1. Datendefinition und –kontrolle in SQL

- Datendefinition
  - Schema, Datentypen, Domains
  - Erzeugen von Tabellen (CREATE TABLE)
  - Schemaevolution: Ändern/Löschen von Tabellen
- Sichtkonzept (Views)
- ACID und Datenkontrolle
- Integritätsbedingungen
  - Klassifikation: Statische vs. Dynamische IB
  - Integritätsbedingungen in SQL
- Integritätsregeln / Trigger
- Zugriffskontrolle/Autorisierung: GRANT, REVOKE

Schemadefinition in SQL

- SQL-Umgebung (Environment) besteht aus
  - Katalogen: pro Datenbank ein Schema
  - Benutzern
  - INFORMATION_SCHEMA (Metadaten über alle Schemata)
    => dreiteilige Objektnamen: <catalog>.<schema>.<object>

```
CREATE SCHEMA      [schema] AUTHORIZATION user
                   [DEFAULT CHARACTER SET char-set]
                   [schema-element-list]
```

- Schema-Definition
  - jedes Schema ist einem Benutzer (user) zugeordnet, z.B. DBA
  - Definition aller
    - Definitionsbereiche
    - Basisrelationen
    - Sichten (Views),
    - Zugriffsrechte
    - Integritätsbedingungen

Beispiel:

```
CREATE SCHEMA  FLUG-DB
       AUTHORIZATION LH_DBA1
```

SQL92-Datentypen

- String-Datentypen

```
CHARACTER              [ ( length ) ]       (Abkürzung: CHAR)
CHARACTER VARYING      [ ( length ) ]       (Abkürzung: VARCHAR)
NATIONAL CHARACTER     [ ( length ) ]       (Abkürzung: NCHAR)
NCHAR VARYING          [ ( length ) ]
BIT                    [ ( length ) ]
BIT VARYING            [ ( length ) ]
```

- Numerische Datentypen

```
NUMERIC  [ ( precision [ , scale] ) ]
DECIMAL  [ ( precision [ , scale ] ) ]      (Abkürzung: DEC)
INTEGER                                     (Abkürzung: INT)
SMALLINT
FLOAT           [ ( precision ) ]
REAL
DOUBLE PRECISION
```

- Datums-/Zeitangaben (Datetimes)

```
DATE
TIME
TIMESTAMP
TIME WITH TIME ZONE
TIMESTAMP WITH TIME ZONE
INTERVAL        (* Datums- und Zeitintervalle *)
```

Definitionsbereiche (Domains)

```
CREATE DOMAIN  domain [AS] data-type
  [DEFAULT  { literal |  niladic-function-ref | NULL} ]
  [[CONSTRAINT constraint] CHECK (cond-exp) [deferrability]]
```

- Festlegung zulässiger Werte durch Domain-Konzept
- Wertebereichseingrenzung durch benamte CHECK-Constraint
- Beispiele:

```
CREATE DOMAIN ABTNR AS CHAR (6)
CREATE DOMAIN AGE AS INT DEFAULT NULL
                  CHECK (VALUE=NULL OR VALUE > 18)
```

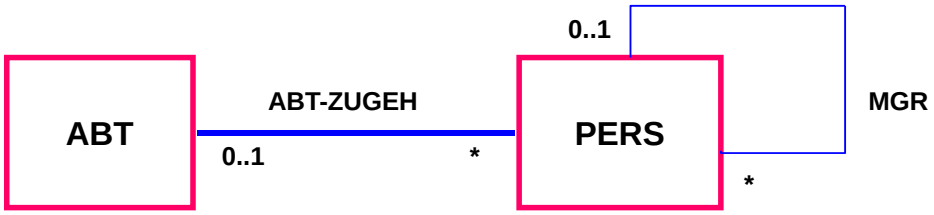
- Beschränkungen
  - keine echten benutzerdefinierten Datentypen
  - keine strenge Typprüfung
  - Domains können in SQL-92 nur bzgl. Standard-Datentypen (nicht über andere Domains) definiert werden

Erzeugung von Basisrelationen

```
CREATE [ [GLOBAL | LOCAL] TEMPORARY] TABLE base-table
      (base-table-element-commalist)
      [ON COMMIT {DELETE | PRESERVE} ROWS]
base-table-element ::= column-def | base-table-constraint-def
```

