Kapitel 6 Relationale DB-Sprache SQL

SEQUEL: Structured English Query Language, 70er Jahre
SQL: System R, SQL/DS, TransBase, Oracle ...
ANSI Standards 1, 2, 3

6.1 Daten-Definitionssprache DDL

Domänen: Grundtypen, alle vordefiniert, z.B.
INTEGER ~ **integer**
NUMERIC (p,s) p: precision, s: scale (nach,) etc.

Problem: Domänen tragen keine Semantik, z.B.
Tel# **integer**
Zi# **integer**
Join-Bedingung S.Tel#= R.Zi# wäre syntaktisch erlaubt.

Data_Type::=

TINYINT 1 B
| SMALLINT 2 B
| INTEGER 4 B
| NUMERIC [(Precision [,Scale])]
| FLOAT
| DOUBLE
| REAL
| CHAR [(Precision)]
| CHAR (*)
| BINCHAR [(Precision)]
| BINCHAR (*)
| STRING

| BOOL
| DATETIME Range_Qualifier
| TIMESPAN Range_Qualifier
| BLOB

Precision ::= Integer_Literal
Scale ::= Integer_Literal
Range_Qualifier

Vereinbarung von Relationen: TABLE

create table TEILE
(T# **string**
Tname **string**
Farbe **string**
Gewicht **integer**)
key is T#

syntaktisch gleich, semantisch verträglich?

Erweiterung durch interne Identifikatoren, wie z.B. TupelId oder RowId (Oracle), die für Benutzer nur bedingt sichtbar sind

drop table TEILE

Segmente:
„logische“ Speicherbereiche, Zusammenfassung von Relationen und Hilfsstrukturen für Zugriffs- und Ladezwecke.

View-Begriff:

Funktionsdefinition über Relationen, Ergebnis ist „berechnete“ Relation, nicht gespeichert, nur bedingt änderbar.

```
create   table  quotations
   (suppno        integer  not   null,
   partno         integer  not   null,
   price          numeric  (6,2),
   deliverytime   integer,
   qonorder       numeric  (4))
          key is  suppno, partno
```



```
create view cashpermonth (supplier, part, total)
   as
   select  qx.suppno, qx.partno,
           qx.price ∗ qx.qonorder
   from    quotations qx
   where   qx.deliverytime < 32 and
           qx. qonorder > 0

drop view  cashpermonth
```

Themen: materialisierte views
view – updates

Indexe:

{(Schlüssel, Tupel-Id)}
{(Attr.Wert, (Tupel-Id)$^+$}

B-Bäume, B*-Bäume, Präfix-B-Bäume über 1 oder mehreren Attributen, hohe Verzweigungsgrade von 100 bis 1000

Primärindex : über Schlüsselattributen

Sekundärindex: über sonstigen Attributen

$\Rightarrow$ pro Attributwert mehrere Tupel-Id

Ziel: Beschleunigung der Suche von

O(|R|) auf **O (log |R|)**

Beispiel: R hat 1.000.000 Tupel, 50 Tupel/Seite, 20.000 Seiten

O(|R|) = 20.000 **O ($\log_{100}$ |R|)** = 3


```
create index     TEILE–Namen
   on            TEILE (Tname)

create unique index   TEILE–Nummern
   on                 TEILE (T#)
```

Hinweis: TEILE–Nummern ist Primärindex, bei manchen DBS überflüssig, z.B. TransBase

TEILE–Namen ist Sekundärindex, mehrere Einträge pro Attr.Wert

create index TEILE-Namen-Farben
on TEILE (Tname, Farbe)

i.e. Sekundäindex über mehrere Attribute, z.B. (Schraube, blau, ()*)

drop index TEILE-Namen

drop table TEILE

zieht automatisch drop für Indexe und abhängige views nach sich, vorher Warnung? Autorisierung?

6.2 Systemtabellen

DB ist ***selbstbeschreibend***, d.h.

- enthält eigene Struktur (Schema)
- nach festem Metaschema (DBS-spezifisch)

Katalog: DB enthält Beschreibung der
Tabellen, Views, Indexe und Attribute etc.,
Zugriffsrechte von Benutzern auf Tabellen;
Beschreibung selbst in Tabellenform = *System – Tabellen*

⇒ DDL selbst mit SQL implementiert, und mit SQL abfragbar, z.B. Attribute und Typ einer Relation R?

