Ludwig Maximilians Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Skript zur Vorlesung

Datenbanksysteme I

Wintersemester 2012/2013

Kapitel 12: Objekt-relationale Erweiterungen

Vorlesung: Christian Böhm
Übungen: Sebastian Goebl, Nina Hubig

http://www.dbs.ifi.lmu.de/cms/Datenbanksysteme_I

Grenzen des relationalen Modells

- Nichtstandard Anwendungen
  - CAD / CAM / CIM
  - Geographie und Kartographie
  - Medizin und Biologie
  - Multimedia
- komplexe, unterschiedlich strukturierte Objekte
- Daten mit sehr großen Attributswerten (z.B. Multimedia-Inhalte)
- oft komplexe Integritätsbedingungen (z.B. aus der Geographie)
- häufig sehr lange Transaktionen auf wenigen Objekten (z.B. CAD)

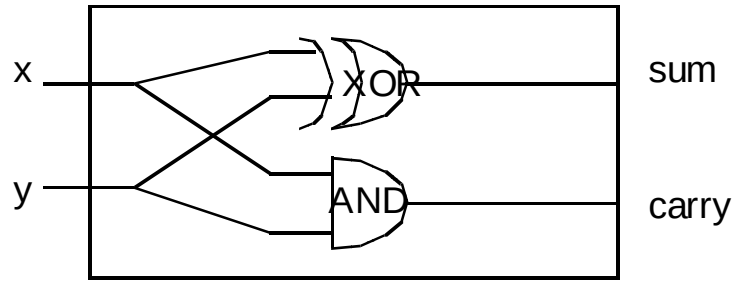
Grenzen des relationalen Modells

- Beispiel-Anwendung 1: Computer Aided Design von VLSI Chips
  - Entwurf eines 1-bit Addierers mit 3 Inputs (x , y und cin) und 2 Outputs (z und cout)
    - Funktionelle Repräsentation definiert für jeden Input den gewünschten Output:

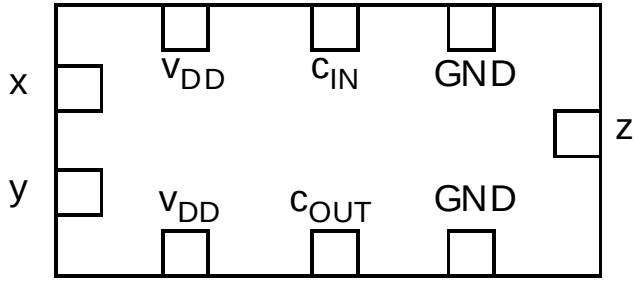
x	y	cin	z	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
…	…	…	…	…

Grenzen des relationalen Modells

- Logische Repräsentation beschreibt die logischen Komponenten des Addierers und ihre logischen Beziehungen. Logische Repräsentation eines 1-bit Halb-Addierer (Komponente eines 1bit-Addierers)



- Geometrische Repräsentation beschreibt das physische Layout des Chips, d.h. die räumliche Lage und Ausdehnung der einzelnen Komponenten, z.B. für den 1-bit Addierer