- permanente und temporäre Relationen
- zwei Typen von temporären Relationen:
  - LOCAL: Lebensdauer auf erzeugende Transaktion begrenzt
  - GLOBAL: Lebensdauer auf „Session“ eines Benutzers begrenzt; Inhalt kann beim Commit zurückgesetzt werden
- Bei der Attributdefinition (column definition) werden folgende Angaben / Integritätsbedingungen spezifiziert:
  - Attributname sowie Datentyp bzw. Domain
  - Default-Werte
  - Eindeutigkeit (UNIQUE bzw. PRIMARY KEY)
  - FOREIGN-KEY-Klausel
  - Verbot von Nullwerten (NOT NULL)
  - CHECK-Bedingung

CREATE TABLE: Beispiel




```
CREATE TABLE PERS
    (PNR      INT                    PRIMARY KEY,
    BERUF     VARCHAR (50),
    PNAME     VARCHAR (50)           NOT NULL,
    PALTER    AGE,     (* siehe Domain-Definition *)
    MGR       INT                    REFERENCES PERS,
    ANR       ABTNR       (*  Domain-Definition *)
    GEHALT    DEC (7) DEFAULT 0  CHECK (VALUE < 120000)
    FOREIGN KEY (ANR)  REFERENCES ABT )

CREATE TABLE ABT
    (ANR      ABTNR                  PRIMARY KEY,
    ANAME     VARCHAR (50)           NOT NULL)
```


Dynamische Änderung einer Relation

```
ALTER  TABLE  base-table
      {  ADD [COLUMN] column-def
       | ALTER [COLUMN] column {SET default-def  | DROP DEFAULT}
       | DROP [COLUMN] column {RESTRICT | CASCADE}
       | ADD  base-table-constraint-def
       | DROP  CONSTRAINT constraint {RESTRICT | CASCADE}}
```

- Schema-Evolution: dynamische Schemaanpassungen während der Lebenszeit (Nutzung) der Relationen
  - Hinzufügen, Ändern und Löschen von Attributen
  - Hinzufügen und Löschen von Check-Constraints
- Beispiele

  ALTER TABLE PERS ADD COLUMN SVNR INT UNIQUE

  ALTER TABLE PERS DROP GEHALT RESTRICT

  - *RESTRICT* : Rückweisung der Operation, wenn das zu löschende Attribut (Column) in einer Sicht oder einer Integritätsbedingung (Check) referenziert wird
  - *CASCADE:* Folgelöschung aller Sichten und Check-Klauseln, die von dem Attribut abhängen

Löschen von Objekten

```
DROP { TABLE base-table | VIEW view  | DOMAIN domain |
             SCHEMA schema }
       {RESTRICT | CASCADE}
```

- Entfernung nicht mehr benötigter Objekte (Relationen, Sichten, ...)
  - *CASCADE:* 'abhängige' Objekte (z.B. Sichten auf Relationen oder anderen Sichten) werden mitentfernt
  - *RESTRICT*: verhindert Löschen, wenn die zu löschende Relation noch durch Sichten oder Integritätsbedingungen referenziert wird
- Beispiele:

  DROP DOMAIN AGE

  DROP TABLE PERS RESTRICT

Sichtkonzept

- Sicht (View): mit Namen bezeichnete, aus Basisrelationen abgeleitete, virtuelle Relation (Anfrage)
- Korrespondenz zum externen Schema bei ANSI/SPARC (Benutzer sieht jedoch i.a. mehrere Views und Basisrelationen)

```
CREATE VIEW view [ (column-commalist ) ] AS table-exp
      [WITH [ CASCADED | LOCAL] CHECK OPTION]
```

- Beispiel: Sicht auf PERS, die alle Programmierer mit einem Gehalt unter 30000 umfasst

```
CREATE VIEW
   ARME_PROGRAMMIERER (PNR, NAME, BERUF, GEHALT, ANR) AS
            SELECT PNR, NAME, BERUF, GEHALT, ANR
            FROM PERS
            WHERE BERUF = 'Programmierer' AND
                        GEHALT < 30 000
```

Sichtkonzept (2)

- Sicht kann wie eine Relation behandelt werden
  - Anfragen / Anwendungsprogramme auf Sichten
  - Sichten auf Sichten sind möglich
- Vorteile:
  - Erhöhung der Benutzerfreundlichkeit
  - erhöhte Datenunabhängigkeit / verbesserte Schema-Evolution
  - Datenschutz / Zugriffskontrolle

