6. Datenmanipulation / -definition / -kontrolle in SQL

Datenmanipulation: INSERT, UPDATE, DELETE von Tupeln

Datendefinition

- Datentypen, Domains
- Erzeugen von Tabellen
- Ändern/Löschen von Tabellen

Sichtkonzept (Views)

Integritätsbedingungen und Trigger

- Klassifikation von Integritätsbedingungen
- Integritätsregeln / Trigger
- Einsatzformen von Triggern

Zugriffskontrolle/Autorisierung: GRANT, REVOKE

Zugriff auf Metadaten

Einfügen von Tupeln (INSERT)

```
INSERT INTO table   [ (column-commalist) ]
     { VALUES row-constr-commalist |  table-exp |  DEFAULT VALUES }
```

Satzweises Einfügen (direkte Angabe der Attributwerte)

```
Bsp.: Füge den Schauspieler Garfield mit der PNR 4711 ein
```

- alle nicht angesprochenen Attribute erhalten Nullwerte
- falls alle Werte in der richtigen Reihenfolge versorgt werden, kann Attributliste entfallen
- Integritätsbedingungen müssen erfüllt werden

mengenorientiertes Einfügen: einzufügende Tupeln werden aus einer anderen Relation mit Hilfe einer SELECT-Anweisung ausgewählt

```
Bsp.: Füge die Schauspieler aus L in die Relation TEMP ein
```

- (leere) Relation TEMP mit kompatiblen Attributen sei vorhanden
- die spezifizierte Tupelmenge wird ausgewählt und in die Zielrelation kopiert
- Die eingefügten Tupel sind unabhängig von denen, von denen sie abgeleitet/kopiert wurden.

Ändern von Tupeln (UPDATE)

```
searched-update ::= UPDATE table SET update-assignment-commalist
                        [WHERE cond-exp]
update-assignment ::= column = {scalar-exp | DEFAULT | NULL }
```

Gib den Schauspielern, die am Schauspielhaus spielen, eine Gehaltserhöhung von 2% (Attribute GEHALT und THEATER seien in SCHAUSPIELER).

Erhöhe das Gehalt der Schauspieler des Schauspielhauses um 2%, das der Schauspieler der Oper um 2.5%.

Löschen von Tupeln (DELETE)

```
searched-delete ::= DELETE FROM table [WHERE cond-exp]
```

Der Aufbau der WHERE-Klausel entspricht dem in der SELECT-Anweisung.

Lösche den Schauspieler mit der PNR 4711

Lösche alle Schauspieler, die nie gespielt haben.

Datendefinition in SQL

SQL-Umgebung (Environment) besteht aus:

- Katalogen
- 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`

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 möglich

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:

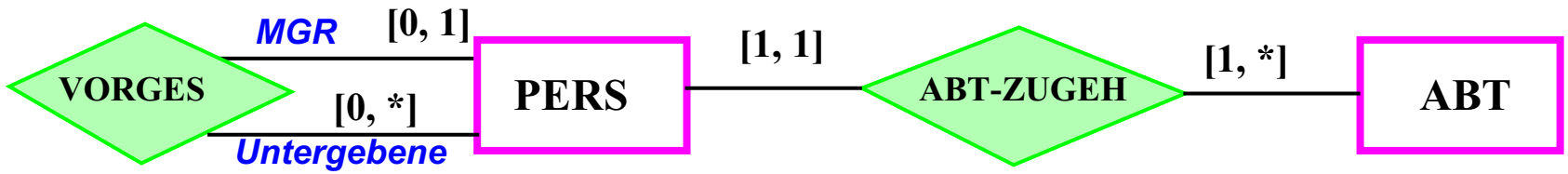
- LOCAL: Lebensdauer auf erzeugende Transaktion begrenzt
- GLOBAL: Lebensdauer auf "Session" eines Benutzers begrenzt; Inhalt kann beim Commit zurückgesetzt werden

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.

- 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
```

- Wenn das Attribut das einzige der Relation ist, wird die Operation zurückgewiesen
- RESTRICT führt zur Rückweisung der Operation, wenn das 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}
```

Falls Objekte (Relationen, Sichten, ...) nicht mehr benötigt werden, können sie 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:

```
DROP   TABLE   PERS RESTRICT
```

PersConstraint sei definiert auf PERS:

```
1. ALTER TABLE PERS DROP CONSTRAINT  PersConstraint  CASCADE
2. 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
```

Sicht kann wie eine Relation behandelt werden (Sichten auf Sichten sind möglich)

Vorteile:

- Erhöhung der Benutzerfreundlichkeit
- erhöhte Datenunabhängigkeit
- Datenschutz / Zugriffskontrolle

Sichtkonzept (2)

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 des SQL:2003 (Draft August 2002), Empfehlung für nächsten Standard, jedoch in vielen DBS verfügbar (CREATE MATERIALIZED VIEW ...)

