6. Datendefinition und -kontrolle in SQL

- Datendefinition
  - Datentypen, Domains
  - Erzeugen von Tabellen (CREATE TABLE)
  - Ändern/Löschen von Tabellen (Schemaevolution)
- Sichtkonzept (Views)
- Zugriffskontrolle/Autorisierung: GRANT, REVOKE
- Integritätsbedingungen und Trigger
  - Klassifikation von Integritätsbedingungen
  - Integritätsregeln / Trigger
  - Einsatzformen von Triggern

Datendefinition in SQL

- SQL-Umgebung (Environment) besteht aus Katalogen und Benutzern (authorization identifiers)
- ein Katalog enthält:
  - für jede Datenbank ein Schema
  - ein INFORMATION_SCHEMA (Metadaten über alle Schemata)
    => dreiteilige Objektnamen: <catalog>.<schema>.<object>
- Schema-Definition

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

  - jedes Schema ist einem Benutzer (user) zugeordnet, z.B. DBA
  - Schema erhält Benutzernamen, falls keine explizite Namensangabe erfolgt
  - Definition aller Definitionsbereiche, Basisrelationen, Sichten (Views), Integritätsbedingungen und Zugriffsrechte
- 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)

- Festlegung zulässiger Werte durch Domain-Konzept

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

- optionale Angabe von Default-Werten
- Wertebereichseingrenzung durch benamte CHECK-Constraint
- Beispiele:

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

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

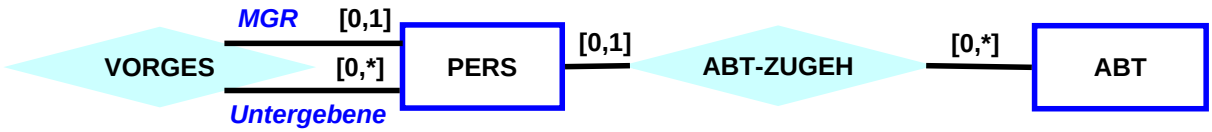
Erzeugung von Basisrelationen

```
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 spezifiziert:
  - Attributname sowie Datentyp bzw. Domain
  - Eindeutigkeit (UNIQUE bzw. PRIMARY KEY), FOREIGN-KEY-Klausel
  - Verbot von Nullwerten (NOT NULL), CHECK-Bedingung, Default-Werte
- Integritätsbedingungen, die mehrere Attribute der Relation betreffen, können als eigene „table constraint“ definiert werden

CREATE TABLE: Beispiel




```
CREATE  TABLE   PERS
    (PNR        INT                     PRIMARY KEY,
    BERUF       VARCHAR (50),
    PNAME       VARCHAR (50)            NOT NULL,
    PALTER      ALTER,      (* 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

- Bei Relationen können dynamisch (während ihrer Lebenszeit) Schemaänderungen durchgeführt werden (Schema-Evolution)
  - Hinzufügen, Ändern und Löschen von Attributen
  - Hinzufügen und Löschen von Constraints

```
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}}
```

- Beispiele

  ALTER TABLE PERS ADD SVNR INT UNIQUE
  ALTER TABLE PERS DROP COLUMN SVNR RESTRICT

  - RESTRICT führt zur Rückweisung der Operation, wenn das zu löschende Attribut (Column) in einer Sicht oder einer Integritätsbedingung (Check) referenziert wird
  - CASCADE dagegen erzwingt die 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}
```

- Nicht mehr benötigte Objekte (Relationen, Sichten, ...) können durch die DROP-Anweisung aus dem System entfernt werden
  - Mit der CASCADE-Option können 'abhängige' Objekte (z.B. Sichten auf Relationen oder anderen Sichten) mitentfernt werden
  - RESTRICT verhindert Löschen, wenn die zu löschende Relation noch durch Sichten oder Integritätsbedingungen referenziert wird
- Beispiele:

  ALTER TABLE PERS DROP CONSTRAINT PersConstraint CASCADE

  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
```


Datenkontrolle

- Zugriffskontrolle
  - Maßnahmen zur Datensicherheit und zum Datenschutz
  - Sichtkonzept
  - Vergabe und Kontrolle von Zugriffsrechten
- Integritätskontrolle
  - *Semantische Integritätskontrolle*
  - Einhaltung der *physischen Integrität* sowie der *Ablaufintegrität* (operationale Integrität)
- Transaktionskonzept (ACID-Eigenschaften)
  - Verarbeitungsklammer für die Einhaltung von semantischen Integritätsbedingungen
  - im SQL-Standard: COMMIT WORK, ROLLBACK WORK, Beginn einer Transaktion implizit
  - Verdeckung der Nebenläufigkeit (concurrency isolation)
  - Verdeckung von (erwarteten) Fehlerfällen (-> Logging und Recovery)

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

Zugriffskontrolle in SQL (2)

- Rücknahme von Zugriffsrechten:

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

Beispiel: REVOKE SELECT ON ABT FROM Weber CASCADE

- ggf. fortgesetztes Zurücknehmen von Zugriffsrechten
- 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
- Führen der Abhängigkeiten in einem *Autorisierungsgraphen*

Semantische Integritätsbedingungen

- Ziele
  - nur 'sinnvolle' und 'zulässige' Änderungen der DB
  - möglichst hohe Übereinstimmung von DB-Inhalt und Miniwelt (Datenqualität)
- bei COMMIT müssen alle semantischen Integritätsbedingungen erfüllt sein (Transaktionskonsistenz)
- zentrale Überwachung von semantischen Integritätsbedingungen durch DBS anstelle durch Anwendungen
  - größere Sicherheit
  - vereinfachte Anwendungserstellung
  - leichtere Änderbarkeit von Integritätsbedingungen
  - Leistungsvorteile
  - Unterstützung von interaktiven sowie programmierten DB-Änderungen
- Integritätsbedingungen der Miniwelt sind explizit bekannt zu machen, um automatische Überwachung zu ermöglichen
  - seit SQL92 umfangreiche Spezifikationsmöglichkeiten (relationale Invarianten, benutzerdefinierte Integritätsbedingungen)

Klassifikation von Integritätsbedingungen

1. modellinhärente vs. anwendungsspezifische Bedingungen

   modellinhärente Integritätsbedingungen, z. B. Relationale Invarianten (Primärschlüsselbedingung, referentielle Integrität) und Wertebereichsbeschränkung 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 SQL92

- 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 SQL92 (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)
- keine direkte Unterstützung für dynamische Integritätsbedingungen in SQL92 -> Trigger (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
AFTER 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, u.a. zur DBS-internen Nutzung

Zusammenfassung

- Datendefinition:
  - CREATE / DROP TABLE, VIEW, ...;
  - ALTER TABLE
  - Standardisierte SQL-Sichten für Metadaten (INFORMATION SCHEMA)
- Sicht-Konzept (Views)
  - Reduzierung von Komplexität, erhöhte Datenunabhängigkeit, Zugriffsschutz
  - Einschränkungen bezüglich Änderbarkeit
- dezentrales Autorisierungskonzept (GRANT, REVOKE)
- Integritätsbedingungen
  - Klassifikation gemäß 4 Kriterien
  - Umfassende Unterstützung in SQL92
- Trigger
  - automatische Reaktion bei DB-Änderungen (-> „aktive DBS“)
  - zahlreiche Anwendungsmöglichkeiten