

# 10. Integritätskontrolle und aktives Verhalten

## • Vision?

- Wir sind bald in der Lage, „alle Informationen“ aufzuheben, d. h., alles kann gespeichert, nichts muss weggeworfen werden<sup>1</sup>
- Die „typischen“ Informationen werden nur noch von Rechnern aufbewahrt, auf „**Richtigkeit**“ überprüft, gesucht und aufbereitet; wir kennen weder die Daten, noch ihren Aufbewahrungsort und die genauen Ableitungsverfahren

## • Semantische Integritätskontrolle

- bisher in SQL schwach ausgebildet (z. B. NOT NULL, UNIQUE)
- Relationale Invarianten erst in SQL2 verbindlich
- benutzerdefinierte Integritätsbedingungen (*assertions*)
- Erweiterungen in SQL:1999 (Trigger)

## • Regelverarbeitung in DBS

- Was heißt aktives Verhalten?
- neue Konzepte: Trigger, Produktionsregeln, Alerter

## • Trigger-Konzept von SQL

- Trigger-Granulate
- Trigger-Einsatz

## • Definition von ECA-Regeln

- Spezifikation von Ereignissen
- Anwendung von ECA-Regeln

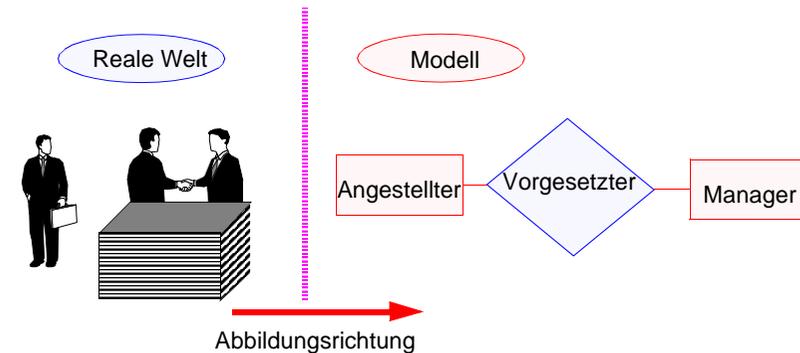
## • Regelausführung

- tupel- und mengenorientierte Ausführung
- ECA-Kopplungsmodi

1. <http://www.lesk.com/mlesk/ksg97/ksg.html>

# Integritätskontrolle

## • Abbildung der Miniwelt



## • Unterschied Konsistenz – Integrität

- Konsistenz beschreibt die Korrektheit der DB-internen Speicherungsstrukturen, Zugriffspfade und sonstigen Verwaltungsinformation.
- *Constraints* (Wertebereiche, Check-Klauseln usw.) sind Sprachkonzepte, die eine Überprüfung der Konsistenz durch das DBS gestatten.
- Integrität beschreibt die Korrektheit der Abbildung der Miniwelt in die in der DB gespeicherten Daten.

➔ Die Integrität kann verletzt sein, obwohl die Konsistenz der DB gewahrt bleibt.

➔ Ein DBS kann nur die Konsistenz der Daten sichern!

## • Trotzdem spricht man in der DB-Welt von Integritätskontrolle (z. B. Referentielle Integrität, nicht Referentielle Konsistenz).

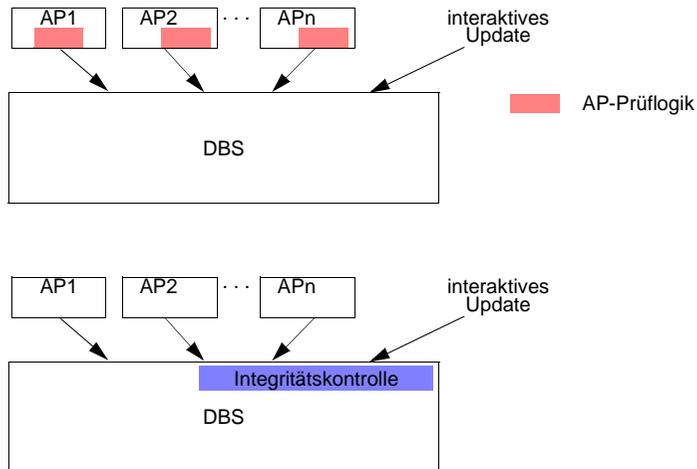
- Integritätsbedingungen (*Constraints*) spezifizieren akzeptable DB-Zustände (und nicht aktuelle Zustände der Miniwelt).
- Änderungen werden nur zurückgewiesen, wenn sie entsprechend der Integritätsbedingungen als falsch erkannt werden.

## Integritätskontrolle (2)

### • ZIEL<sup>2</sup>

- Nur DB-Änderungen zulassen, die allen definierten *Constraints* entsprechen (offensichtlich 'falsche' Änderungen zurückweisen!)
- Möglichst hohe Übereinstimmung von DB-Inhalt und Miniwelt (Datenqualität)

### • Integritätskontrolle durch Anwendungen oder durch DBS?



➔ *Integritätsbedingungen der Miniwelt sind explizit bekannt zu machen, um automatische Überwachung zu ermöglichen.*

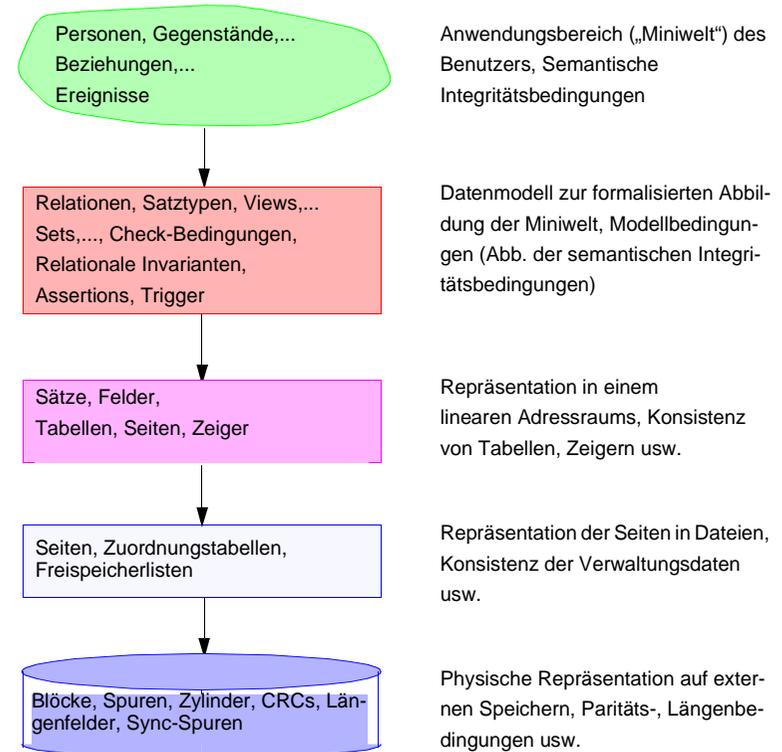
### • DBS-basierte Integritätskontrolle

- größere Sicherheit
- vereinfachte Anwendungserstellung
- Unterstützung von interaktiven sowie programmierten DB-Änderungen
- leichtere Änderbarkeit von Integritätsbedingungen
- ggf. Leistungsvorteile

2. „Golden Rule“ nach C. J. Date: No update operation must ever be allowed to leave any relation or view (relvar) in a state that violates its own predicate. Likewise no update transaction must ever be allowed to leave the database in a state that violates its own predicate.