Grenzen des relationalen Modells

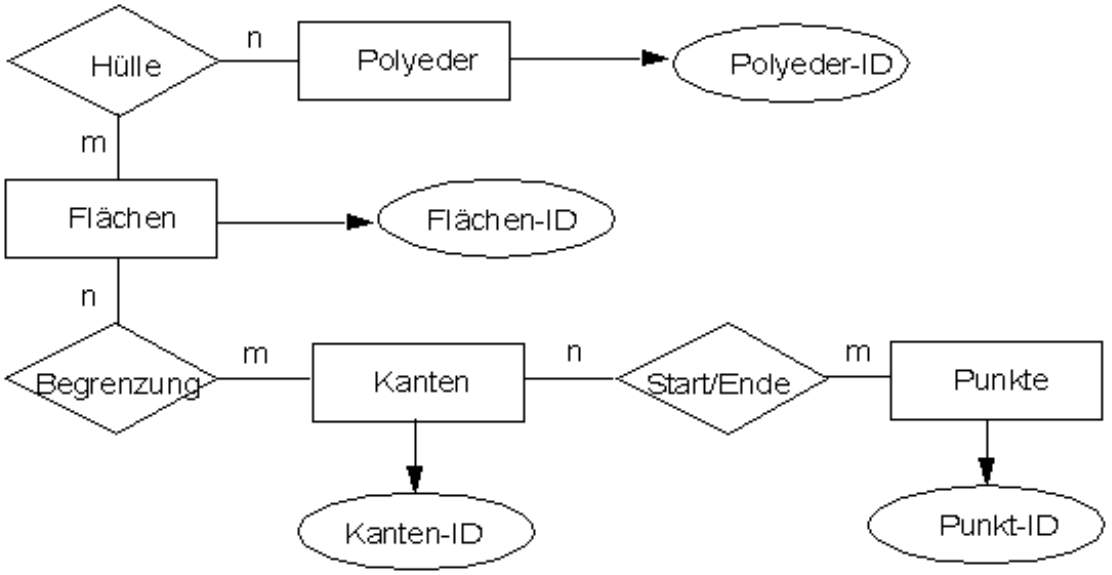
- Die Verbindungen der Komponenten werden in einem weiteren Schritt definiert. Es sind zahlreiche Konsistenzbedingungen einzuhalten, z.B. dürfen die Bestandteile des Layouts nicht beliebig, sondern nur in Vielfachen von 90 Grad rotiert werden.

– Anforderungen an das DBS

- Verwaltung komplexer Objekte mit Komponenten, die wiederum Komponenten besitzen
- Verwaltung verschiedener Repräsentationen desselben Objekts die bei Updates alle miteinander zu ändern sind
- Einhaltung von Konsistenzbedingungen aus der Anwendung
- Wiederverwendung von vorhandenen Basis-Bausteinen, die nicht immer wieder neu entworfen werden sollen

Grenzen des relationalen Modells

- Beispiel-Anwendung 2: Modellierung von 3D-Polyedern
  - Polyeder können durch ihre Begrenzungen (*Boundary Representation*) beschrieben werden
  - ER-Diagramm:



Grenzen des relationalen Modells

– Übertragung ins relationale Modell:

Sowohl für die vier Entity Sets als auch für die drei Beziehungen jeweils eine Relation

Polyeder§

Poyeder-ID§	Volumen§	. . .§
. . .§	. . .§	. . .§
poly-5§	1000.00§	. . .§
. . .§	. . .§	. . .§

§

Flächen

Flächen-ID§	Umfang§	. . .§
f1§	40.0§	. . .§
f2§	40.0§	. . .§
. . .§	. . .§	. . .§
f6§	40.0§	. . .§

§

Kanten

Kanten-ID§	Länge§	. . .§
k1§	10.0§	. . .§
k2§	10.0§	. . .§
. . .§	. . .§	. . .§

§ Punkte

Punkt-ID§	X§	Y§	Z§
p1§	0.0§	0.0§	0.0§
p2§	10.0§	0.0§	0.0§
. . .§	. . .§	. . .§	. . .§

Hülle§

Poyeder-ID§	Flächen-ID§
poly-5§	f1§
poly-5§	f2§
. . .§	. . .§
poly-5§	f6§

Begrenzung§

Flächen-ID§	Kanten-ID§
. . .§	. . .§
f1§	k1§
f1§	k2§
f1§	k3§
f1§	k4§
. . .§	. . .§
f6§	. . .§

Start/Ende§

Kanten-ID§	Punkt-ID§
. . .§	. . .§
k1§	p1§
k1§	p2§
k2§	p2§
k2§	p3§
k12§	. . .§
. . .§	. . .§

Grenzen des relationalen Modells

- Schwächen
  1. Aufspaltung der logischen Objekte:

     Ein logisches Objekt der Anwendung muss in mehrere Tupel verschiedener Relationen aufgespalten werden
     - Das macht alle Operationen auf den logischen Objekten komplex
     - Bei Updates des Objekts müssen alle zugehörigen Tupel aktualisiert werden
     - Zur Bearbeitung von Anfragen müssen die Objekte durch teure Joins aus den einzelnen Tupeln zusammengesetzt werden.