Sichtkonzept (3)

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 = "K55"
```

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
```

Einschränkungen der Abbildungsmächtigkeit

- keine Schachtelung von Aggregatfunktionen und Gruppenbildung (GROUP-BY)
- keine Aggregatfunktionen in WHERE-Klausel möglich

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

Sichtkonzept (4)

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)

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

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 ("system enforced integrity") 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

- zunächst schwache Unterstützung in SQL (z.B. NOT NULL, UNIQUE)
- erst seit SQL92 umfangreiche Spezifikationsmöglichkeiten (relationale Invarianten, benutzer-definierte Integritätsbedingungen)

Klassifikation von Integritätsbedingungen

modellinhärente vs. anwendungsspezifische Bedingungen

- modellinhärente Integritätsbedingungen, z.B. Relationale Invarianten (Primärschlüsselbedingung, referentielle Integrität) und Wertebereichsbeschränkung für Attribute

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

Zeitpunkt der Überprüfbarkeit

- unverzögert (sofort bei Änderungsoperation)
- verzögert (am Transaktionsende)

Art der Überprüfbarkeit

- Zustandsbedingungen (statische Integritätsbedingungen)
- dynamische Integritätsbedingungen: Übergangsbedingungen oder temporale Bedingungen

Beispiele dynamischer Integritätsbedingungen:

- Übergang von FAM-STAND von 'ledig' nach 'geschieden' ist unzulässig
- Gehalt darf nicht kleiner werden
- temporale Bedingung: 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

```
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 IB zu erreichen

- SQL92: deklarative Festlegung referentieller Folgeaktionen (CASCADE, SET NULL, ...)
- SQL99: Trigger

Trigger: Festlegung von Folgeaktionen für Änderungsoperationen INSERT, UPDATE oder DELETE

Beispiel: Wartung der referentiellen Integrität

deklarativ:

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

Trigger-Lösung:

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

Trigger wesentlicher Mechanismus von ***aktiven Datenbanksystemen***

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)

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

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>
```

Merkmale

- 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
  - Ausführung von Trigger-Bedingung und -Aktion kann für jedes betroffene Tupel einzeln erfolgen (FOR EACH ROW) oder nur einmal für die auslösende Anweisung (FOR EACH STATEMENT)

Trigger (3)

weiteres Beispiel: Wartung einer materialisierten Sicht ARME_PROGRAMMIERER

CREATE TRIGGER

Probleme von Triggern

- teilweise prozedurale Semantik (Zeitpunkte, Verwendung alter/neuer Werte, Aktionsteil im Detail festzulegen)
- Trigger i.a. 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, ...)

Verallgemeinerung durch sogenannte ECA-Regeln (Event / Condition / Action)

Zugriffskontrolle (Autorisierung)

Subjekte: Benutzer, Terminals

Objekte: Programme (Anwendungs-, Dienstprogramme), DB-Objekte (Relationen, Sichten, Attribute)

Operationen: Lesen, Ändern, Ausführen, Erzeugen, etc
Weitergabe von Zugriffsrechten

Berechtigungsmatrix

Subjekte, Benutzer	**Objekte**				
	O_1	O_2	O_3	• • •	O_n
B_1	P_1, P_2		P_3		P_i
B_2		P_1	P_2, P_3		P_1
B_3		P_2, P_3	P_2		
• • •					
B_m	P_1, P_2	P_i	P_1		P_i, P_k

Klassifikationskriterien

- zentrale Vergabe (DBA) vs. dezentrale Vergabe (Eigentümer) von Zugriffsrechten
- wertabhängige vs. wertunabhängige Objektfestlegung

Zugriffskontrolle in SQL

Sicht-Konzept: wertabhängiger Zugriffsschutz (Untermengenbildung, Verknüpfung von Relationen, Verwendung von Aggregatfunktionen)

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

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 | CASCA-
```

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* erforderlich

Zugriff auf Metadaten

jeder SQL-Katalog enthält ein INFORMATION_SCHEMA zur Beschreibung der Metadaten aller Datenbanken (Schemas) des Katalogs

INFORMATION_SCHEMA enthält vordefinierte Sichten, auf die über normale SQL-Anweisungen (lesend) zugegriffen werden kann

Folgende Sichten sind u.a. vorgesehen:

DB-Objekte

```
SCHEMATA
DOMAINS
TABLES
VIEWS
COLUMNS
```

Constraints

```
TABLE_CONSTRAINTS
REFERENTIAL_CONSTRAINTS
DOMAIN_CONSTRAINTS
CHECK_CONSTRAINTS
ASSERTIONS
```

Abhängigkeiten

```
COLUMN_DOMAIN_USAGE
VIEW_TABLE_USAGE
VIEW_COLUMN_USAGE
CONSTRAINT_TABLE_USAG
CONSTRAINT_COLUMN_USAGE
```

Zugriffsberechtigungen

```
TABLE_PRIVILEGES
COLUMN_PRIVILEGES
USAGE_PRIVILEGES
```

Zusammenfassung

Datenmanipulation: INSERT, UPDATE, DELETE von Tupeln

Datendefinition: CREATE / DROP TABLE, VIEW, DOMAIN, SCHEMA; ALTER TABLE

Sicht-Konzept (Views)

- Korrespondenz zum externen Schema bei ANSI/SPARC
- Reduzierung von Komplexität, erhöhte Datenunabhängigkeit, Zugriffsschutz
- Einschränkungen bezüglich Änderbarkeit

Integritätsbedingungen

- Klassifikation: modellinhärent/anwendungsspezifisch, statisch/dynamisch, unverzögert/verzögert, Reichweite
- Unterstützung in SQL (CREATE TABLE, CREATE DOMAIN, ASSERTIONS, Trigger)

Trigger

- automatische Reaktion bei DB-Änderungen (-> "aktive DBS")
- zahlreiche Anwendungsmöglichkeiten: Integritätskontrolle, materialisierte Sichten, ...

dezentrales Autorisierungskonzept zur Zugriffskontrolle (GRANT, REVOKE)

Standardisierte SQL-Sichten für Metadaten (INFORMATION SCHEMA)