## Integritätskontrolle (3)

### • Ebenen der Abbildungshierarchie



### • Physische Konsistenz der DB ist Voraussetzung für logische Konsistenz

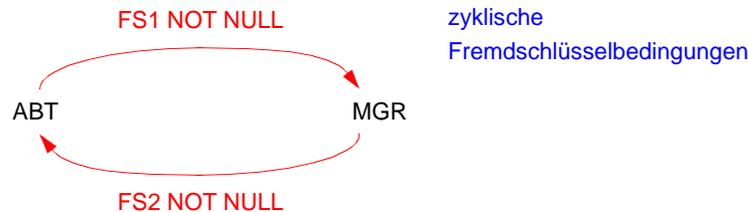
- Gerätekonsistenz
- Dateikonsistenz
- Speicherkonsistenz (Aktionskonsistenz) (Speicherungsstrukturen/Zugriffspfade/Zeiger sind konsistent)

### • Logische Konsistenz (TA-Konsistenz)

- modellinhärente Bedingungen (z. B. Relationale Invarianten)
- benutzerdefinierte Bedingungen aus der Miniwelt

## Arten von Integritätsbedingungen

- **Modellinhärente vs. sonstige (modellunabhängige) Integritätsbedingungen**
  - Modellinhärente Bedingungen des RM: 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)
- ➔ **geringere Reichweite = einfachere Überprüfung**
- **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 IB**: temporale IBs für längerfristige Abläufe (z.B. Gehalt darf innerhalb von 3 Jahren nicht um mehr als 25% steigen).
- **Zeitpunkt der Überprüfbarkeit: unverzögert vs. verzögert**

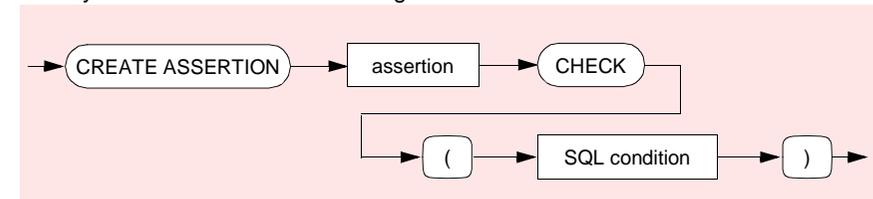


- **Verzögerte Bedingungen** lassen sich nur durch eine Folge von Änderungen erfüllen (typisch: mehrere Tupel, mehrere Relationen) und
- benötigen Transaktionsschutz (als zusammengehörige Änderungssequenzen)

10 - 5

## Integritätsbedingungen in SQL92

- **Primary Key- und Foreign Key-Klauseln**
  - referentielle Integrität: **deklarative Spezifikation** unterschiedlicher Reaktionsmöglichkeiten für Wegfall (Löschung, Änderung) eines referenzierten Satzes bzw. Primärschlüssels (CASCADE, ...)
- **Festlegung von Wertebereichen für Attribute durch Angabe eines Datentyps bzw. Domains**
  - optional: Angabe von Default-Werten, Eindeutigkeit (UNIQUE), Nullwerteausschluss (NOT NULL)
  - allgemeine Wertebereichsbeschränkungen über CHECK-Klausel
- **Spezifikation allgemeiner, z.B. tabellenübergreifender Bedingungen**
  - beziehen sich typischerweise auf mehrere Relationen
  - lassen sich als eigenständige DB-Objekte definieren
  - erlauben die Verschiebung ihres Überprüfungszeitpunktes
  - Syntax der Assertion-Anweisung



- **Direkte (IMMEDIATE) oder verzögerte (DEFERRED) Überwachung**

```
CREATE TABLE Pers
(Pnr INT PRIMARY KEY,
Gehalt INT CHECK (VALUE<500000),
Anr INT NOT NULL
FOREIGN KEY REFERENCES Abt
ON DELETE CASCADE
...);
```

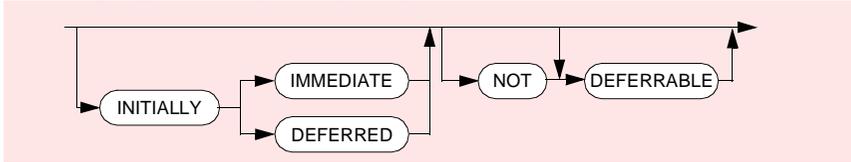
```
CREATE ASSERTION A1
CHECK (NOT EXISTS
(SELECT * FROM Abt A
WHERE A.Anzahl_Angest <>
(SELECT COUNT(*) FROM Pers
WHERE P.Anr = A.Anr)))
INITIALLY DEFERRED;
```

- ➔ **Bei welchen Operationen und wann muss überprüft werden?**

10 - 6

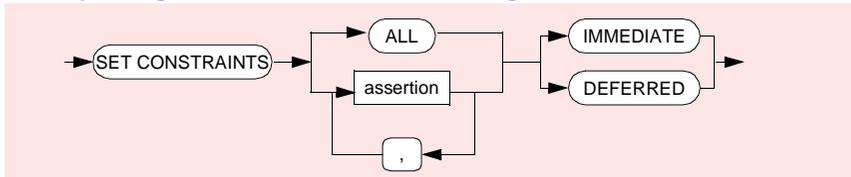
## Integritätsbedingungen in SQL92 (2)

### Festlegung des Überprüfungszeitpunktes:



- IMMEDIATE: am Ende der Änderungsoperation (Default)
- DEFERRED: am Transaktionsende (COMMIT)

### Überprüfung kann durch Constraint-Modus gesteuert werden



- Zuordnung gilt für die aktuelle Transaktion
- Bei benannten Constraints ist eine selektive Steuerung der Überprüfung möglich; so können gezielt Zeitpunkte vor COMMIT ausgewählt werden.

### Beispiel

| IB  | Zustand bei BOT       | t1: SET IB1 | t2: SET IB2 |
|-----|-----------------------|-------------|-------------|
| IB1 | IMMEDIATE, DEFERRABLE |             |             |
| IB2 | DEFERRED, DEFERRABLE  |             |             |

TA |-----|

## Beispiel-DB

| Abt | Anr | Aname    | Ort | Anzahl_Angest |
|-----|-----|----------|-----|---------------|
|     | K51 | PLANUNG  | KL  | 1             |
|     | K53 | EINKAUF  | F   | 1             |
|     | K55 | VERTRIEB | F   | 2             |

| Pers | Pnr | Name   | Alter | Gehalt | Anr | Mnr |
|------|-----|--------|-------|--------|-----|-----|
|      | 406 | COY    | 47    | 50 000 | K55 | 123 |
|      | 123 | MÜLLER | 32    | 43 500 | K51 | -   |
|      | 829 | SCHMID | 36    | 45 200 | K53 | 777 |
|      | 574 | ABEL   | 28    | 30 000 | K55 | 123 |

- Gehaltssumme an Abt anhängen
- Gehaltssumme mit Werten füllen
- Einfügen eines neuen Angestellten

Wann wird Constraint A2 überprüft?

```
CREATE ASSERTION A2
CHECK (NOT EXISTS
(SELECT * FROM Abt A
WHERE A.Geh_Summe <>
(SELECT SUM (P.Gehalt) FROM Pers P
WHERE P.Anr = A.Anr)))
INITIALLY DEFERRED;
```

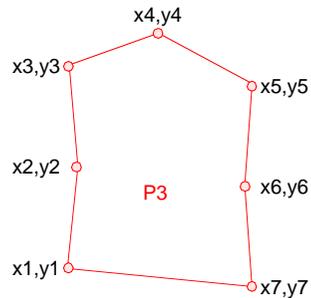
## Beispiel-DB (2)

- **Integritätsbedingungen auf einer Relation**

```
CREATE TABLE Polygon
(Id CHAR(5) NOT NULL,
 PktNr INTEGER NOT NULL,
 X DECIMAL NOT NULL,
 Y DECIMAL NOT NULL,
 PRIMARY KEY (Id, PktNr));
```

- Mehrere Polygone (Id) sind in Relation Polygon enthalten
- Position jedes Punktes (PktNr) in „Liste“ der Polygon-Punkte erforderlich!

Polygon P3 und seine relationale Darstellung



| Polygon |       |    |    |
|---------|-------|----|----|
| Id      | PktNr | X  | Y  |
| ...     |       |    |    |
| P3      | 1     | x1 | y1 |
| P3      | 2     | x2 | y2 |
| P3      | 3     | x3 | y3 |
| P3      | 4     | x4 | y4 |
| P3      | 5     | x5 | y5 |
| P3      | 6     | x6 | y6 |
| P3      | 7     | x7 | y7 |
| ...     |       |    |    |

- **Wie lassen sich folgende Integritätsbedingungen formulieren?**

- Folgeänderungen bei „Füge neuen Eckpunkt von P3 zwischen (x2,y2) und (x3,y3) ein!“
- Die Linien eines Polygons sollen sich nicht kreuzen!
- Der Umfang U eines Polygons darf nicht größer als Konstante c sein!

$$U = \sum_{i=2}^n \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} + \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2}$$

➔ **Deskriptive Formulierungen sind hier sehr schwierig!**

## Regelverarbeitung in DBS

- **Die Überprüfung von Integritätsbedingungen** und die Durchführung von Standardaufgaben der Datenverwaltung (z. B. Redundanz-Nachführung) **durch das Anwendungsprogramm** sind ineffizient.

- **Außerdem:**

Verletzung einer Integritätsbedingung impliziert Zurücksetzen der DB-Operation oder beim Zeitpunkt COMMIT (Option DEFERRED) gar der ganzen Transaktion.