Sichtkonzept (3)

- Sichtsemantik
  - allgemeine Sichten werden nicht materialisiert, sondern als Anfrageergebnis interpretiert, das dynamisch beim Zugriff generiert wird
  - Sicht entspricht einem „dynamisches Fenster“ auf zugrundeliegenden Basisrelationen
  - Sicht-Operationen müssen durch (interne) *Query-Umformulierung* auf Basisrelationen abgebildet werden
  - eingeschränkte Änderungen: aktualisierbare und nicht-aktualisierbare Sichten

- Sonderform: *Materialisierte Sichten*
  - physische Speicherung des Anfrageergebnisses
  - unterstützt schnelleren Lesezugriff
  - Notwendigkeit der Aktualisierung (automatisch durch das DBS)
  - erhöhter Speicherbedarf
  - kein Bestandteil von SQL92, jedoch in vielen DBS verfügbar (CREATE MATERIALIZED VIEW ...)

Sichtkonzept (4)

- Abbildung von Sicht-Operationen auf Basisrelationen
  - Sichten werden i.a. nicht explizit und permanent gespeichert, sondern Sicht-Operationen werden in äquivalente Operationen auf Basisrelationen umgesetzt
  - Umsetzung ist für Leseoperationen meist unproblematisch

```
SELECT  NAME, GEHALT
FROM  ARME_PROGRAMMIERER
WHERE ANR = ‘A05‘
```

```
SELECT  NAME, GEHALT
FROM  PERS
WHERE ANR = ‘A05‘
```

- Abbildungsprozess auch über mehrere Stufen durchführbar

```
CREATE  VIEW  V  AS  SELECT  ... FROM R  WHERE  P
CREATE  VIEW  W  AS  SELECT  ... FROM V  WHERE  Q

SELECT ... FROM  W  WHERE  C
```

```
SELECT  …FROM  V
WHERE C  AND Q
```

Sichtkonzept (5)

- Problemfälle aufgrund von SQL-Einschränkungen
  - keine Schachtelung von Aggregatfunktionen und Gruppenbildung (GROUP-BY)
  - keine Aggregatfunktionen in WHERE-Klausel möglich

```
CREATE VIEW ABTINFO (ANR, GSUMME)AS
    SELECT ANR, SUM(GEHALT)
    FROM PERS
    GROUP BY ANR
```

```
SELECT AVG (GSUMME) FROM ABTINFO
```


Sichtkonzept (6)

- Probleme für Änderungsoperationen auf Sichten
  - erfordern, dass zu jedem Tupel der Sicht zugrundeliegende Tupel der Basisrelationen eindeutig identifizierbar sind
  - Sichten auf einer Basisrelation sind nur aktualisierbar, wenn der Primärschlüssel in der Sicht enthalten ist.
  - Sichten, die über Aggregatfunktionen oder Gruppenbildung definiert sind, sind nicht aktualisierbar
  - Sichten über mehr als eine Relation sind im allgemeinen nicht aktualisierbar

```
CREATE VIEW READONLY (BERUF, GEHALT) AS
       SELECT BERUF, GEHALT  FROM PERS
```

- CHECK-Option:
  - Einfügungen und Änderungen müssen das die Sicht definierende Prädikat erfüllen. Sonst: Zurückweisung
  - nur auf aktualisierbaren Sichten definierbar
- Löschen von Sichten:

```
DROP VIEW  ARME_PROGRAMMIERER CASCADE
```

ACID und Datenkontrolle

- Transaktionskonzept (ACID-Eigenschaften)
  - im SQL-Standard: COMMIT WORK, ROLLBACK WORK, Beginn einer Transaktion implizit
  - Einhaltung der logischen DB-Konsistenz (Consistency)
  - Verdeckung der Nebenläufigkeit (concurrency isolation)
  - Verdeckung von (erwarteten) Fehlerfällen (-> Logging und Recovery)
- Integritätskontrolle
  - *Semantische Integritätskontrolle:* möglichst hohe Übereinstimmung von DB-Inhalt und Miniwelt (Datenqualität)
  - nur 'sinnvolle' und 'zulässige' Änderungen der DB: bei COMMIT müssen alle semantischen Integritätsbedingungen erfüllt sein (Transaktionskonsistenz)
  - Einhaltung der *physischen Integrität* sowie der *Ablaufintegrität* (operationale Integrität)
- Zugriffskontrolle
  - Maßnahmen zur Datensicherheit und zum Datenschutz
  - Sichtkonzept
  - Vergabe und Kontrolle von Zugriffsrechten

