

7. Datendefinition in SQL

■ Datendefinition

- Schema, Datentypen, Domains
- Erzeugen von Tabellen (CREATE TABLE)
- Einsatz von Large Objects (BLOB, CLOB)
- Schemaevolution: Ändern/Löschen von Tabellen

■ Sichtkonzept (Views)

- CREATE VIEW / DROP VIEW
- Problemfälle (nicht änderbare Views)
- materialisierte Sichten

Def. von Integritätsbedingungen -> Kap. 8

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, TIMESTAMPTAMP	
TIME WITH TIME ZONE	
TIMESTAMP WITH TIME ZONE	
INTERVAL	(* Datums- und Zeitintervalle *)

SQL99-Datentypen

- Boolean
- Large Objects (für Texte, Fotos, etc. in der Datenbank)
 - **BLOB** (Binary Large Object)
 - **CLOB** (Character Large Object): Texte mit 1-Byte Character-Daten
 - **NCLOB** (National Character Large Objects): 2-Byte Character-Daten für nationale Sonderzeichen (z. B. Unicode)
- Komplexere Typen (-> Vorlesung DBS2)
 - **ROW**: zusammengesetzte Attribute
 - **ARRAY**
 - **MULTISET**: mengenwertige Attribute (seit SQL:2003)
 - user-defined types

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 benannte CHECK-Constraint
- Beispiele:

```
CREATE DOMAIN ABTNR AS CHAR (6)
```

```
CREATE DOMAIN AGE AS INT DEFAULT NULL
```

```
        CHECK (VALUE=NULL OR VALUE > 18)
```

■ Beschränkungen

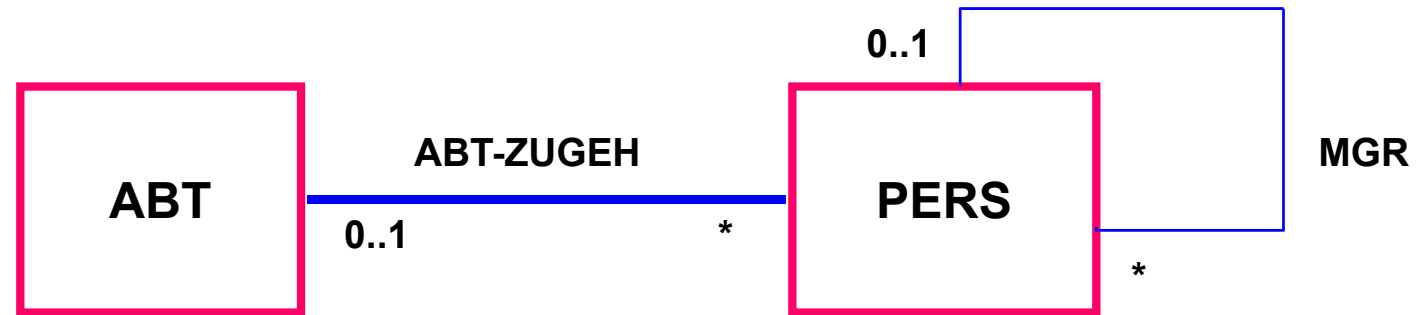
- Domains können in SQL-92 nur bzgl. Standard-Datentypen (nicht über andere Domains) definiert werden
- echte benutzerdefinierten Datentypen und strenge Typprüfung erst ab SQL:1999-Standard

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 = „Session“ eines Benutzers;
Inhalt kann beim Commit zurückgesetzt werden
- Angaben / Integritätsbedingungen bei
Attributdefinition (column definition):
 - 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)

Beispiel für Large Objects

```
CREATE TABLE Pers (PNR      INTEGER,  
                  Name     VARCHAR (40),  
                  Vollzeit  BOOLEAN,  
                  Lebenslauf CLOB (75K),  
                  Unterschrift BLOB (1M),  
                  Bild       BLOB (12M))
```

■ unterstützte Operationen

- Suchen und Ersetzen von Werten (bzw. partiellen Werten)
- LIKE-Prädikate, **CONTAINS**, **POSITION**, **SIMILAR TO** „*SQL (1999 / 2003)*“
- Konkatination ||, **SUBSTRING**, **LENGTH**, **IS [NOT] NULL ...**

Bsp.:
SELECT Name FROM Pers
WHERE CONTAINS (
AND POSITION (

■ einige Operationen sind auf LOBs nicht möglich

- Schlüsselbedingung
- Kleiner/Größer-Vergleiche
- Sortierung (**ORDER BY**, **GROUP BY**)



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 von Drop, wenn Attribut in einer Sicht oder einer Integritätsbedingung (Check) referenziert wird
- *CASCADE*: Folgelöschung aller Sichten / 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 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]
DROP VIEW view [RESTRICT | CASCADE]
```

- Beispiel: Sicht auf PERS, die alle Programmierer mit einem Gehalt unter 30.000 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

■ Abbildung von Sicht-Operationen auf Basisrelationen

- Umsetzung ist für Leseoperationen meist unproblematisch

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

Sichtkonzept (4)

- Abbildungsprozess auch über mehrere Stufen durchführbar

```
CREATE VIEW V AS
SELECT A, B, C
FROM R
WHERE D>10
```

```
CREATE VIEW W AS
SELECT A, B
FROM V
WHERE C=4
```

Anfrage: SELECT A FROM W WHERE B<40

Sichtkonzept (5)

- Auch bei Views mit Aggregatfunktionen und Gruppenbildung (GROUP-BY) oft Umsetzung möglich
 - z.B. durch Übernahme der View-Query in die FROM-Klausel

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

```
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

Materialisierte Sichten

- Sonderform von Sichten mit physischer Speicherung des Anfrageergebnisses (redundante Datenspeicherung)
 - Query-Umformulierung und Ausführung auf Basisrelationen entfallen
 - ermöglicht sehr schnellen Lesezugriff
 - Notwendigkeit der Aktualisierung/Refresh (automatisch durch das DBS)
 - erhöhter Speicherbedarf
 - kein Bestandteil von SQL92, jedoch in vielen DBS verfügbar

- Beispiel (Oracle-Syntax)

```
CREATE MATERIALIZED VIEW Monatsumsatz_mv
REFRESH COMPLETE ON DEMAND
AS      SELECT Monat, SUM(Betrag)
        FROM Umsatz
        GROUP BY Monat;
```

- Refresh-Optionen: complete, fast (inkrementell) ...
- Refresh-Zeitpunkte: on demand, on commit, never ...

Zusammenfassung

■ Datendefinition:

- CREATE / DROP TABLE, VIEW, ...;
- SQL-92: nur einfache Datentypen und einfaches Domänenmodell
- Schema-Evolution: ALTER TABLE

■ Sicht-Konzept (Views)

- Reduzierung von Komplexität
- erhöhte Datenunabhängigkeit
- Zugriffsschutz
- Einschränkungen bezüglich Änderbarkeit
- Materialisierte Sichten zur Performance-Verbesserung für Lesezugriffe