CHAR(10) NOT NULL,
   tbestand        NUMBER CHECK tbestand >= tminbestand
   tminbestand     NUMBER)

CREATE TABLE Struktur (
  fk_oteil_id      INTEGER
  CONSTRAINT fkrekursiv FOREIGN KEY (fk_oteil_id)
          REFERENCES Struktur (pk_uteil_id)
          ON DELETE CASCADE
  pk_uteil_id      INTEGER  PRIMARY KEY,
  CONSTRAINT fkteil2 FOREIGN KEY (pk_uteil_id)
          REFERENCES Teil (pk_teil_id),
  menge NUMBER,
  CONSTRAINT uqstruk UNIQUE (fk_oteil_id, pk_uteil_id)
);
```

Beispiel-Definition

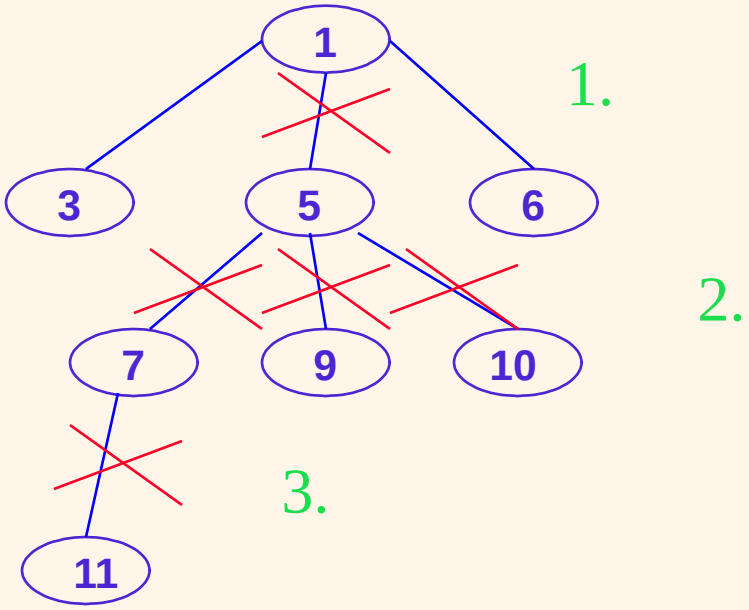
```
CREATE TABLE  Lager (
   pk_lager_id   INTEGER PRIMARY KEY,
   labez         CHAR(30),
   laplz         NUMBER,
   laort         CHAR(20) );

CREATE TABLE Lagerzuordnung (
  fk_lager_id     INTEGER
  CONSTRAINT fklagerzuordnung1 FOREIGN KEY (fk_lager_id)
         REFERENCES Lager (pk_uteil_id)
 fk_teil_id       INTEGER
  CONSTRAINT fklagerzuordnung2 FOREIGN KEY (fk_teil_id)
         REFERENCES Teil (pk_teil_id),
  bestand NUMBER,
  CONSTRAINT pk_lagerzuordnung PRIMARY KEY
         (fk_lager_id, fk_teil_id)
);
```

Beispiel-Szenario Löschen

- Wirkungsweise des Constraints fkrekursiv:

DELETE FROM Struktur
WHERE fk_uteil_id = ‘0005‘



Trigger-Konzept

- Spezielle Form von gespeicherten Prozeduren (stored procedures), die serverseitig ausgeführt werden
- Höhere Ausdrucksmächtigkeit als Constraints
- Aufruf erfolgt nicht explizit sondern bei Eintreten eines bestimmten Ereignisses (automatisches Starten von Operationen)
- Einsatzgebiete:
  - Sicherung von Datenintegrität
  - Zugriffskontrolle
  - Protokollierung
- Geschachtelte Ausführung von Triggern möglich
- Vorteil:
  DBMS-Funktionalität aus den Anwendungen heraus extrahiert und in der Datenbank zentralisiert (z.B. Integritätssicherung, Zugriffskontrolle)
- Nachteil:
  Kann unübersichtlich werden (z.B. Kaskaden)

Aufbau von Triggern