Grenzen des relationalen Modells

- Beispiel
  - » Das Polyeder poly-5 wird in 1 + 6 + 12 + 8 + 6 + 24 + 24 = 81 Tupel von 7 verschiedenen Relationen aufgespalten
  - » Die Operation “rotiere poly-5 um 90 Grad” muss die 8 Punkte des Polyeders suchen und ihre Koordinaten geeignet ändern
  - » Die Anfrage “finde die X, Y, Z Koordinaten aller Punkte von Polyedern, deren Volumen größer als 10 ist” lässt sich in SQL formulieren als:

select X, Y, Z **from** Punkte **where** Punkt-ID IN
select Punkt-ID **from** Start/Ende **where** Kanten-ID IN
select Kanten-ID **from** Begrenzung **where** Flächen-ID IN
select Flächen-ID **from** Hülle **where** Polyeder-ID IN
select Polyeder-ID **from** Polyeder **where** Volumen > 10.0

  - » 4 Joins nötig
  - » Solche Anfragen sind sowohl aufwendig in der Formulierung als auch sehr teuer in der Anfragebearbeitung

Grenzen des relationalen Modells

2. Künstliche Schlüssel
   - Um den Zusammenhang zwischen den einzelnen Tupeln eines logischen Objekts repräsentieren zu können, werden künstliche Schlüssel-Attribute eingeführt (z.B. Flächen-ID, Kanten-ID und Punkt-ID, die in Hülle, Begrenzung und Start/Ende als Fremdschlüssel verwendet werden)
   - Folgende Probleme treten dabei auf:
     - **Eindeutige Schlüssel:** Die künstlichen Schlüssel müssen eindeutig sein, z.B. muss die Flächen-ID für alle Flächen aller Polygone eindeutig sein und diese Eindeutigkeit muss vom Anwendungsprogrammierer / Anwender garantiert werden
     - **Existenz (Referentielle Integrität):** Für alle Werte, die als Fremdschlüssel verwendet werden, muss in der entsprechenden Relation ein Tupel mit diesem Schlüsselwert existieren. Neuere relationale DBMS garantieren bereits die referentielle Integrität.

Grenzen des relationalen Modells

3. Keine Beziehungen zwischen den Relationen
   - Beziehungen zwischen Objekten sollen modelliert werden können (z.B. Vererbung, um gemeinsamen Eigenschaften wieder verwenden zu können)
   - *Beispiel:* Rechtecke, Kreise, Polygone sind Spezialfälle von Flächen

4. Keine Modellierung des Verhaltens
   - Relationales Modell behandelt nur die Struktur, nicht das Verhalten von Objekten, d.h. es werden nur Standard-Operationen zur Anfrage und zum Update von einzelnen Tupeln angeboten, aber keine anwendungs- bzw. objektspezifischen Operationen.

Grenzen des relationalen Modells

- Objekt-Semantik ist in Anwendungsprogrammen versteckt
  - » mehrere Anwendungsprogramme für dieselbe Operation
  - » möglicherweise verschiedene Semantiken für dieselbe Operation
  - » *Beispiel*

    Für Polyeder sind u.a. folgende Operationen zu definieren:

    skaliere (Faktor)
    bewege (X, Y, Z)
    rotiere (Achse, Winkel)
    …

Grenzen des relationalen Modells

5. Problematische Schnittstelle DML-Programmiersprache
   - Eine relationale DML, wie z.B. SQL, liefert Mengen von Tupeln als Antworten. Eine Programmiersprache, wie z.B. Java, kann jedoch nur ein Tupel zur Zeit verarbeiten. Abhilfe schafft das sog. Cursor-Konzept um die Antworten einzeln zu verarbeiten.
   - Ein Cursor besitzt jedoch folgende für Non-Standard Anwendungen gravierenden Schwächen:
     - » nur sequentieller Zugriff auf die Tupel in der gegebenen Reihenfolge (nur genau einmal); bei interaktiven Anwendungen ist jedoch meist wahlfreier Zugriff erforderlich
     - » i.a. keine Updates möglich, z.B. wenn ein Cursor mit einer Anfrage assoziiert ist, die einen Join enthält, da es unentscheidbar ist, wie ein Update sich auf den zugrunde liegenden Relationen auswirkt.