- **Idee**

Explizite und möglichst deskriptive Beschreibung von Sachverhalten, Aktionen usw. durch **Regeln**, die eine DBS-kontrollierte Reaktion erlauben.

➔ Welche Regeln sind **bereits in SQL** eingebaut?

- **Anwendungsunabhängige Spezifikation und Handhabung von Regeln<sup>3</sup>**

- Vereinfachung der Anwendungsentwicklung
- einfachere Wartbarkeit
- Wiederbenutzung von Code
- garantierte Auslösung von „Geschäftsregeln“ (business rules)

➔ **Widerspruchsfreiheit und Vollständigkeit von Regelmengen sind schwierig zu überprüfen.**

(Bei prozeduralen Regelanteilen ist das unmöglich!)

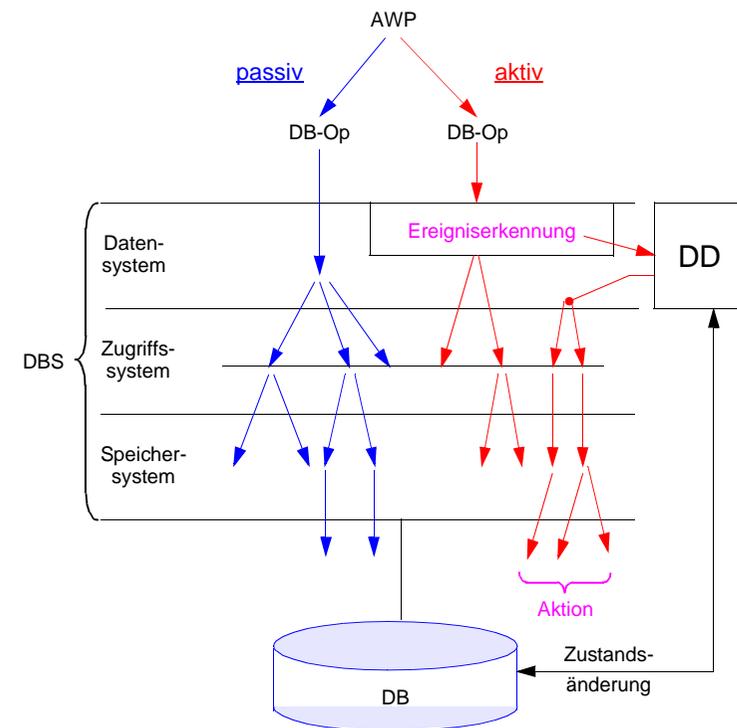
3. Solche Regelmengen gestatten eine deklarative Beschreibung von Situationen/Ereignissen und den zugehörigen Reaktionen, ohne dabei die Programmabläufe, in denen sie auftreten können, vorausplanen und spezifizieren zu müssen. Die Erkennung solcher Situationen/Ereignisse und die prozedurale Umsetzung der spezifizierten Reaktion wird dabei dem DBS überlassen. Weiterhin gestatten sie eine leichtere Erweiterbarkeit, was Hinzufügen, Löschen und Austauschen von Regeln sehr einfach gestaltet. Allerdings können Abhängigkeiten zwischen Regeln auftreten, wenn sie auf gemeinsame Daten Bezug nehmen und dabei Änderungen vollzogen werden. Das wird immer dann zu einem Problem bei der Regelausführung, wenn mehrere Regeln gleichzeitig ausgelöst und diese parallel bearbeitet werden sollen.

## Regelverarbeitung in DBS (2)

- **Wo und wie sollte die Realisierung erfolgen?**
    - in jeder Anwendung: schlechtes Software-Engineering
    - als Zusatzmechanismus im DBS mit zyklischer Abfrage durch einen Dämon (*polling daemon*): zu häufig oder zu selten
    - ➔ Entwurf eines **einheitlichen Mechanismus und integrierte Bereitstellung** in einem DBS
  - **Rolle des DD (Data Dictionary)**
    - DD enthält alle dem DBS bekannten Regeln
      - **100%-Ansatz:** Integrität wird ausschließlich durch Regeln im DD beschrieben
      - verbindlich für alle Benutzer, auch im verteilten Fall
    - einmalige Realisierung, zentrale Verwaltung der Regeln
    - effiziente Speicherung und Bereitstellung von Regeln: Wie sucht (indexiert) man Regeln?
  - **Wo soll Regelverarbeitung eingesetzt werden?**
    - Integritätssicherung
    - Kontrolle abgeleiteter Daten (Wartung von Redundanzen)
    - ...
- ➔ **Was heißt dabei aktives Verhalten?**
- ➔ Wie lässt sich folgendes Ziel erreichen:  
**Verallgemeinerung von Spezifikation und Reaktion durch Trigger, Regeln, Alerter**

## Was heißt „aktives Verhalten“?

- **Ausführung von DB-Operationen**
  - Wirkung der DB-Operationen in jeder DBS-Schicht festgelegt
  - expliziter Aufruf der DB-Operation erforderlich
  - DBS ist letztendlich „passiv“!
- **Wo beginnt aktives DBS-Verhalten?**
  - Erkennung von Ereignissen oder Zustandsänderungen
  - „selbständige“ Durchführung von Operationen



## Trigger, Regeln und Alerter

- **Integritätsbedingungen beschreiben, was innerhalb der DB gültig und zulässig ist.**
- **Neue Idee:**  
**Spezifikation und Durchführung von Reaktionen**  
auf bestimmte Situationen oder Ereignisse in der DB<sup>4</sup>
  - ➔ Oft synonyme Nutzung der Begriffe Produktionsregel, Regel, Aktive Regel, Trigger, Alerter
- **Neue Anforderung:**  
Wirkungsweise kann nicht durch ein **statisches Prädikat** spezifiziert werden, sondern als „**Zusammenhangsregel**“
  - ➔ Wenn <Bedingung verletzt>, dann <Führe Korrekturmaßnahmen durch>
- Zusammenhang durch **kausale**, **logische** oder „**beliebige**“ Verknüpfung
  - ➔ Wenn eine Abteilung auf weniger als 5 Angestellte verkleinert wird, dann kürze ihr Budget um 25%
  - ➔ Wenn durchschnittlicher Aktienkurs im Jahr > Limit, dann zahle Bonus an Mgr
  - ➔ Wenn Pers.Gehalt um mehr als 10% erhöht wird, benachrichtige Top-Level-Manager
- **Wer ist für die Triggerausführung verantwortlich?**
  - Kopplung mit Transaktionen
  - Ereignisse (events) unabhängig von Transaktionen?

4. Je mehr Semantik des modellierten Systems explizit repräsentiert ist, umso häufiger kann das DBS „aktiv“ werden!

## Trigger, Regeln und Alerter (2)

- **Trigger**
  - „triggernde“ Operation wird als *Event* bezeichnet
  - Reaktion auf Event besteht aus Folge von DB-Operationen
    - AFTER <Event> ON <Table>
    - DO <DML-Operation>, ...
  - In relationalen Systemen sind i. Allg. nur Modifikationsoperationen als Events vorgesehen.
  - In objektorientierten Systemen kann i. Allg. jeder Aufruf einer Methode ein Event sein.
    - ➔ DBS muss das Auftreten der Events erkennen und die zugehörigen Operationen durchführen.
- **Ausführung von Triggern**
  - mehrere Trigger für ein Event
  - rekursive Auflösung von Triggern
    - ➔ zentrale Probleme:  
**Terminierung und Reihenfolge der Regelausführung**

## Terminierung bei Triggern

- Kontrolle der Regelausführung**

Datenänderungen triggern Regeln, die Daten ändern, die Regeln triggern, die Daten ändern, ...

