

7. Datendefinition in SQL

- Definition von Tabellen
 - Schema, Datentypen, Domains
 - Erzeugen von Tabellen (CREATE TABLE)
 - Einsatz von Large Objects (BLOB, CLOB)
- Sichtkonzept (Views)
 - CREATE VIEW / DROP VIEW
 - Problemfälle (nicht änderbare Views)
 - materialisierte Sichten
- Schemaevolution
 - Ändern/Löschen von Tabellen
- Integritätsbedingungen
 - Klassifikation von Integritätsbedingungen
 - Integritätsbedingungen in SQL
- Integritätsregeln / Trigger



Schemadefinition in SQL

- SQL-Umgebung (Environment) besteht aus
 - Katalogen: pro Datenbank ein Schema
 - Benutzerinformationen
 - **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  FLIGHT-DB
              AUTHORIZATION LH_DBA1
...
```



Basis-Datentypen (SQL92)

■ 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 *)

Weitere SQL-Datentypen

■ Boolean (SQL:1999)

■ 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 DEPTNO AS CHAR (6)
CREATE DOMAIN AGE AS INT DEFAULT NULL
      CONSTRAINT ACheck CHECK (VALUE=NULL OR VALUE > 17)
```

■ 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



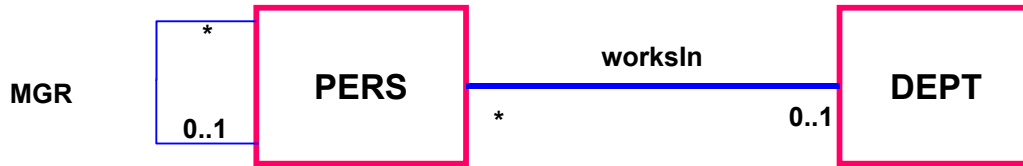
Erzeugung von Basisrelationen

```
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

```
(PNO      INT                PRIMARY KEY,
OCCUPATION VARCHA(50)        VARCHA(50),
PNAME     VARCHA(50)         NOT NULL,
PAGE      AGE, (* siehe Domain-Definition *)
MGR       INT                REFERENCES PERS,
DNO       DEPTNR (* Domain-Definition *)
SALARY    DEC(7) DEFAULT 0 CHECK (VALUE < 120000)
FOREIGN KEY (DNO) REFERENCES DEPT )
```

CREATE TABLE DEPT

```
(DNO      DEPTNR             PRIMARY KEY,
DNAME     VARCHA(50)        NOT NULL)
```

Beispiel für Large Objects

```
CREATE TABLE Pers (PNO      INTEGER,
                   PName    VARCHA(40),
                   Fulltime  BOOLEAN,
                   CV        CLOB(75K),
                   Signature  BLOB(1M),
                   Photo     BLOB(12M))
```

■ unterstützte Operationen

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

```
SELECT Name FROM Pers
WHERE CONTAINS (CV, „Daten“ AND „UML“)
AND POSITION (CV, „SQL“) < 500
```

■ nicht möglich auf LOBs sind

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

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 40.000 umfasst

CREATE VIEW

```
POOR_PROGRAMMER (PNO, NAME, OCCUP, SALARY, DNO) AS
    SELECT      PNO, PNAME, OCCUPATION, SALARY, DNO
    FROM        PERS
    WHERE       OCCUPATION = 'Programmer' AND Salary < 40 000
```



Sichtkonzept (2)

- Sicht kann wie eine Relation behandelt werden
 - Anfragen / Anwendungsprogramme auf Sichten
 - Sichten auf Sichten sind möglich
- Vorteile:
 - Erhöhung der Benutzerfreundlichkeit
 - Datenschutz / Zugriffskontrolle
 - erhöhte Datenunabhängigkeit
 - verbesserte Schemaevolution
 - Attributumbenennung
 - Änderung Datentyp für Attribut
 - neues Attribut
 - Löschen von Attribut



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, SALARY
FROM POOR_PROGRAMMER
WHERE DNO = 'K05'
```



```
SELECT PNAME, SALARY
FROM PERS
WHERE DNO = 'K05'
AND OCCUPATION = 'Programmer'
AND SALARY < 40 000
```

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

→ SELECT A FROM V WHERE B < 40 AND

→ SELECT A FROM R WHERE B < 40 AND

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 DEPTINFO (DNO, SALSUM) AS
  SELECT DNO, SUM(SALARY)
  FROM PERS
  GROUP BY DNO
```

```
SELECT AVG (SALSUM) FROM DEPTINFO
```

naive (falsche) Umsetzung:

```
SELECT AVG (SUM(SALARY)) FROM PERS GROUP BY DNO
```

korrekte Lösungsmöglichkeit:

```
SELECT AVG (S.SALSUM)
FROM (SELECT DNO, SUM(SALARY) AS SALSUM
      FROM PERS GROUP BY DNO) AS S
```

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 (OCCUP, SALARY) AS
  SELECT OCCUPATION, SALARY 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
 - Nutzung auch als Daten-Snapshot /Kopie
 - Notwendigkeit der Aktualisierung/Refresh (automatisch durch das DBS)
 - erhöhter Speicherbedarf
 - nicht standardisiert, jedoch in vielen DBS verfügbar (Oracle, DB2, PostgreSQL...)

■ Beispiel (Oracle-Syntax)