Grenzen des relationalen Modells

» Beispiel

Auf dem Cursor

```
exec sql declare Kanten/Punkte cursor for
select s.Kanten-ID, p.Punkt-ID, p.X, p.Y, p.Z
from Start/Ende s, Punkte p
where s.Punkt-ID = p.Punkt-ID
```

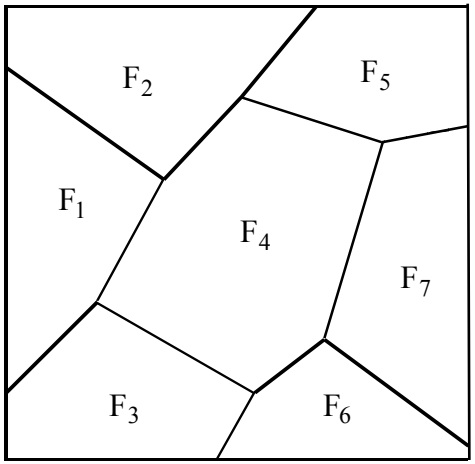
können keine Updates ausgeführt werden. Der Join muss im Anwendungsprogramm explizit programmiert werden:

```
exec sql declare Kante/Punkt cursor for
        select * from Start/Ende;
exec sql open Kante/Punkt;
while (true) {
        exec sql fetch Kante/Punkt into :Kanten#, :Punkt#;
        exec sql select X, Y, Z into :Xvar, :Yvar, :Zvar
                from Punkte where Punkt-ID = :Punkt#;
        . . .
        exec sql update Punkte set . . .;
}
```

Grenzen des relationalen Modells

- Beispiel-Anwendung 3: Darstellung geometrischer Objekte (hier Parzellen) in normalisierten Relationen:

  Eine redundanzfreie Repräsentation der Parzellen erfordert die Verteilung der Informationen auf drei Relationen: „Parzellen“, „Kanten“ und „Punkte“:



Parzellen	
FNr	KNr
F_1	K_1
F_1	K_2
F_1	K_3
F_1	K_4
F_4	K_2
F_4	K_5
F_4	K_6
F_4	K_7
F_4	K_8
F_4	K_9
F_7	K_7
F_7	K_{10}
F_7	K_{11}
F_7	K_{12}

Kanten		
KNr	PNr_1	PNr_2
K_1	P_1	P_2
K_2	P_2	P_3
K_3	P_3	P_4
K_4	P_4	P_1
K_5	P_2	P_5
K_6	P_5	P_6
K_7	P_6	P_7
K_8	P_7	P_8
K_9	P_8	P_3
K_{10}	P_6	P_9
K_{11}	P_9	P_{10}
K_{12}	P_{10}	P_7

Punkte		
PNr	X-Koord.	Y-Koord.
P_1	X_{P1}	Y_{P1}
P_2	X_{P2}	Y_{P2}
P_3	X_{P3}	Y_{P3}
P_4	X_{P4}	Y_{P4}
P_5	X_{P5}	Y_{P5}
P_6	X_{P6}	Y_{P6}
P_7	X_{P7}	Y_{P7}
P_8	X_{P8}	Y_{P8}
P_9	X_{P9}	Y_{P9}
P_{10}	X_{P10}	Y_{P10}

Grenzen des relationalen Modells

- Sollen jetzt (geometrische) Anfragen auf den Parzellen bearbeitet werden, so müssen die erforderlichen Informationen zunächst stets wieder zusammengesetzt werden
- *Beispiel:* Gesucht sind alle Eckpunkte der Parzelle mit Flurnummer 2

```
select   Punkte.PNr, X-Koord, Y-Koord
from     Parzellen, Kanten, Punkte
where    FNr = “F2”
         and
         Parzellen.KNr = Kanten.KNr
         and
         (Kanten.PNr1 = Punkte.PNr or Kanten.PNr2 = Punkte.PNr)
```

Grenzen des relationalen Modells

- Obwohl die Geometrie des Objektes noch gar keine Rolle gespielt hat, benötigen wir zur Bearbeitung dieser einfachen Anfrage zwei oder drei Joins.
- Ursache: Art der Datenmodellierung
  - Die Zerlegung der Objekte in einfache Komponenten (Flächen, Linien und Punkte) ergibt ein normalisiertes Datenbankschema
  - Die Informationen über ein Objekt (Parzelle) liegt aber nun über mehrere Relationen („Parzellen“, „Kanten“ und „Punkte“) verteilt