- hier: rekursive Auflösung von Triggern

**Beispiel zur Terminierung:**

```

Create Trigger V
  After Update (Gehalt) ON Pers A
    Update Pers P
    Set P.Gehalt = 1.02 * P.Gehalt
    Where P.Pnr = A.Mnr
    
```

}

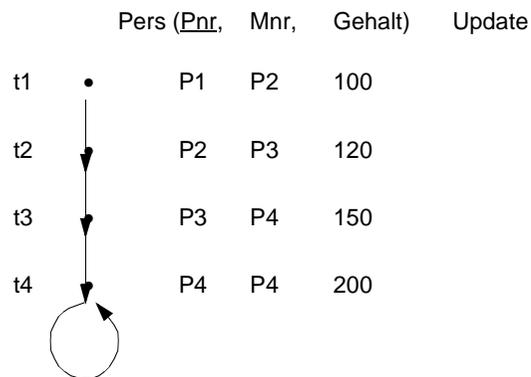
Event

}

Action

Op1: Update Pers P  
 Set P.Gehalt = 1.05 \* P.Gehalt  
 Where P.Pnr = P1

**Ablauf**



## Auswertungsreihenfolge bei Triggern

- Kontrolle der Regelausführung**

- hier: mehrere Trigger für ein Event

**Erweiterung:**

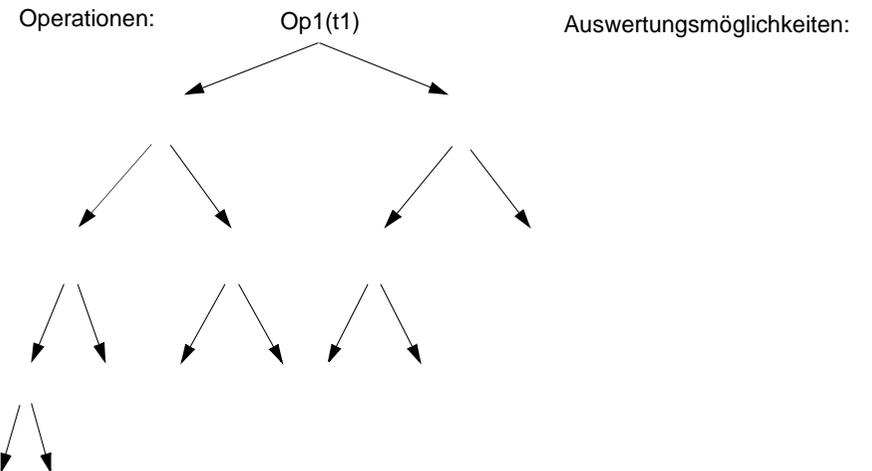
```

Create Trigger NHV
  After Update (Gehalt) ON Pers A
  Update Pers P
  Set P.Gehalt = 1.01 * P.Gehalt
  Where P.Pnr =
    ( Select X.Mnr
      From Pers X
      Where X.Pnr = A.Mnr)
    
```

➔ Es existieren V und NHV!

**Ablauf**

|     | Pers ( <u>Pnr</u> , Mnr, Gehalt) |
|-----|----------------------------------|
| t1: | P1 P2 100                        |
| t2: | P2 P3 120                        |
| t3: | P3 P4 150                        |
| t4: | P4 --- 200                       |



## Trigger, Regeln und Alerter (3)

### • Regeln

- Grundlagen der Regelverarbeitung wurde im Bereich der XPS entwickelt
  - Verarbeitungswissen liegt in der Form von WENN-DANN-Regeln vor
    - explizite Ableitung der in der DB implizit enthaltenen Information mit Hilfe der Regeln
    - Aktualisierung der abgeleiteten Daten, falls sich die Basisdaten ändern
  - **Konzeptioneller Unterschied zu Triggern**
    - Aktivität (DANN ...) wird nur indirekt von Events ausgelöst
    - Es wird logisch die Situation spezifiziert (WENN), aus der sich weitere Dinge ableiten lassen
- ➔ **Situation ist ein DB-Zustand.**  
Wann/Wie wird Situation erkannt?

### • Alerter

- Es werden als Reaktionen nicht nur Manipulationen der DB erlaubt, es ist vielmehr auch ein **direkter Anwendungsbezug** möglich.
  - Realisierung eines aktiven Verhaltens des DBS unter Einbezug der Anwendung/des Benutzers
  - Beispiele: Auslösung von Nachbestellungen, Benachrichtigung von Personen, interaktiver Dialog mit dem Benutzer
- ➔ **Besonderes Problem ist die Integration von Anwendungsprogramm(-teil)en in das DBS.**  
**Semantik der Reaktion ist dem DBS nicht mehr bekannt!**

## Spezifikation von Triggern

### • Idee:

Automatische Korrektur des DB-Zustandes:  
Starten von Folgeänderungen zur Wahrung der DB-Integrität

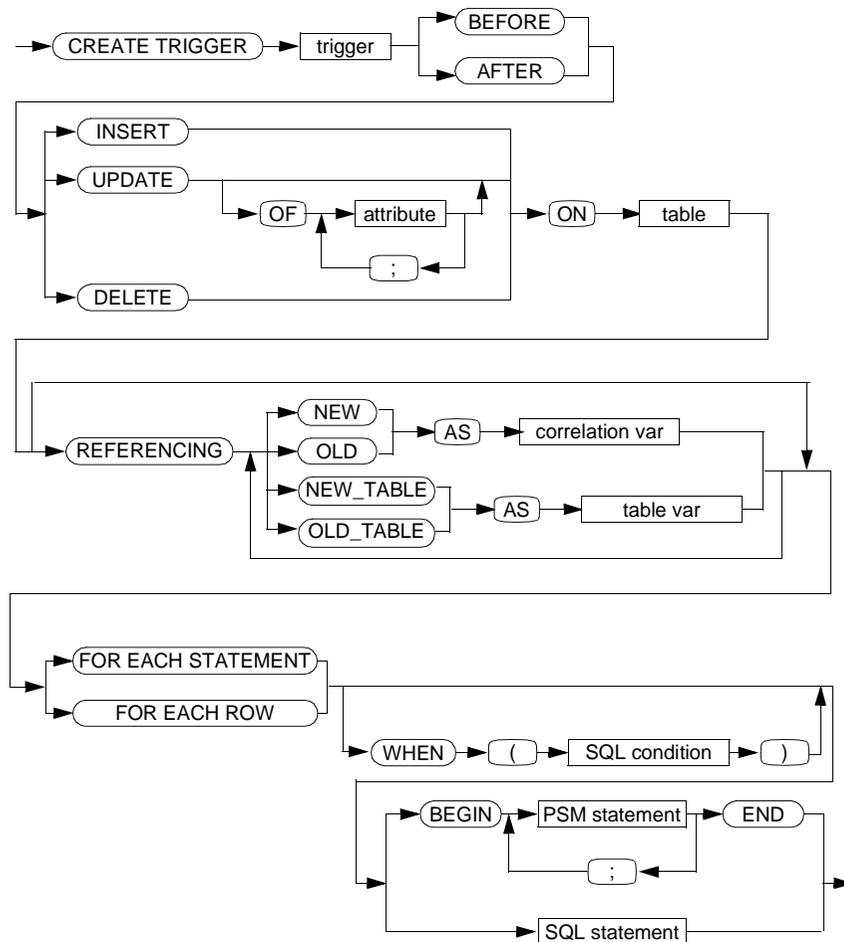
➔ **Schwierig: Trigger werden schon seit ~1985 in relationalen DBS eingesetzt.**  
Ihre Standardisierung wurde jedoch erst in SQL:1999 vorgenommen.

### • Trigger-Konzept

- Wann soll ein Trigger ausgelöst werden?
  - Zeitpunkte: BEFORE / AFTER
  - auslösende Operation: INSERT / DELETE / UPDATE
- Wie spezifiziert man (bei Übergangsbedingungen) Aktionen?
  - Bezug auf verschiedene DB-Zustände erforderlich
  - OLD/NEW erlaubt Referenz von alten/neuen Werten
- Ist die Trigger-Ausführung vom DB-Zustand abhängig?  
(WHEN-Bedingung optional)
- Was soll wie verändert werden?
  - pro Tupel oder pro DB-Operation (Trigger-Granulat)
  - mit einer SQL-Anweisung oder mit einer Prozedur aus PSM-Anweisungen (persistent stored module, stored procedure)
- Existiert das Problem der Terminierung und der Auswertungsreihenfolge?
  - mehrere Trigger-Definitionen pro Relation (Tabelle) sowie
  - mehrere Trigger-Auslösungen pro Ereignis möglich

## Spezifikation von Triggern (2)

### • Trigger-Syntax in SQL:1999



10 - 19

## Spezifikation von Triggern (3)

### • Übergangstabellen und -variablen

- Sie vermerken Einfügungen (bei INSERT), Löschungen (bei DELETE) und die alten und neuen Zustände (bei UPDATE).
- Übergangstabellen (transition tables) enthalten mengenorientierte Änderungen, während Übergangsvariablen (transition variables) die zeilenweisen Änderungen aufnehmen.

