6. Datendefinition und Zugriffskontrolle in SQL

- Datendefinition
  - Schema, Datentypen, Domains
  - Erzeugen von Tabellen (CREATE TABLE)
  - Schemaevolution: Ändern/Löschen von Tabellen
- Sichtkonzept (Views)
- Zugriffskontrolle/Autorisierung: GRANT, REVOKE

Integritätsbedingungen und Trigger

siehe DBS2

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 ALTER AS INT DEFAULT NULL
                    CHECK (VALUE=NULL OR VALUE > 18)
```

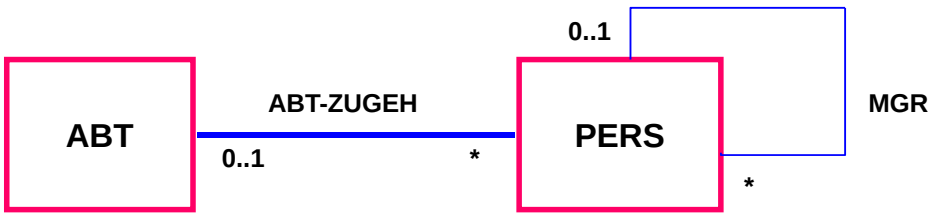
- Beschränkungen
  - keine echten benutzerdefinierten Datentypen
  - keine strenge Typprüfung
  - Domains können in SQL-92 nur bzgl. Standard-Datentypen (nicht über andere Domains) definiert werden

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    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

```
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 Alter

  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
```

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
  - Standardisierte SQL-Sichten für Metadaten (INFORMATION SCHEMA)
- Sicht-Konzept (Views)
  - Reduzierung von Komplexität
  - erhöhte Datenunabhängigkeit
  - Zugriffsschutz
  - Einschränkungen bezüglich Änderbarkeit
- dezentrales Autorisierungskonzept
  - GRANT (with Grant Option)
  - REVOKE