Grenzen des relationalen Modells

- Zusammenfassung
  - Information über logische Objekte auf mehrere Relationen verteilt
  - Kein Objektverhalten
  - Keine Wiederverwendung von Code
  - Verwaltung verschiedener Repräsentationen desselben Objekts die bei Updates alle miteinander zu ändern sind
  - Einhaltung von Konsistenzbedingungen aus der Anwendung
  - Abspeicherung von Objekten mit großen Attributwerten (z.B. Multimedia-Inhalte)
  - …

Vom relationalen zum objekt-relationalen Model

- Lösungsmöglichkeit 1: NF^2-Relationen (NF^2 = *non first normal form*)
  - Die erste Normalform des relationalen Modells wird umgangen, d.h. auch nicht-atomare Werte (Relationen) sind als Attribute zuzulassen
  - Beispiel: Parzellen

Parzellen				
	Kanten			
		Punkte		
FNr	KNr	PNr	X-Koord.	Y-Koord.
F_1	K_1	P_1	X_{P1}	Y_{P1}
		P_2	X_{P2}	Y_{P2}
	K_2	P_2	X_{P2}	Y_{P2}
		P_3	X_{P3}	Y_{P3}
	K_3	P_3	X_{P3}	Y_{P3}
		P_4	X_{P4}	Y_{P4}
	K_4	P_4	X_{P4}	Y_{P4}
		P_1	X_{P1}	Y_{P1}
F_4	K_2	P_2	X_{P2}	Y_{P2}
		P_3	X_{P3}	Y_{P3}
	K_5	P_2	X_{P2}	Y_{P2}
		P_5	X_{P5}	Y_{P5}
	K_6	P_5	X_{P5}	Y_{P5}
		P_6	X_{P6}	Y_{P6}
	K_7	P_6	X_{P6}	Y_{P6}
		P_7	X_{P7}	Y_{P7}
	K_8	P_7	X_{P7}	Y_{P7}
		P_8	X_{P8}	Y_{P8}
	K_9	P_8	X_{P8}	Y_{P8}
		P_3	X_{P3}	Y_{P3}
F_7	K_7	P_6	X_{P6}	Y_{P6}
		P_7	X_{P7}	Y_{P7}
	K_{10}	P_6	X_{P6}	Y_{P6}
		P_9	X_{P9}	Y_{P9}
	K_{11}	P_9	X_{P9}	Y_{P9}
		P_{10}	X_{P10}	Y_{P10}
	K_{12}	P_{10}	X_{P10}	Y_{P10}
		P_7	X_{p7}	Y_{P7}

Vom relationalen zum objekt-relationalen Model

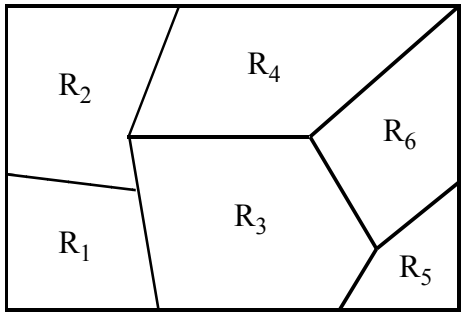
- Vorteile
  - Vermeidung von Joins durch Abspeicherung aller Informationen in einer Relation → höhere Effizienz bei der Anfragebearbeitung.
  - *Geclusterte* Abspeicherung der relationenwertigen Attribute wird möglich → weitere Effizienzsteigerung
- *Aber:*
  - Keine echte Unterstützung von Anfragen, die sich auf die Geometrie der Objekte beziehen
  - Kein Objektverhalten
  - Konsistenzbedingungen???
  - Große Attributwerte???

Vom relationalen zum objekt-relationalen Model

- Lösungsmöglichkeit 2: Objekt-relationale Erweiterung
  - Idee: Integration benutzerdefinierter Objekte und Operationen als elementare Datentypen in den Kern eines Datenbanksystems.
  - Durch Aufgreifen dieser Idee lassen sich Wertebereiche wie z.B. *Rechteck* oder *Polygon* realisieren, die eine adäquate Modellierung geometrischer Szenen ermöglichen und objekt-spezifische Methoden bereitstellen.