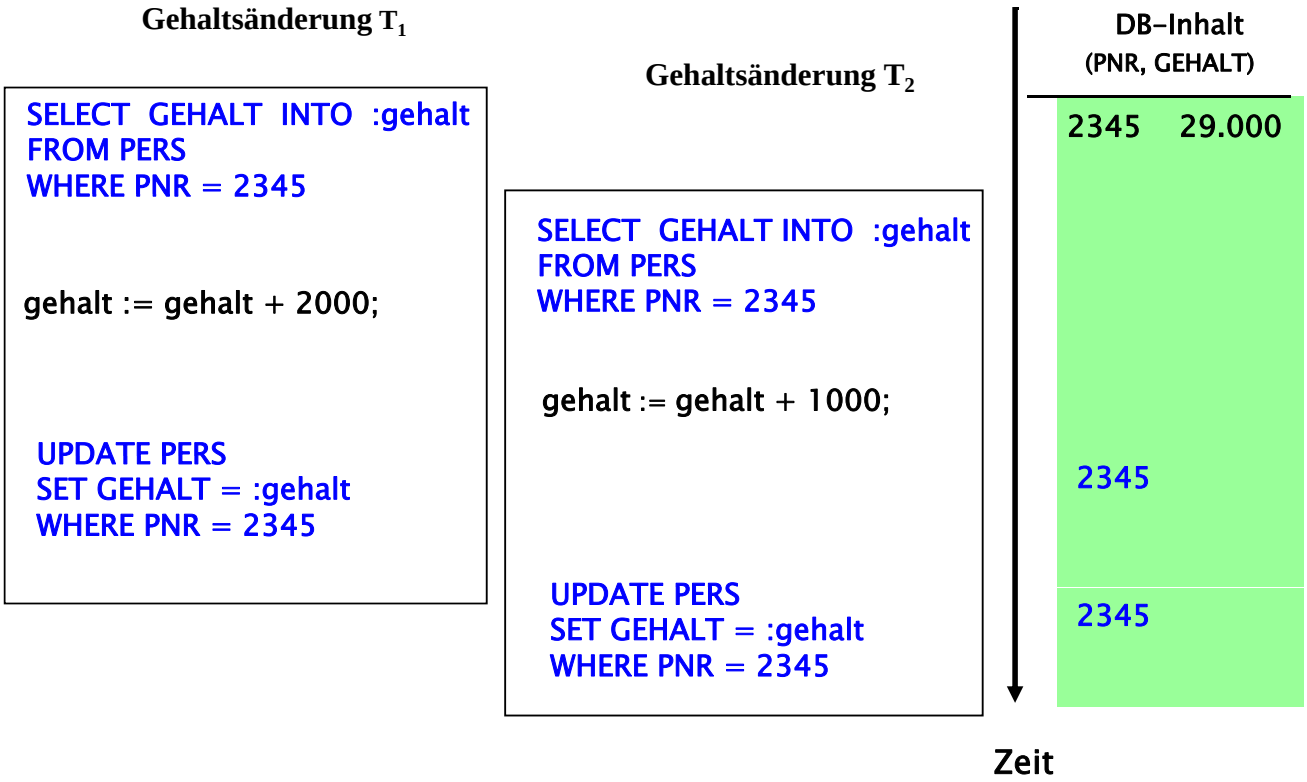
ACID und Datenkontrolle (2)

Art der Integrität	Transaktions-eigenschaft	realisierende DBS-Komponente
Semantische Integrität	C (Consistency, Konsistenz)	(Semantische) Integritätskontrolle
Physische Integrität	D: Dauerhaftigkeit A: Atomarität	Logging, Recovery (+ korrekte Implementierung der DB-Operationen)
Ablaufintegrität	I (Isolation)	Synchronisation (z. B. Sperrverwaltung)