### • Trigger-Granulat

- FOR EACH STATEMENT: mengenorientiertes Verarbeitungsmodell
- FOR EACH ROW: zeilenorientiertes Verarbeitungsmodell
- TRA: Trigger-Aktion



### • Einsatzbeispiel

- Die Gehaltssumme in Abt soll bei Änderungen in Pers, die „Gehälter“ betreffen, automatisch aktualisiert werden.
- Es sind Trigger für INSERT/DELETE/UPDATE erforderlich. Sie werden bei Auftreten der spezifizierten Änderungsoperationen sofort ausgeführt.

10 - 20

## Trigger-Einsatz

| Abt | Anr | Aname    | Ort            | Geh_Summe |
|-----|-----|----------|----------------|-----------|
|     | K51 | PLANUNG  | KAISERSLAUTERN | 43500     |
|     | K53 | EINKAUF  | FRANKFURT      | 45200     |
|     | K55 | VERTRIEB | FRANKFURT      | 80000     |

| Pers | Pnr | Name   | Alter | Gehalt | Anr | Mnr |
|------|-----|--------|-------|--------|-----|-----|
|      | 406 | COY    | 47    | 50 000 | K55 | 123 |
|      | 123 | MÜLLER | 32    | 43 500 | K51 | -   |
|      | 829 | SCHMID | 36    | 45 200 | K53 | 777 |
|      | 574 | ABEL   | 28    | 30 000 | K55 | 123 |

## Trigger-Einsatz (2)

| Abt | Anr | Aname    | Ort            | Geh_Summe |
|-----|-----|----------|----------------|-----------|
|     | K51 | PLANUNG  | KAISERSLAUTERN | 43500     |
|     | K53 | EINKAUF  | FRANKFURT      | 45200     |
|     | K55 | VERTRIEB | FRANKFURT      | 80000     |

| Pers | Pnr | Name   | Alter | Gehalt | Anr | Mnr |
|------|-----|--------|-------|--------|-----|-----|
|      | 406 | COY    | 47    | 50 000 | K55 | 123 |
|      | 123 | MÜLLER | 32    | 43 500 | K51 | -   |
|      | 829 | SCHMID | 36    | 45 200 | K53 | 777 |
|      | 574 | ABEL   | 28    | 30 000 | K55 | 123 |

- Gehaltserhöhung von K55-Angestellten um 10%:

- Einfügen eines Angestellten:

Einsatz von Übergangsvariablen: NEW : NP.Anr . . . NP.Gehalt

Einsatz von Übergangsvariablen: OLD : OP.Anr . . . OP.Gehalt

NEW : NP.Anr . . . NP.Gehalt

- Wie wird Trigger T1 ausgeführt?

```
CREATE TRIGGER T1
AFTER INSERT ON Pers          (* Ereignis *)
REFERENCING NEW AS NP
FOR EACH ROW
  UPDATE Abt A                (* Aktion *)
  SET A.Geh_Summe =
    A.Geh_Summe + NP.Gehalt
  WHERE A.Anr = NP.Anr;
```

- Wie wird Trigger T2 ausgeführt?

```
CREATE TRIGGER T2
AFTER UPDATE OF Gehalt ON Pers (* Ereignis *)
REFERENCING OLD AS OP NEW AS NP
FOR EACH ROW
  UPDATE Abt A                (* Aktion *)
  SET A.Geh_Summe =
    A.Geh_Summe + (NP.Gehalt - OP.Gehalt)
  WHERE A.Anr = NP.Anr;
```

### Trigger-Einsatz (3)

| Abt | Anr | Aname    | Ort            | Geh_Summe |
|-----|-----|----------|----------------|-----------|
|     | K51 | PLANUNG  | KAISERSLAUTERN | 43500     |
|     | K53 | EINKAUF  | FRANKFURT      | 45200     |
|     | K55 | VERTRIEB | FRANKFURT      | 80000     |

| Pers | Pnr | Name   | Alter | Gehalt | Anr | Mnr |
|------|-----|--------|-------|--------|-----|-----|
|      | 406 | COY    | 47    | 50 000 | K55 | 123 |
|      | 123 | MÜLLER | 32    | 43 500 | K51 | -   |
|      | 829 | SCHMID | 36    | 45 200 | K53 | 777 |
|      | 574 | ABEL   | 28    | 30 000 | K55 | 123 |

- **Gehaltserhöhung von K55-Angestellten um 10%:**

Einsatz von Übergangstabellen: OLD\_TABLE: OT.Anr . . . OT.Gehalt

NEW\_TABLE: NT.Anr . . . NT.Gehalt

- **Wie wird Trigger T3 ausgeführt?**

```

CREATE TRIGGER T3
AFTER UPDATE OF Gehalt ON Pers (* Ereignis *)
REFERENCING OLD_TABLE AS OT NEW_TABLE AS NT
FOR EACH STATEMENT
UPDATE Abt A (* Aktion *)
SET A.Geh_Summe = A.Geh_Summe +
(SELECT SUM (Gehalt) FROM NT WHERE Anr = A.Anr) -
(SELECT SUM (Gehalt) FROM OT WHERE Anr = A.Anr)
WHERE A.Anr IN (SELECT Anr FROM NT);
    
```

### Trigger-Einsatz (4)

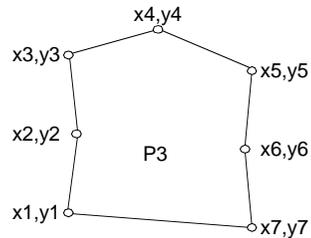
- **Gültige Kombinationen für Trigger-Granulate und Übergangstabellen und -variablen**

| Granulat   | Aktivierungszeit | Triggernde Operation | Übergangsvariablen erlaubt | Übergangstabellen erlaubt |
|------------|------------------|----------------------|----------------------------|---------------------------|
| ROW        | BEFORE           | INSERT               | NEW                        | NONE                      |
|            |                  | UPDATE               | OLD, NEW                   |                           |
|            |                  | DELETE               | OLD                        |                           |
|            | AFTER            | INSERT               | NEW                        | NEW_TABLE                 |
|            |                  | UPDATE               | OLD, NEW                   | OLD_TABLE, NEW_TABLE      |
|            |                  | DELETE               | OLD                        | OLD_TABLE                 |
| STATE-MENT | BEFORE           | INSERT               | NONE                       | NONE                      |
|            |                  | UPDATE               |                            |                           |
|            |                  | DELETE               |                            |                           |
|            | AFTER            | INSERT               | NONE                       | NEW_TABLE                 |
|            |                  | UPDATE               |                            | OLD_TABLE, NEW_TABLE      |
|            |                  | DELETE               |                            | OLD_TABLE                 |

## Trigger-Beispiel

- Umfang eines Polygons:  $U \leq c$

Polygon P3 und seine relationale Darstellung



| Polygon |       |    |    |
|---------|-------|----|----|
| Id      | PktNr | X  | Y  |
| ...     |       |    |    |
| P3      | 1     | x1 | y1 |
| P3      | 2     | x2 | y2 |
| P3      | 3     | x3 | y3 |
| P3      | 4     | x4 | y4 |
| P3      | 5     | x5 | y5 |
| P3      | 6     | x6 | y6 |
| P3      | 7     | x7 | y7 |
| ...     |       |    |    |

- Trigger für die Aktualisierung von Tupeln (PL/SQL von Oracle)

```

CREATE TRIGGER UmfangSQLDialektUpdatePolygon
AFTER UPDATE ON Polygon REFERENCING NEW AS NP
FOR EACH ROW
DECLARE
    PktNr INTEGER,
    X1, Y1, Xi, Yi, Xminus1, Yminus1 DECIMAL,
    Umfang := 0.0 REAL,
    CURSOR Stützpunkt IS
        SELECT PktNr, X, Y
        FROM Polygon
        WHERE Id = NP.Id
        ORDER BY PktNr;
BEGIN
    OPEN Stützpunkt;
    FETCH Stützpunkt INTO :PktNr, :X1, :Y1;
    Xminus1 := X1;
    Yminus1 := Y1;
    LOOP
        FETCH Stützpunkt INTO :PktNr, :Xi, :Yi;
        Umfang := Umfang + SQRT ((Xi-Xminus1)2+(Yi-Yminus1)2);
        Xminus1 := Xi;
        Yminus1 := Yi;
    END LOOP;
    CLOSE Stützpunkt;
    Umfang := Umfang + SQRT ((X1-Xminus1)2+(Y1-Yminus1)2);
    IF Umfang > c
    THEN RAISE ERROR (1000, 'Aktualisieren von Polygon: Umfang zu groß.');
```

```

ENDIF;
END;
```

10 - 25

## Trigger-Beispiel (2)

- Auch das ist ein Trigger!