```
CREATE MATERIALIZED VIEW MonthRevenue_mv
REFRESH COMPLETE ON DEMAND
AS      SELECT Month, SUM(Amount)
        FROM SALES
        GROUP BY Month;
```

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



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}}
```

■ Schemaevolution: 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 SVNO INT UNIQUE
ALTER TABLE PERS DROP SALARY 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

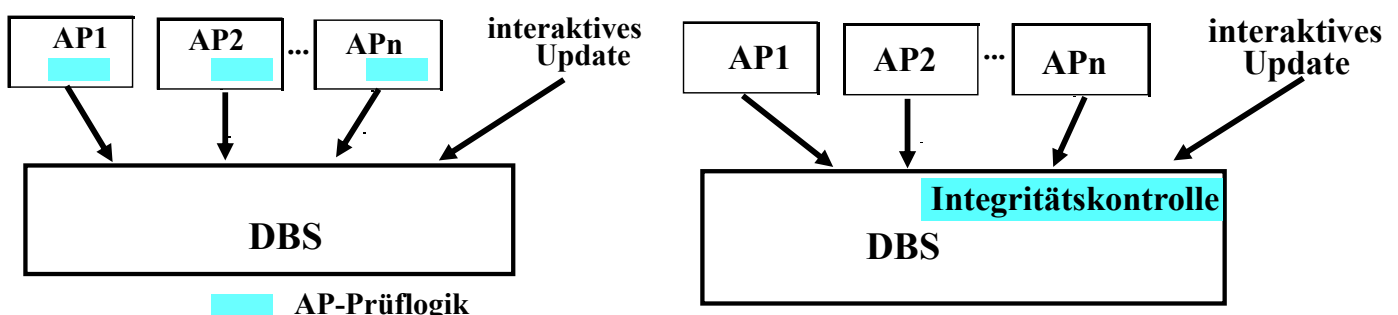
■ Beispiele:

```
DROP DOMAIN AGE CASCADE
```

```
DROP TABLE PERS RESTRICT
```

Semantische Integritätskontrolle

- Logische DB-Konsistenz: Überwachung von semantischen Integritätsbedingungen durch Anwendungen oder 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	BIRTHYEAR ist numerisch, 4-stellig
Satzausprägung	DEPT.SALARYSUM < DEPT.YEARBUDGET
Satztyp	PNO ist eindeutig
mehrere Satztypen	DEPT. SALARYSUM ist Summe aller Personengehälter in PERS

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 '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 ...
PNO 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 ....  
  BIRTHYEAR INT  
  CHECK (VALUE BETWEEN 1900 AND 2025)  
CREATE TABLE ABT .....  
  CHECK (SALARYSUM < YEARBUDGET)
```

Anweisung CREATE ASSERTION

```
CREATE ASSERTION A1  
  CHECK (NOT EXISTS  
    (SELECT * FROM DEPT D  
      WHERE SALARYSUM <>  
        (SELECT SUM (P.SALARY) FROM PERS P  
          WHERE P.DNO = D.DNO)))  
  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
(PNO INT PRIMARY KEY,
 DNO INT FOREIGN KEY
 REFERENCES DEPT
 ON DELETE CASCADE
 ... );
```

- **durch Trigger**

```
CREATE TRIGGER DELETE-EMPLOYEE
BEFORE DELETE ON DEPT
REFERENCING OLD AS D
DELETE FROM PERS P
WHERE P.DNO = D.DNO;
```

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
 - 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 (*Gehalt darf nicht kleiner werden*):

```
CREATE TRIGGER SALARY-CHECK  
AFTER UPDATE OF SALARY ON PERS  
REFERENCING OLD SALARY1, NEW SALARY2  
WHEN (SALARY2 < SALARY1) ROLLBACK;
```

- Wartung einer materialisierten Sicht POOR_PROGR_MV

```
CREATE TRIGGER PP-INSERT  
AFTER INSERT ON PERS  
FOR EACH ROW  
REFERENCING NEW AS N  
WHEN N.SALARY = „Programmer“ AND N.GEHALT < 40000  
INSERT INTO POOR_PROGR_MV (PNO, NAME, OCCUP, SALARY)  
VALUES (N.PNO, N.PNAME, N.OCCUPATION, N.SALARY)
```

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)
- i. a. keine verzögerte Auswertung von Triggern
- Gefahr zyklischer, nicht-terminierender Aktivierungen
- Korrektheit des DB-/Trigger-Entwurfes (Regelabhängigkeiten, parallele Regelausführung, ...)

dennoch sind Trigger mächtiges und wertvolles Konstrukt

- „aktive“ DBS
- DBS-interne Nutzungsmöglichkeiten, z.B. zur Realisierung von Integritätskontrolle und Aktualisierung von materialisierten Sichten / Replikaten

Zusammenfassung

■ Datendefinition:

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

■ Schema-Evolution: ALTER TABLE

- DROP zum Entfernen von Tabellen / Sichten etc.

■ Sicht-Konzept (Views)

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

■ Integritätsbedingungen

- Klassifikation gemäß 4 Kriterien
- umfassende Unterstützung in SQL

■ Trigger

- automatische Reaktion bei DB-Änderungen (-> „aktive DBS“)
- zahlreiche Nutzungsmöglichkeiten: Integritätskontrolle, mat. Sichten ...
-