Vom relationalen zum objekt-relationalen Model

– Beispiel: Forstgebiete (Revier: Polygon, Förster: String, Fläche: Integer)



Forstgebiete		
Revier	Förster	Fläche (m^2)
R_1	Schmidt	3900
R_2	Behrens	4250
R_3	Schultz	6700
R_4	Schmidt	5400
R_5	Meier	1900
R_6	Schröder	4600

– Durch diese Art der Datenmodellierung werden Anfragen folgender Art möglich:

In einer Schonung definiert durch das Polygon *S* soll eine Neuanpflanzung durchgeführt werden. Welche Förster sind von der Aktion betroffen?

select Förster

from Forstgebiete

where ObjectIntersects(S, Revier)

Vom relationalen zum objekt-relationalen Model

- Innerhalb des Systems steht Datentyp *Polygon* mit dem Prädikat *ObjectIntersects* zur Verfügung, das testet, ob sich zwei Polygone schneiden

– Vorteile dieser Datenmodellierung

- Integration geometrischer Datentypen erfordert lediglich eine *Erweiterung* des bestehenden relationalen Modells sowie der zugehörigen Datenbanksprachen
- Geometrische Attribute werden als *eigenständige* Datentypen aufgefasst. Dies erleichtert den Datenbankentwurf und die Formulierung von Anfragen.
- Geometrische Datentypen (Wertebereiche und Operationen) können durch effiziente Datenstrukturen und Algorithmen im Kern eines Datenbanksystems realisiert werden

Übersicht über objekt-relationale Konzepte

- Große Objekte (LOBs = Large OBjects)
  - Speicherung von großen Attributwerten (GB-Bereich)
- Mengenwertige Attribute
  - Attribut kann Menge von Werten als Domain besitzen
  - Anfragesprache muss Schachtelung (Bildung) bzw. Entschachtelung („Flachklopfen“) von Mengen unterstützen
- Geschachtelte Relationen
  - Attributwerte können selbst wieder Relationen sein
- Benutzerdefinierte Datentypen
  - Wertebasierte, abstrakte Datentypen können nur als Komponente eines Tupels vorkommen
  - Tupeltypen (auch Objekttypen, *row types*) können als eigenständiger Datensatz in einer Relation vorkommen

Übersicht über objekt-relationale Konzepte

- Referenzen
  - Attributwert ist direkte Referenz auf Tupel/Objekte
  - Dadurch keine Fremdschlüssel mehr nötig, auch n:m-Beziehung nun ohne eigenständige Relation umsetzbar
- Objektidentität
  - Zur Identifikation von Objekten
  - Nicht veränderbar (im Ggs. zum Schlüssel)
- Pfadausdrücke
  - Referenzausdrücke erfordern Pfadausdrücke in Anfragesprache
- Vererbung
  - Generalisierung/Spezialisierung
- Operationen
  - Modellierung objektspezifischen Verhaltens

Large Objects (LOBs)

- Arten
  - CLOB (Character LargeOBjects)
    - Speicherung langer Texte
    - Verbesserter Zugriff im Vergelich zu **varchar()**
  - BLOB (Binary LargeOBjects)
    - Werden vom DBMS nicht interpretiert sondern nur gespeichert (als Bitsequenz), z.B. Videos, etc.
  - NCLOB (National Character LargeOBjects)
    - Für Texte mit Sonderzeichen (z.B. Unicode) da CLOBs nur Texte mit 1-Byte Kodierung speichert
- Wichtig:
  - Bei Verarbeitung von LOB-Daten vermeide Trasfer der Daten vom DBS zum Anwendungsprogramm (insbesondere in Client/Server-Szenarien) => *Locator*-Operationen

Large Objects (LOBs)

- Beispiel

```
create table Professoren
(
    PersNr          integer primary key,
    Name            varchar(30) not null,
    Rang            character(2)
                        check Rang in (`C2´, `C3´, `C4´, `W1´, `W2´, `W3´),
    Raum            integer unique,
    Passfoto        BLOB(2M),
    Lebenslauf      CLOB(75K)
);
```