| Trigger-Name | Beschreibung (Trigger-Einsatz in einem Workflow-System)  |
|--------------|--|
| ResultToS    | Wenn das Resultat in den Status „S“ (abgeschlossen) wechselt, dann ist der Vorgang abgeschlossen. In der Tabelle CurrentFlow wird der Status des Vorgangs auf „S“ (abgeschlossen) geändert und das Attribut FlowTimeEnd auf die aktuelle Zeitmarke gesetzt. Danach wird der gesamte Vorgang in die History-Tabellen kopiert und aus den Current-Tabellen gelöscht. |

```

CREATE TRIGGER ResultToS
AFTER UPDATE OF AcStatus ON CurrentAc
REFERENCING NEW AS n
FOR EACH ROW MODE DB2SQL
WHEN (n.AcChar = 'R' AND n.AcStatus = 'S')
BEGIN ATOMIC
    UPDATE CurrentFlow
    SET FlowStatus = 'S',
        FlowTimeEnd = CASE
            WHEN FlowTimeEnd IS NULL
            THEN CURRENT_TIMESTAMP
            ELSE FlowTimeEnd
        END
    WHERE FlowId = n.FlowId;
    INSERT INTO HistoryFlow SELECT * FROM CurrentFlow WHERE FlowId = n.FlowId;
    INSERT INTO HistoryAc SELECT * FROM CurrentAc WHERE FlowId = n.FlowId;
    INSERT INTO HistoryTr SELECT * FROM CurrentTr WHERE FlowId = n.FlowId;
    INSERT INTO HistoryTrAfterAc SELECT *
        FROM CurrentTrAfterAc
        WHERE FlowId = n.FlowId;
    INSERT INTO HistoryTrBeforeAc SELECT *
        FROM CurrentTrBeforeAc
        WHERE FlowId = n.FlowId;
    DELETE FROM CurrentTrBeforeAc WHERE FlowId = n.FlowId;
    DELETE FROM CurrentTrAfterAc WHERE FlowId = n.FlowId;
    DELETE FROM CurrentTr WHERE FlowId = n.FlowId;
    DELETE FROM CurrentAc WHERE FlowId = n.FlowId;
    DELETE FROM CurrentFlow WHERE FlowId = n.FlowId;
```

```

END;
```

10 - 26

## ECA-Regeln

- **In SQL existieren Gemeinsamkeiten zwischen den Konzepten**
  - Wertebereichsdefinition ist Spezialform allgemeiner Zusicherungen
  - Referentielle Integrität kann durch Trigger gewartet werden (referentielle Aktionen)
  - Erweiterungen für allgemeine Geschäftsregeln erforderlich
    - ➔ **Bereitstellung der gesamten Funktionalität (und mehr) durch einen vereinheitlichten Mechanismus**
- **Erweiterung der Struktur**
  - *Event*: Auslöser
  - *Condition*: Bedingungsteil
  - *Action*: Aktionsteil
    - ➔ **Bedingung stellt eine zusätzliche, auf der gesamten DB definierbare Voraussetzung für den Ablauf des Aktionsteils dar**
    - ➔ **Kontrollstrukturen (*stored procedures*) im Aktionsteil wünschenswert**
- **Fragen**
  - Welche *Events* werden unterstützt?
  - Wie komplex sind *Conditions*?
  - Wie komplex sind *Actions*?
  - Können *Conditions* und *Actions* sich auf *Events* beziehen?

## ECA-Regeln (2)

- **Übersicht über die Struktur von ECA-Regeln**
  - Vereinheitlichte Darstellung
  - Erweiterungen gegenüber SQL:1999
    - ➔ **ECA-Regeln haben komplexere Syntax**
- **Ereignisangabe (Event ,E')**
  - nach SQL:1999: before / after insert / update / delete bezogen auf eine Relation
  - erweitert: instead of / ..., select / ...
  - bezogen auf Transaktionszustand: on bot / commit / abort / ...
  - zeitgesteuert: at / during / repeat / ...
  - benutzerdefiniert: on event ...
- **Bedingungsangabe (Condition ,C')**
  - nach SQL:1999: Boole'scher Ausdruck (,search condition') über den Daten in der Datenbank
  - erweitert: Volle Mächtigkeit eines Auswahl ausdrucks (in SQL entsprechend dem ,select')
- **Aktionsangabe (Action ,A')**
  - nach SQL:1999: nur DML-Anweisungen und PSM-Anweisungen (stored procedures, user-defined routines (UDRs))
  - erweitert: auch DDL-Anweisungen, externe Funktions-/Prozeduraufrufe und Transaktionsanweisungen (z. B. ROLLBACK WORK)

## ECA-Regeln (3)

### • Die wesentlichen Syntaxelemente der ECA-Regeln

```
<eca-rule definition> ::=
CREATE ECA-RULE <rulename1> AS
{ {
  { BEFORE | AFTER }
  { INSERT | DELETE | {UPDATE | READ} [OF(<column1> [, ...]) ] }
} [, ... ]
ON <table1>
[ REFERENCING
  [ OLD AS <oldname1> ]
  [ NEW AS <newname1> ]
]
[ { FOR EACH STATEMENT | ROW OF {OLD | NEW} } ]

|
ON {BOT | COMMIT | ABORT [ <transaction name> ]}
|
ON EVENT <eventname1> [ <param1> [, ... ] ]
|
{ AT <datetime1> | DURING <datetime2> - <datetime3>
  REPEAT EACH <timeval1> }
}

[ CHECK [ CONDITION <condname1> ]
  [ { IMMEDIATE [ NOT ] DEFERRABLE | DEFERRED } ]
  [ { COUPLED [ NOT ] DECOUPABLE |
    { DEPENDENT | INDEPENDENT } DECOUPLED } ]
  IF <search condition> [THEN ] ]

[ DO [ ACTION <actionname1> ]
  [ { IMMEDIATE [ NOT ] DEFERRABLE | DEFERRED } ]
  [ { COUPLED [ NOT ] DECOUPABLE |
    { DEPENDENT | INDEPENDENT } DECOUPLED } ]
  [ REFERENCING COND AS <condname2> ]
  { <sql statement> [ , ... ] | RESTRICT }
]
END ECA-RULE [ <rulename1> ]
```

## ECA-Regeln (4)

### • Bemerkungen

- Es muss mindestens ein Event angegeben werden (relationen-, ereignis-, zeitbezogen oder benutzerdefiniert). Wird bei zeitbezogenen Events auch REPEAT EACH angegeben, so wird jedesmal nach Ablauf der Zeit „timeval<sub>1</sub>“ die ECA-Regel erneut ausgelöst, falls „datetime<sub>2</sub>“ noch nicht erreicht oder nicht angegeben ist.
- Der Bedingungsteil ist vollständig optional (CHECK...IF...).
- Der Aktionsteil (DO...END) muss angegeben werden. FOR EACH steuert dabei, ob die ECA-Regel für jedes Tupel einzeln (ROW) oder für alle betroffenen Tupel zusammen (STATEMENT) ausgelöst werden soll.
- Es wird auch der Einsatz von zusammengesetzten Events vorgeschlagen.

### • Umsetzung auf ECA-Regeln

- **Trigger** benötigen vor allem Event-Spezifikationen und DML-Operationen im Aktionsteil. Die Überprüfung einer Bedingung ist im allgemeinen nicht erforderlich.
- **Alerter** wurden für anwendungsnahe Aktionsmöglichkeiten eingeführt. Dafür werden allgemeine Prozeduraufrufe im Aktionsteil eingesetzt.
- **Regeln** spezifizieren logische „Voraussetzungen“ für Folgeaktionen:  
(IF <search condition> THEN...)

## ECA-Regeln (5)

### Kopplungsmodi von ECA-Regeln

- E-C-Kopplung  
zwischen Event und Auswertung der Bedingung
- C-A-Kopplung  
zwischen der Bedingung und der Aktion

### Kopplungsmodus zerfällt in zwei Teile

- **zeitlicher Bezug** (IMMEDIATE, DEFERRED)
- **transaktionaler Bezug**: beteiligte Elemente werden
  - in derselben TA (COUPLED) oder
  - in verschiedenen TAs ausgeführt (DECOUPLED)

### Verfeinerung der Kopplung<sup>5</sup>

- Veränderung des Kopplungsmodus zur Laufzeit:  
COUPLED [NOT] DECOUPABLE
- **DEPENDENT DECOUPLED**: abhängig gestartete TA darf nur dann zum Commit kommen, wenn dies auch für die TA gilt, aus der heraus sie gestartet wurde
- **INDEPENDENT DECOUPLED**: Commit-Abhängigkeit gilt hier nicht. Beispielsweise kann letztere auch dann einen Vorgang protokollieren (audit trail), wenn die auslösende TA mittels Abort abgebrochen wird.