SYSTABLE enthält 1 Tupel pro Relation bzw. View:

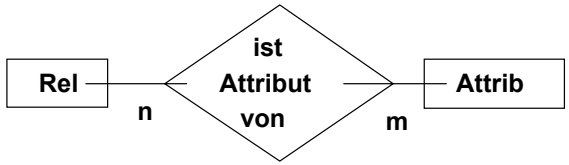
```
createtable  SYSTABLE
          ( tname     string  not   null ,
            ttype     string  not   null ,
            segno     integer
            colno     integer   not   null)
                  key  is   tname
```

tname : Name von Rel. oder View
ttype : „R“ Relation
„V“ View
segno : interne Segmentnummer,
null bei Views
colno : Anzahl der Attribute, i.e. Stelligkeit der Relation

⇒ d.h. SYSTABLE ist fest vorgegebene Metatabelle, enthält Strukturbeschreibung für alle anderen Tabellen **und** sich selbst, d.h. initialisiert mit:

(SYSTABLE, „R“, ?, 4)

Relation und Attribute:



Im Prinzip kann Attribut mit derselben Bezeichnung in verschiedenen Relationen unterschiedliche Typen haben (Schlechter Entwurf, aber nicht verboten)

⇒Typ etc. muß Teil der „ist-Attribut von“ Beziehung sein

ist-Attribut-von ≡ SYSCOLUMN Relation,
enthält 1 Tupel pro (Rel, Attr)

create table SYSCOLUMN
(tname **string not null** ,
cname **string not null** ,
ctype **string not null** ,
cpos **integer not null** ,
ckey **integer** {Pos. in Schlüssel oder
0 oder null (view)}
notnull **string** {„Y“ oder „N“ oder null für view}
indexno **integer** {interne Index #})
key is tname, cname

Beispiel für interne Integritätsbed:
Anzahl Einträge mit tname in SYSCOLUMN = colno in SYSTABLE

Hinweis:
SYSCOLUMN enthält Tupel zur Selbstbeschreibung:

SYSCOLUMN	tname	string	1	1	Y	0
SYSCOLUMN	cname	string	2	2	Y	0
SYSCOLUMN	cpos	integer	4	0	Y	0
SYSCOLUMN	ckey	integer	5	0	N	0

z.B. query für Attribute von TEILE:

select cname, ctype
from SYSCOLUMN
where tname = „TEILE“
order by cpos

Query für Schlüsselattribute von TEILE?
Parametrisierung der Query für Rel?

View Beschreibungen:

create view BLAUE-TEILE
Teil-Name, Gewicht, Teil-Nr.
as
select Tname, Gewicht, T#
from TEILE
where Farbe = ´blau´

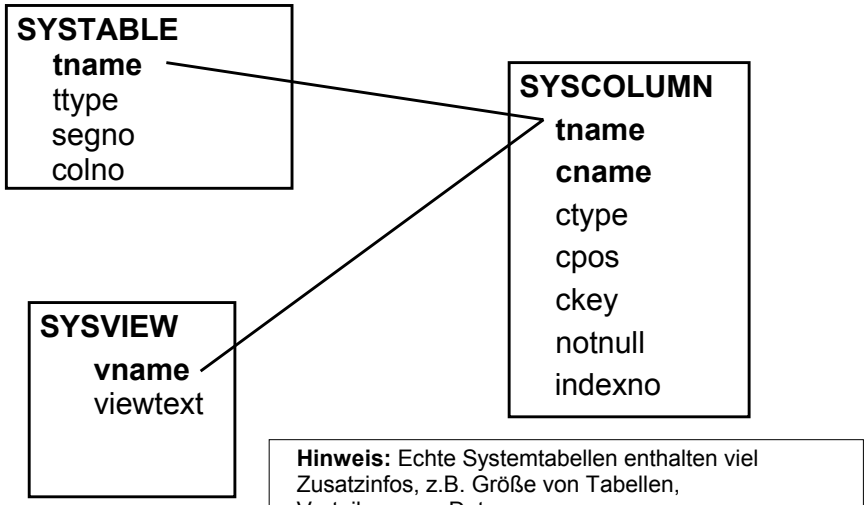
Kopfleiste steht in SYSTABLE,
Def. von Body steht in SYSVIEW

```
create   table  SYSVIEW
    (vname      string    not    null ,
    viewtext    string    not    null)
       key is   vname
```

vname: Name der View

viewtext: Query-Teil der View-Definition in übersetzer und optimierter Form (als optimierter Operator – Baum). Wird ausgewertet, falls Wert der View benötigt wird.

Hinweis: Attribute der View stehen unter vname in Relation SYSCOLUMN



Hinweis: Echte Systemtabellen enthalten viel Zusatzinfos, z.B. Größe von Tabellen, Verteilung von Daten, minimale und maximale Werte, etc.