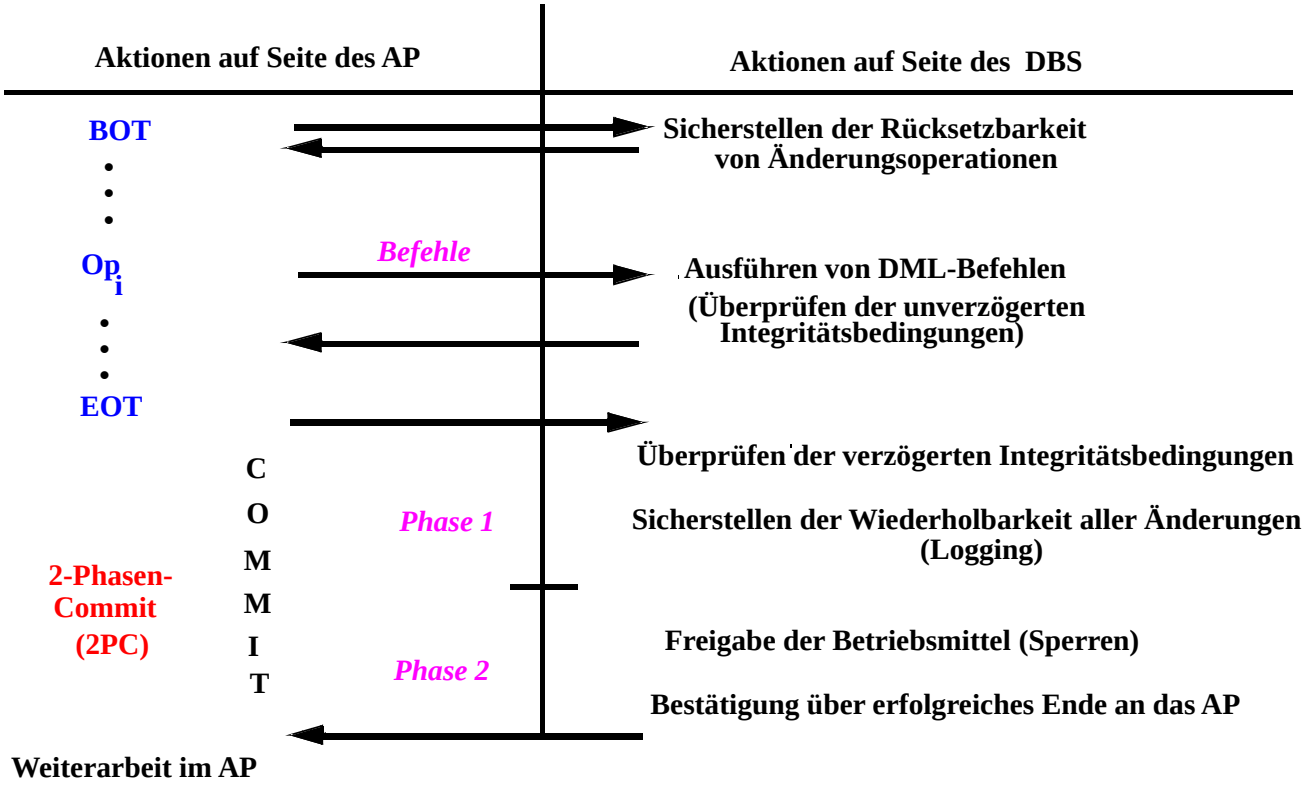
Synchronisation

- DBS müssen Mehrbenutzerbetrieb unterstützen
- ohne Synchronisation kommt es zu so genannten Mehrbenutzer-Anomalien
  - Verlorengegangene Änderungen (lost updates)
  - Abhängigkeiten von nicht freigegeben Änderungen (dirty read, dirty overwrite)
  - inkonsistente Analyse (non-repeatable read)
  - Phantom-Probleme
- Anomalien sind nur durch Änderungen verursacht
- Synchronisation erfolgt automatisch durch das DBS, z.B. durch Setzen von Sperren vor Datenzugriff (Freigabe der Sperren am Transaktionsende)

Verloren gegangene Änderung (Lost Update)