```
CREATE TRIGGER  trigger-name
BEFORE | AFTER
INSERT | UPDATE [OF column1, column2, ... ] | DELETE
        ON table-name
[ FOR EACH ROW ]
WHEN [ predicate ]
[ sql-block ]
```

- Trigger-Name
- Trigger-Zeitpunkt (BEFORE oder AFTER)
- Trigger-Ereignis (Einfügen, Löschen, Ändern in Tabellen), können mit OR verknüpft werden
- Trigger-Typ:
  - FOR EACH ROW: zeilenorientiert (anderenfalls befehlsorientiert)
- Trigger-Restriktion
  - Überprüfung einer Bedingung (Prädikat in WHEN-Klausel)
- Trigger-Rumpf
  - Aktion, die vom Trigger ausgeführt wird (falls Bedingung erfüllt ist)

Aufbau von Triggern (Forts.)

- Trigger-Name
  - Freie Namenswahl möglich
  - Namenskonventionen sinnvoll z.B.: Tabellenname/Triggerart/Ereignis
- Trigger-Zeitpunkt
  - before: Trigger feuert *genau einmal* vor Ausführung des Befehls
  - after: Trigger feuert *genau einmal* nach Ausführung des Befehls
  - Bei zeilenorientierten Triggern wird gefeuert vor oder nach jedem durch den Befehl betroffenen Tupel
- Trigger-Ereignis
  - Ereignisse können mittels OR verknüpft werden
  - Im Trigger-Rumpf kann Fallunterscheidung gemacht werden: z.B. if inserting then ... if deleting then ....
- Trigger-Typ
  - zeilenorientiert (FOR EACH ROW): Ausführung des Triggers pro Datensatz
  - befehlsorientiert: einmal pro auslösendem Befehl

Ausführung von Triggern und Constraints

1. Nach Update-, Insert- oder Delete-Befehl, Setzen aller Sperren und Kennungen
2. Ausführung befehlsorientierter Before-Trigger
3. Für jede angesprochene Zeile wird:
   a) der jeweilige zeilenorientierte Before-Trigger ausgeführt
   b) die Datenänderung durchgeführt und anhand der definierten Constraints überprüft soweie
   c) der jeweilige zeilenorientierte After-Trigger ausgeführt
4. Ausführung des befehlsorientierten After-Triggers
5. Ende der Abarbeitung durch nochmaliges Prüfen der Einhaltung der Constraints, Aufheben der Sperren und Kennungen

Im Fehlerfall erfolgt ein Transaktionsabbruch (*rollback*)

Trigger für Zugangskontrolle (Beispiel)

Befehlsorientierter Trigger regelt den Zugang zur Tabelle Teil.

Alle Benutzer außer dem SYSTEM-Benutzer dürfen nur zwischen 08:00 und 17:00 Uhr Datensätze anlegen, löschen oder ändern.

```
CREATE TRIGGER Teil_B
BEFORE DELETE OR INSERT OR UPDATE ON Teil
WHEN (USER != 'SYSTEM'
DECLARE
  Eingabe_unzulaessig EXCEPTION;
BEGIN
  IF TO_CHAR(SYSDATE, 'HH24:MI')
      NOT BETWEEN '08:00' AND '17:00' THEN
  RAISE Eingabe_unzulaessig
EXCEPTION
  WHEN Eingabe_unzulaessig THEN
  RAISE_APPLICATION_ERROR (-20001,
  'Nur zwischen 08:00 und 17:00 Uhr Daten eingeben');
END;
```

Protokoll-Trigger (Beispiel)

Protokolliere für jede Aktion auf der Tabelle Teil Benutzernamen, Statement, Datum in einer Protokolltabelle (in Oracle PL-SQL)

```
CREATE TABLE T_Protokoll (
  benutzer      VARCHAR2(30),
  statement     VARCHAR2(10),
  datum         DATE);

CREATE TRIGGER Teil_B
BEFORE DELETE OR INSERT OR UPDATE ON Teil
WHEN (USER != 'SYSTEM')
DECLARE
  statement varchar2(30);
BEGIN
  IF DELETING THEN   :statement := 'DELETE' END IF;
  IF INSERTING THEN :statement := 'INSERT' END IF;
  IF UPDATING THEN :statement := 'UPDATE' END IF;
  INSERT INTO T_Protokoll
     VALUES (USER, :statement, SYSDATE)
END;
```

Änderungs-Trigger (Beispiel)

After-Update-Trigger für den automatischen Update der Spalte *tbestand* in der Teil-Tabelle (Gesamtbestand eines Teils). Ziel: Jede Bestandsänderung in der Tabelle Lagerzuordnung muß korrekt in die Teil-Tabelle übernommen werden.

```
CREATE TRIGGER Lagerzuordnung_A_Update
AFTER UPDATE OF bestand ON Lagerzuordnung
BEGIN
  UPDATE Teil T
  SET T.tbestand =
          (SELECT SUM(bestand)
           FROM Lagerzuordnung lz
           WHERE T.pk_teil_id = lz.fk_teil_id
           GROUP BY lz.fk_teil_id);
END;
```

Sehr ineffiziente Lösung, da stets die gesamte Tabelle Teil aktualisiert wird, auch wenn nur wenige Tupel betroffen.

Änderungs-Trigger (Forts. Beispiel)

Bessere Lösung für gleiches Problem: Zeilenorientierter Trigger

```
CREATE TRIGGER Lagerzuordnung_AR_Update
AFTER UPDATE OF bestand ON Lagerzuordnung
FOR EACH ROW
DECLARE
  differenz Lagerzuordnung.bestand%TYPE   -- Variable gleichen Typs
BEGIN
  differenz := :NEW.bestand - :OLD.bestand;
  UPDATE Teil
  SET T.tbestand = tbestand + :differenz
  WHERE pk_teil_id = :NEW.fk_teil_id;
END;
```