- Herstellerspezifische Features
  - LOB-Attribute vom Logging freistellen
  - Gesonderten Tablespace für Speicherung definieren
  - Optionale komprimierte Speicherung

Distinct Types

- Distinct Types sind einfache benutzerdefinierte Datentypen, die man eins-zu-eins auf einen existierenden Datentypen abbilden kann
- Beispiel
  - Note als 3-stellige Dezimalzahl
    mit 2 Nachkommastellen

    create final type NotenTyp **as decimal**(3,2) **with comparisons**;

    Vergleiche von zwei NotenTyp-Werten ist erlaubt
  - Damit kann man vermeiden, dass Attributwerte semantisch falsch verwendet werden
- Achtung!!!
  - Um unterschiedliche Datentypen zu verarbeiten, muss man ggfs. Casting einsetzen (z.B. auch um Werte aus NotenTyp mit **decimal**(3,2) zu vergleichen)

Distinct Types

- Typ-spezifische Operationen können definiert werden
  - Beispiel (DB2-Syntax)
    - Berechne Durchschnittsnote (verwendet die Durchschnittsberechnung, die für **decimal** zur Verfügung steht)

```
create function NatenDurchschnitt(NotenTyp) returns NotenTyp
    source avg(decimal());
```

  - Hier auch Methoden aus Programmiersprachen einbindbar

Table Functions

- Table Functions
  - Methoden, die eine Tabelle (Menge von Tupeln) zurück liefert
  - Kann in Anfragen fast wie eine normale Relation oder View verwendet werden
  - *Wrapper*, die externe Datenquellen zugänglich machen können
  - Beispiel: suche im WWW die Biographien von Professoren der LMU

Eingabeparameter (Name des Professors)

```
create function Biographien(varchar(20)) returns
   table (
                  URL varchar(4),      ← wo die Biographie zu finden ist
                  Sprache varchar(2),  ← in welcher Sprache sie verfasst ist
                  Ranking decimal      ← Relevanz der Datenquelle
          )
…                  ←  Methodenrumpf (Informationen wie Name
…;                    der Datei, in der der Quellcode zu finden ist,
                      verwendete Prog.Sprache, …)
```

Relationale Indexstrukturen

- Ziel

  Effiziente Organisation von komplexen Objekten (z.B.: CAD- / Geo-Daten) unter der Verwendung von Standard-Datenbank-Systemen.

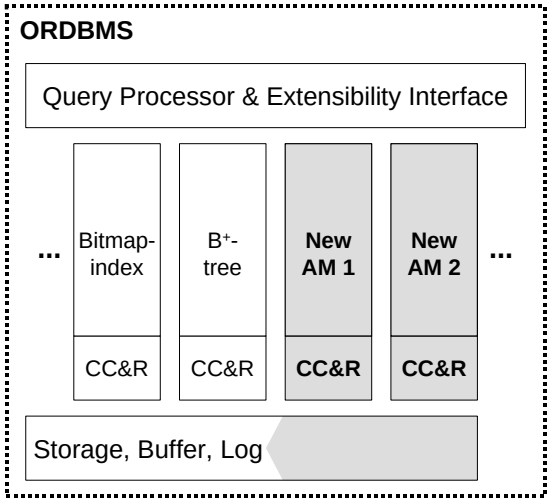
- Idee

  Integration von benutzerdefinierten Indexstrukturen in kommerzielle DBMS

- Vorteile
  - Einfache Integration in vorhandene Systeme
  - Verwendung von stabilen kommerziellen Systemen zur Organisation der Daten
  - Blockverwaltung wird dem DBMS überlassen
  - Portabilität, d.h. Implementierung ist unabhängig von der jeweiligen Platform
  - Nutzung von etablierten Konzepten für Mehrbenutzer-Betrieb, Wiederherstellung und Speichermanagement
  - Verwendung von integrierten Anfrageoptimierern und Anfragebearbeitung
  - deklarative Schnittstelle (SQL-basierte Anfragebearbeitung und Evaluation der Kostenabschätzung)