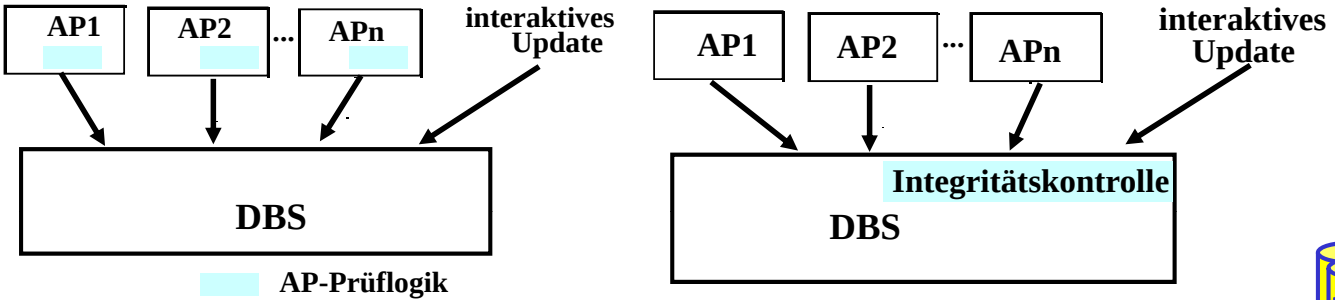


Die Transaktion als Schnittstelle zwischen Anwendungsprogramm und DBS



(Semant.) Integritätskontrolle

- Wahrung der logischen DB-Konsistenz
- Überwachung von semantischen Integritätsbedingungen durch Anwendungen oder durch DBS
- DBS-basierte Integritätskontrolle
  - größere Sicherheit
  - vereinfachte Anwendungserstellung
  - Unterstützung von interaktiven sowie programmierten DB-Änderungen
  - leichtere Änderbarkeit von Integritätsbedingungen
- Integritätsbedingungen der Miniwelt sind explizit bekannt zu machen, um automatische Überwachung zu ermöglichen



Klassifikation von Integritätsbedingungen

1. Modellinhärente Integritätsbedingungen (vs. Anwendungsspezifische IB)
   - Primärschlüsseleigenschaft
   - referentielle Integrität für Fremdschlüssel
   - Definitionsbereiche (Domains) für Attribute
2. Reichweite

Reichweite	Beispiele
Attribut	GEB-JAHR ist numerisch, 4-stellig
Satzausprägung	ABT.GEHALTSSUMME < ABT.JAHRESETAT
Satztyp	PNR ist eindeutig
mehrere Satztypen	ABT.GEHALTSSUMME ist Summe aller Angestelltengehälter

3. Zeitpunkt der Überprüfbarkeit
   - unverzögert (sofort bei Änderungsoperation)
   - verzögert (am Transaktionsende)
4. Art der Überprüfbarkeit
   - Zustandsbedingungen (statische Integritätsbedingungen)
   - dynamische Integritätsbedingungen

Dynamische Integritätsbedingungen

- Beziehen sich im Gegensatz zu statischen IB auf Änderungen selbst und damit auf mehrere Datenbankzustände
- Zwei Varianten
  - *Übergangsbedingungen*: Änderung von altem zu neuem DB-Zustand wird eingeschränkt
  - *temporale Bedingungen*: Änderungen in bestimmtem zeitlichen Fenster werden eingeschränkt
- Beispiele dynamischer Integritätsbedingungen
  - Übergang von FAM-STAND von 'ledig' nach 'geschieden' ist unzulässig
  - Gehalt darf nicht kleiner werden
  - Gehalt darf innerhalb von 3 Jahren nicht um mehr als 25% wachsen

Integritätsbedingungen in SQL

- Eindeutigkeit von Attributwerten
  - UNIQUE bzw. PRIMARY KEY bei CREATE TABLE
  - Satztypbedingungen

  Bsp.: CREATE TABLE PERS ...
  PNR INT UNIQUE (bzw. PRIMARY KEY)

- Fremdschlüsselbedingungen
  - FOREIGN-KEY-Klausel
  - Satztyp- bzw. satztypübergreifende Bedingung

- Wertebereichsbeschränkungen von Attributen
  - CREATE DOMAIN
  - NOT NULL
  - DEFAULT
  - Attribut- und Satztyp-Bedingungen

Integritätsbedingungen in SQL (2)

- Allgemeine Integritätsbedingungen
  - CHECK-Constraints bei CREATE TABLE
  - allgemeine Assertions, z. B. für satztypübergreifende Bedingungen

CHECK-Constraints bei CREATE TABLE

```
CREATE TABLE PERS ....
   GEB-JAHR  INT
   CHECK (VALUE BETWEEN 1900 AND 2100)
CREATE TABLE ABT .....
   CHECK (GEHALTSSUMME <  JAHRESETAT)
```

Anweisung CREATE ASSERTION

```
CREATE ASSERTION A1
   CHECK (NOT EXISTS
   (SELECT * FROM ABT A
    WHERE GEHALTSSUMME <>
   (SELECT SUM (P.GEHALT) FROM PERS P
         WHERE P.ANR = A.ANR)))
   DEFERRED
```