## ECA-Regeln (6)

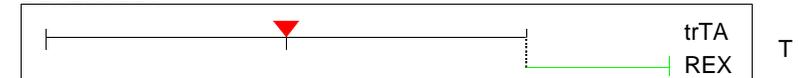
### Zeitlicher und transaktionaler Bezug der Kopplungs-Modi:

auslösende TA (trTA) und Regelausführung (REX)

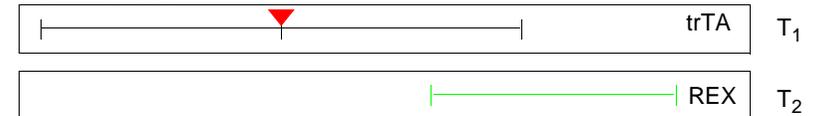
*IMMEDIATE:*



*DEFERRED:*

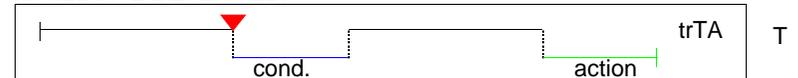


*DECOUPLED:*

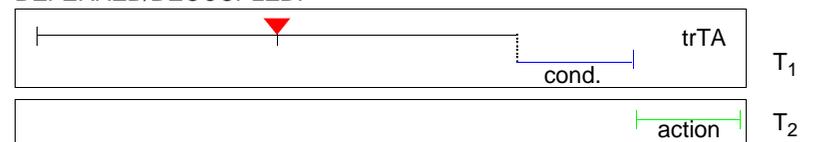


### Individuelle Kopplungs-Modi: E – C / C – A

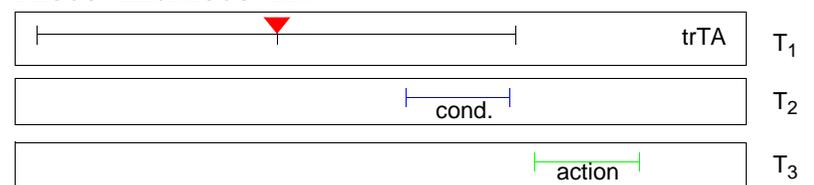
*IMMEDIATE/DEFERRED:*



*DEFERRED/DECOUPLED:*



*DECOUPLED/DECOUPLED:*



5. Dayal, U. et al: The HiPAC Project: Combining Active Databases and Timing Constraints, SIGMOD Record 17:1, 1988, pp. 51-70

## Spezifikation von Ereignissen

- **Ereignis ist grundsätzlich ein Zeitpunkt**
  - Es sind nur Zeitpunkte von Interesse, für die Regeln spezifiziert sind.
    - ➔ **Überwachungsintervall**
  - Das Eintreten eines Ereignisses wird vom System entdeckt; es wird der zuständigen Komponente signalisiert.
  - Realisierung hat dafür zu sorgen, dass zwischen Eintreten und Entdeckung nur eine tolerierbare Zeitspanne vergeht.
    - ➔ **besondere Anforderungen bei Realzeitsystemen**
- **Ereignisklasse und Ereignisinstanz**
  - In Regeln werden stets Ereignisklassen (kurz: Ereignis) spezifiziert.
  - Ereignisinstanz: aktuelles Eintreten eines Ereignisses einer Ereignisklasse
- **Parametrisierung von Ereignisklassen**
  - ermöglicht Weitergabe von Informationen, die beim Eintreten des Ereignisses zur Verfügung stehen (z. B. Zeitpunkt der Entdeckung), an die Bedingung und/oder Aktion der Regel
  - ist wichtig zur Modellierung zusammengesetzter Ereignisse
- **Spezifikation von Ereignissen**
  - Ereignis kann in mehreren Regeldefinitionen vorkommen
    - ➔ **separate Definition von Ereignissen und Identifikation (Ereignisname) wünschenswert**
  - vorteilhaft bei komplexen und zusammengesetzten Ereignissen

## Spezifikation von Ereignissen (2)

- **Primitive Ereignisse**
  - **Zeitereignis** – Zeitpunkt: absolut, relativ, periodisch wiederkehrend
  - **Methodenergebnis** – Reaktion auf eine Nachricht (BEFORE/AFTER)
  - **Wertereignis** – Operation auf einem Wert
  - **Transaktionsergebnis** – BOT/ABORT/COMMIT
  - **Abstraktes Ereignis**: wird vom System nicht automatisch erkannt
    - Vergabe eines Namens: DEFINE EVENT <Name>
    - Signalisieren des Ereignisses: RAISE <Name>
- **Zusammengesetzte Ereignisse**
  - Definition durch Ereignisalgebra  
sechs Ereigniskonstruktoren zur Verknüpfung von einfachen oder zusammengesetzten Ereignissen
  - **Verknüpfung** von Ereignissen
    - **Disjunktion**:  $E = (E1 \mid E2)$
    - **Sequenz**:  $E = (E1; E2)$
    - **Konjunktion**:  $E = (E1, E2)$
  - **Überwachung** des Eintretens von Ereignisinstanzen einer bestimmten Klasse innerhalb eines Zeitintervalls  $[s - e]$ 
    - **Negatives Ereignis**:  $\text{NOT } E [s_1 - e_1]$   
(NOT any) überwacht das Nicht-Eintreten aller Ereignisse
    - **Sternoperator**:  $*E [s_2 - e_2]$   
Das wiederholte Auftreten von E im Zeitintervall  $[s_2 - e_2]$  wird nur einmal, und zwar beim ersten Mal, signalisiert.
    - **Geschichtsoperator**:  $\text{TIMES}(n, E) [s_3 - e_3]$   
Ereignis wird signalisiert, sobald E innerhalb des Zeitintervalls n-mal aufgetreten ist

## Beispiele zur Ereignisspezifikation

- Gegeben: „Elementarereignisse“ E1, E2, E3
- Disjunktion:
- Sequenz:
- Konjunktion:
- Überwachungsintervall:
- Negation:
- Sternoperator:  
Gegebene Ereignisfolge: E1 E2 E1 E1 E2 E2 E2 E1
- Geschichtsoperator:

## Anwendung von ECA-Regeln

- weitere Aspekte
  - DROP, ALTER, ACTIVATE, DEACTIVATE ECA-RULE
  - Konfliktauflösung, z. B. Prioritätsreihenfolge von Regeln
  - Regelmengen
- Anwendung bei Integritätsbedingungen (allgemeine Constraints)

```
CREATE ECA-RULE Gehaltsbereich
AFTER INSERT ON Pers
    REFERENCING NEW AS Neu
CHECK
IF Neu.Gehalt < 20K OR Neu.Gehalt > 100K
DO
    ROLLBACK WORK
END ECA-RULE
```
- Anwendung bei materialisierten abgeleiteten Daten

```
CREATE ECA-RULE Einfügen-Top-Verdiener AS
AFTER INSERT ON Pers
    REFERENCING NEW AS Neu-Eingefügt
DO
    INSERT INTO Top-Verdiener
        SELECT * FROM Neu-Eingefügt
        WHERE Gehalt > 100K
END ECA-RULE
```
- „Selbstausslösende“ Regeln können rekursiv abgeleitete Daten warten

## Anwendung von ECA-Regeln (2)

- **Anwendung bei Triggern und Alertern**

```
CREATE ECA-RULE Große-Gehaltserhöhung As
  AFTER UPDATE OF Gehalt ON Pers
  REFERENCING OLD AS Alt NEW AS Neu
CHECK
  IF Neu.Gehalt - Alt.Gehalt > 10K
DO
  Benachrichtige-Manager (Alt.Gehalt, Neu.Gehalt)
END ECA-RULE
```

- **Weitere Einsatzmöglichkeiten**

neben Integritätssicherung und Kontrolle abgeleiteter Daten

- Einhaltung von Geschäftsregeln
- Übernahme nicht trivialer Aufgaben (Erkennung komplexer Situationen, „intelligente“ Reaktionen, selbständige Ausführung)
  - Verwaltung von Abhängigkeiten
  - Unterstützung von Kooperation (Benachrichtigung)
  - Aktienhandel
  - Fabrikautomatisierung
- Zugriffskontrolle und Autorisierung
- alle Überwachungsaufgaben (*Monitoring*)
  - Leistungsmessung (Sammlung von Statistiken, Lastbalancierung)
  - Fehlerkontrolle
  - Auditing
  - ...