Einbettung von Indexstrukturen in ORDBMS

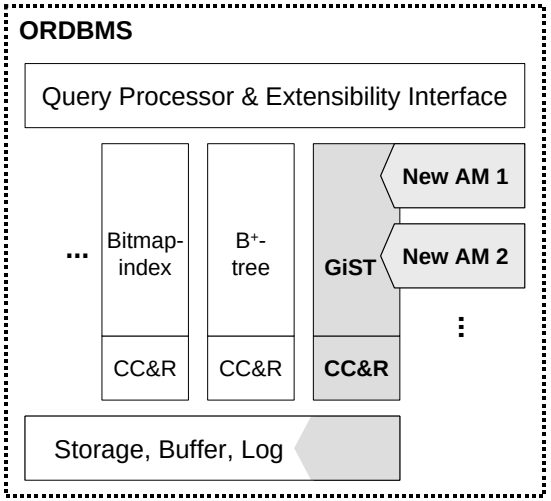
- Integrations-Ansatz:
  - Feste Verdrahtung der Indexstruktur im DBMS-Kern



- Nachteile:
  - ACID Konzept muss für jede benutzerdefinierte Indexstruktur implementiert werden
  - Freier Zugang zum Speicher-, Puffer- und Protokoll- Manager muss dem Benutzer gewährleistet werden

Einbettung von Indexstrukturen in ORDBMS

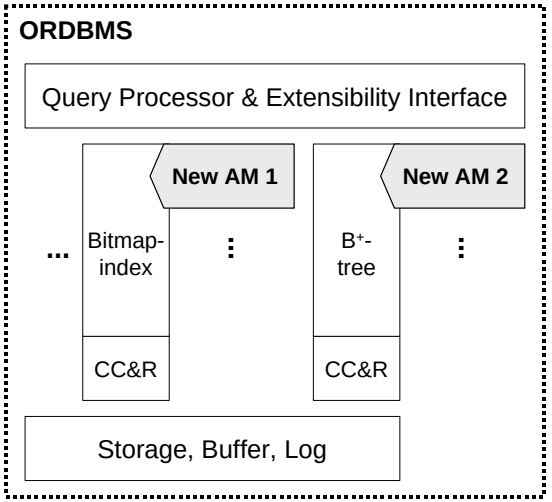
- Generischer Ansatz:
  - Verwendung eines integrierten *Generalized Search Tree* (*GiST*)
  - *GiST* fungiert als Gerüst zur Integration von blockbasierten Indexstrukturen
  - Das ACID-Prinzip wird dabei vollständig unterstützt.



- Nachteile:
  - Kein bekanntes DBMS unterstützt die volle Funktionalität von *GiST*
  - Plattformunabhängigkeit würde eine Standardisierung von *GiST* voraussetzen

Einbettung von Indexstrukturen in ORDBMS

- Relationaler Ansatz
  - Einbettung der benutzerdefinierten Indexstruktur in das relationale Schema
  - Organisation der Daten wird an die jeweilige Indexstruktur delegiert (z.B.: Bitmap-Index, B+-Baum)



- Vorteile:
  - Keine Erweiterung bzw. Modifikation des DBMS-Kerns notwendig
  - Integration ist plattform- und systemunabhängig, da SQL als Schnittstelle zum DBMS dient

Einbettung von Indexstrukturen in ORDBMS

- Relationaler Ansatz
  - Indexstruktur muss auf relationales Schema abgebildet werden
  - *Relationale Zugriffsstruktur*: jeder Index-Eintrag ist ausschließlich in einer relationalen Tabelle gespeichert oder wird von dieser abgefragt wird
  - Eine Instanz einer relationalen Zugriffsstruktur wird *Relationaler Index* genannt
  - Die folgenden Datenbank-Tabellen umfassen die persistenten Daten des relationalen Index:
    - Benutzer-Tabelle: Tabelle mit den indexierten originalen Benutzer-Daten
    - Index-Tabellen: n Tabellen ($n \geq 0$) mit den Index-Daten der Benutzer-Tabelle
    - Meta-Tabelle: Eine Tabelle für jede Datenbank und jede relationale Zugriffsstruktur, die O(1) Tabelleneinträge für jede Instanz des Indexes enthält.