- Festlegung des Überprüfungszeitpunktes:
  - IMMEDIATE: am Ende der Änderungsoperation (Default)
  - DEFERRED: am Transaktionsende (COMMIT)
- Unterstützung für dynamische Integritätsbedingungen durch Trigger (ab SQL:1999)

Integritätsregeln

- Standardreaktion auf verletzte Integritätsbedingung: ROLLBACK
- Integritätsregeln erlauben Spezifikation von Folgeaktionen, z. B. um Einhaltung von Integritätsbedingungen zu erreichen
  - SQL92: deklarative Festlegung referentieller Folgeaktionen (CASCADE, SET NULL, ...)
  - SQL99: Trigger
- Trigger: Festlegung von Folgeaktionen für Änderungsoperationen
  - INSERT
  - UPDATE oder
  - DELETE
- Trigger wesentlicher Mechanismus von ***aktiven DBS***
- Verallgemeinerung durch sogenannte ECA-Regeln (Event / Condition / Action)

Integritätsregeln (2)

- Beispiel: Wartung der referentiellen Integrität
  - **Deklarativ**

```
CREATE TABLE PERS
 (PNR INT PRIMARY KEY,
    ANR INT FOREIGN KEY
     REFERENCES ABT
     ON DELETE CASCADE
   ... );
```

  - **Durch Trigger**

```
CREATE TRIGGER MITARBEITERLÖSCHEN
BEFORE DELETE ON ABT
REFERENCING OLD AS A
DELETE FROM PERS P
WHERE P.ANR = A.ANR;
```

Trigger

- ausführbares, benanntes DB-Objekt, das implizit durch bestimmte Ereignisse (“triggering event”) aufgerufen werden kann
- Triggerspezifikation besteht aus
  - auslösendem Ereignis (Event)
  - Ausführungszeitpunkt
  - optionaler Zusatzbedingung
  - Aktion(en)
- zahlreiche Einsatzmöglichkeiten
  - Überwachung nahezu aller Integritätsbedingungen, inkl. dynamischer Integritätsbedingungen
  - Validierung von Eingabedaten
  - automatische Erzeugung von Werten für neu eingefügten Satz
  - Wartung replizierter Datenbestände
  - Protokollieren von Änderungsbefehlen (Audit Trail)
  - • • •

Trigger (2)

- SQL99-Syntax

```
CREATE TRIGGER <trigger name>
 {BEFORE|AFTER}{INSERT|DELETE|
   UPDATE [OF <column list>]}
  ON <table name>
 [ORDER <order value>]
 [REFERENCING <old or new alias list>]
 [FOR EACH {ROW|STATEMENT}]
 [WHEN (<search condition>)]
   <triggered SQL statement>
```

```
<old or new alias> ::=
OLD [AS]<old values correlation name>|
NEW [AS]<new values correlation name>|
OLD_TABLE [AS]<old values table alias>|
NEW_TABLE [AS]<new values table alias>
```

  - Trigger-Events: INSERT, DELETE, UPDATE
  - Zeitpunkt: BEFORE oder AFTER
  - mehrere Trigger pro Event/Zeitpunkt möglich (benutzerdefinierte Aktivierungsreihenfolge)
  - Bedingung: beliebiges SQL-Prädikat (z. B. mit komplexen Subqueries)
  - Aktion: beliebige SQL-Anweisung (z. B. auch neue prozedurale Anweisungen)
  - Trigger-Bedingung und -Aktion können sich sowohl auf alte als auch neue Tupelwerte der betroffenen Tupel beziehen
  - Trigger-Ausführung für jedes betroffene Tupel einzeln (FOR EACH ROW) oder nur einmal für auslösende Anweisung (FOR EACH STATEMENT)

Trigger-Beispiele

- Realisierung einer dynamischen Integritätsbedingung

```
CREATE TRIGGER GEHALTSTEST
AFTER UPDATE OF GEHALT ON PERS
REFERENCING OLD AS AltesGehalt,
NEW AS NeuesGehalt
WHEN (NeuesGehalt < AltesGehalt)
        ROLLBACK;
```

- Wartung einer materialisierten Sicht ARME_PROGRAMMIERER

```
CREATE TRIGGER AP-INSERT
AFTER INSERT ON PERS
FOR EACH ROW
REFERENCING NEW AS N
WHEN N.BERUF = „Programmierer“ AND N.GEHALT < 30000
  INSERT INTO ARME_PROGRAMMIERER
  VALUES (N.PNR, N.NAME, N.BERUF, …)
```