## Regelausführung

- **Wann ist die Regelausführung korrekt?**

- Ergebnis kann von der Ausführungsreihenfolge abhängen
- Regeln können sich gegenseitig auslösen!
- Terminierung?

➔ Wir benötigen ein **Ausführungsmodell!**

- **Grundmodell**

- A1: In einer während des gesamten Systembetriebs laufenden Endlosschleife werden Ereignisse entdeckt und die zugehörigen Regeln ermittelt und ausgelöst
- A2: Wann immer bzw. solange ausgelöste Regeln vorliegen, werden folgende Schritte durchgeführt:
  - S1: Wähle eine ausgelöste Regel R aus
  - S2: Evaluiere die Bedingung C von R
  - S3: Wenn C wahr ist, führe die Aktion von R aus

- **Große Variationsbreite des Grundmodells**

- Aktivitäten A1 und A2 können durchaus **parallel und/oder zu vorher festgelegten Zeitpunkten** ausgeführt werden
- So können in einem konkreten System Ausführungsmodelle mit **sehr unterschiedlicher Regelausführungssemantik** implementiert werden

➔ **Ausführungsmodell bestimmt die Semantik der Regelausführung**

## Regelausführung (2)

- **Ausführungsgranulat:**

- **Wie oft wird A2 ausgeführt?**

- Prinzipiell kann A2 erfolgen, sobald in A1 eine Regel ausgelöst wurde
      - Oft wird A2 nur dann angestoßen, wenn
        - **interne DB-Operation** (ein Tupel betreffend)
        - **DB-Anweisung** (eine ganze Menge betreffend)
        - **Transaktion**
- endet

- **Beispiel auf Relation Pers:**

Event: DELETE FROM Pers WHERE Pers.Gehalt ≥ 100K

Action: Verringere Gehalt von Pers.Mgr um 10%

| Pnr | Gehalt  | Mgr |
|-----|---------|-----|
| P1  | 100.000 | P2  |
| P2  | 100.000 | P3  |

nach interner  
DB-Operation:

| Pnr | Gehalt  | Mgr |
|-----|---------|-----|
| P1  | 100.000 | P2  |
| P2  | 100.000 | P3  |

nach DB-Anweisung:

| Pnr | Gehalt  | Mgr |
|-----|---------|-----|
| P1  | 100.000 | P2  |
| P2  | 100.000 | P3  |

## Regelausführung (3)

- **Wann wird ausgelöst?**

- während/am Ende der Benutzeranweisung → **Bruttoeffekte!**
  - am Ende der Transaktion → **Nettoeffekte!**

Bruttoeffekt

Nettoeffekt

- **Tupel- oder mengenorientierte Ausführung (AVG-Problem)**

Event: ...

Action: **UPDATE** Pers P

**SET** P.Gehalt = 1.1 \* (**SELECT AVG** (Gehalt) **FROM** Pers);

- **tupelorientierte Auswertung**

- nach 1. Tupel: t1 t2 t3

- nach 2. Tupel: t1 t2 t3

- nach 3. Tupel: t1 t2 t3

- **mengenorientierte Auswertung**

- nach allen Tupeln: t1 t2 t3

## Regelausführung (4)

- Was passiert, wenn mehrere Regeln ausgelöst werden?
  - Bestimmung der Reihenfolge
  
  - Sequentielle Ausführung: nur eine Regel kann gleichzeitig ausgelöst werden: Konfliktauflösung, wenn erforderlich
  - Parallele Ausführung: mehrere Regeln können zu einem Zeitpunkt ausgelöst werden → **Korrektheitskriterien?**
  
- Wie hängt die Regelausführung mit dem TA-Konzept zusammen?
  - IMMEDIATE oder DEFERRED
  - in einer oder in mehreren Transaktionen
  - in gekoppelten oder entkoppelten Transaktionen

## Regelausführung (5)

- Bezug zwischen E, C und A
  - zeitlicher Bezug: IMMEDIATE, DEFERRED
  - Verarbeitungskontext: COUPLED, DECOUPLED
  - Übergabe von Parametern: E – C, E – A
  - Übergabe des Ergebnisses der Bedingung: C – A
  
- Trennung von Event und Condition wichtig!
  - Events spezifizieren, **wann** überprüft werden soll
  - Conditions spezifizieren, **was** überprüft werden soll
  - ermöglicht Optimierung: Überprüfung von Condition nur bei speziellen Events
  - erlaubt flexible Ausführung: Auswertung der Condition zu einem späteren Zeitpunkt oder in einer anderen Transaktion
  
- Kopplungs-Modi
  - Zu welchem Zeitpunkt und in welchem Kontext werden ausgelöste Regeln ausgeführt?
  - Zwischen TA, auf die sich Ereignis bezieht (trTA), und Regelausführungs-TA (REX) kann es verschiedene Arten der Kopplung geben
    - E – C } IMMEDIATE, DEFERRED
    - C – A } COUPLED
    - C – A } DEPENDENT/INDEPENDENT DECOUPLED
  - Wahl des Kopplungsmodus ist durch System fest zu implementieren oder durch regelspezifische Angaben der Anwender festzulegen

## Regelausführung (6)

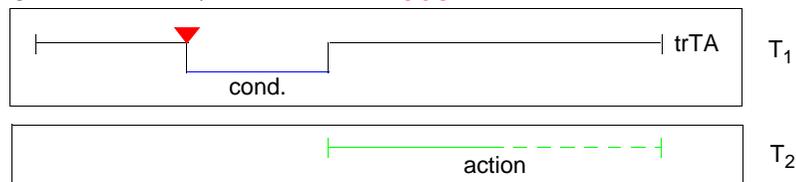
- Einsatzbeispiele mit verfeinerten Kopplungsmodi

- Beispiel 1

- Aktion erfordert komplexe Berechnung und bezieht sich auf die „Außenwelt“ (Bestellung, Benachrichtigung). Sie soll nur erfolgen, wenn trTA erfolgreich beendet wird

E-C: IMMEDIATE / COUPLED

C-A: IMMEDIATE / **DEPENDENT DECOUPLED**



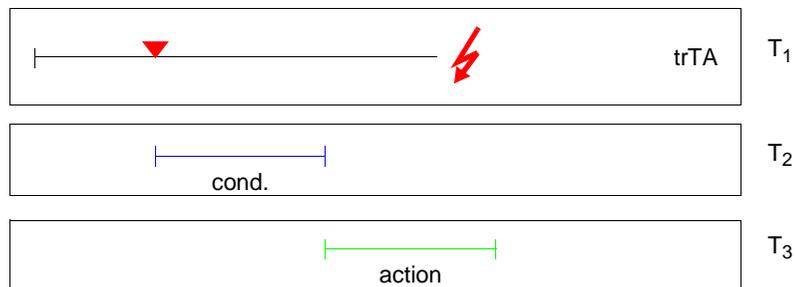
- **Commit (T2)** (als Sub-TA oder gekoppelte TA) wird **mit Commit (T1) synchronisiert**, wobei Nachrichten von T2 verzögert werden

- Beispiel 2

- „Verstöße“ einer TA sollen von unabhängigen TA geprüft und aufgezeichnet werden

E-C: IMMEDIATE / **INDEPENDENT DECOUPLED**

C-A: IMMEDIATE / **INDEPENDENT DECOUPLED**



## Zusammenfassung

- **Semantische Integritätskontrolle**

- Relationale Invarianten, referentielle Integrität und Aktionen
- Benutzerdefinierte Integritätsbedingungen (*assertions*)

➔ zentrale Spezifikation/Überwachung im DBS wird immer wichtiger

- **Aktives DB-Verhalten zur**

- Integritätssicherung
- Wartung abgeleiteter Daten
- Durchführung allgemeiner Aufgaben (Regeln, Alerter, Trigger)

- **Triggerkonzept in SQL:1999 standardisiert**

- **Verallgemeinertes Konzept: ECA-Regeln**

- *Event* Welche Events werden unterstützt?
- *Condition* Wie komplex sind Conditions?
- *Action* Wie komplex sind Actions?

- **Regelausführung**

**Datenänderungen triggern Regeln, die Daten ändern, die Regeln triggern, die Daten ändern, ...**

- ist inhärent dynamisch, prozedural und unstrukturiert
- kann zu nicht voraussagbaren Ergebnissen führen
- ist schwierig zu beschreiben (operationale Semantik)

➔ Dies trifft auf alle aktiven DBS zu!

➔ Die Entwicklung einer korrekten Menge von Regeln kann sehr schwierig sein!