Probleme von Triggern

- teilweise prozedurale Semantik (Zeitpunkte, Verwendung alter/neuer Werte, Aktionsteil im Detail festzulegen)
- Trigger derzeit beschränkt auf Änderungsoperationen einer Tabelle (UPDATE, INSERT, DELETE)
- derzeit i. a. keine verzögerte Auswertung von Triggern
- Gefahr zyklischer, nicht-terminierender Aktivierungen
- Korrektheit des DB-/Trigger-Entwurfes (Regelabhängigkeiten, parallele Regelausführung, ...)

Dennoch sind Trigger sehr mächtiges und wertvolles Konstrukt, auch zur DBS-internen Nutzung

Zugriffskontrolle in SQL

- Sicht-Konzept: wertabhängiger Zugriffsschutz
  - Untermengenbildung
  - Verknüpfung von Relationen
  - Verwendung von Aggregatfunktionen
- GRANT-Operation: Vergabe von Rechten auf Relationen bzw. Sichten

```
GRANT  {privileges-commalist | ALL PRIVILEGES}
       ON accessible-object TO grantee-commalist [WITH GRANT OPTION]
```

- Zugriffsrechte: SELECT, INSERT, UPDATE, DELETE, REFERENCES, USAGE
  - Erzeugung einer „abhängigen“ Relation erfordert REFERENCES-Recht auf von Fremdschlüsseln referenzierten Relationen
  - USAGE erlaubt Nutzung spezieller Wertebereiche (character sets)
  - Attributeinschränkung bei INSERT, UPDATE und REFERENCES möglich
  - dynamische Weitergabe von Zugriffsrechten: WITH GRANT OPTION (dezentrale Autorisierung)
- Empfänger: Liste von Benutzern bzw. PUBLIC

Vergabe von Zugriffsrechten: Beispiele

- GRANT SELECT ON ABT TO PUBLIC
- GRANT INSERT, DELETE ON ABT TO Mueller, Weber WITH GRANT OPTION
- GRANT UPDATE (GEHALT) ON PERS TO Schulz
- GRANT REFERENCES (PRONR) ON PROJEKT TO PUBLIC

Rücknahme von Zugriffsrechten: Revoke

```
REVOKE      [GRANT OPTION FOR]  privileges-commalist
      ON  accessible-object
      FROM grantee-commalist {RESTRICT | CASCADE }
```

- ggf. fortgesetztes Zurücknehmen von Zugriffsrechten

Beispiel: REVOKE SELECT
ON ABT
FROM Weber CASCADE

- wünschenswerte Entzugssemantik:
  - Der Entzug eines Rechtes ergibt einen Zustand der Zugriffsberechtigungen, als wenn das Recht nie erteilt worden wäre
- Probleme:
  - Rechteempfang aus verschiedenen Quellen
  - Zeitabhängigkeiten

Revoke (2)

- Führen der Abhängigkeiten in einem *Autorisierungsgraphen* pro Objekt (zB Tabelle)
  - *Knoten: User-Ids*
  - *Gerichtete Kanten: Weitergabe von Zugriffsrechten (Recht, ggf. Zeitpunkt)*
- Beispiel für Tabelle R
  - User A: GRANT UPDATE ON R TO B WITH GRANT OPTION
  - User B: GRANT UPDATE ON R TO C
  - User D: GRANT UPDATE ON R TO B WITH GRANT OPTION
  - User A: REVOKE UPDATE ON R FROM B CASCADE
- Autorisierungsgraph für R:

Zusammenfassung

- Datendefinition:
  - CREATE / DROP TABLE, VIEW, ...;
  - ALTER TABLE
- Sicht-Konzept (Views)
  - Reduzierung von Komplexität
  - erhöhte Datenunabhängigkeit
  - Zugriffsschutz
  - Einschränkungen bezüglich Änderbarkeit
- Integritätsbedingungen
  - Klassifikation gemäß 4 Kriterien
  - Umfassende Unterstützung in SQL92
- Trigger
  - automatische Reaktion bei DB-Änderungen (-> „aktive DBS“)
  - zahlreiche Anwendungsmöglichkeiten: Integritätskontrolle, materialisierte Sichten, ..
- dezentrales Autorisierungskonzept (Grant with Grant Option)