Objektrelationale und erweiterbare Datenbanksysteme

- Erweiterbarkeit
- SQL:1999
- (Objekt-relationale Modellierung)

In der Vorlesung werden nur die Folien 1-12 behandelt.

Kapitel 14

Konzepte objekt-relationaler Datenbanken

- Große Objekte (Large OBjects, LOBs)

  Hierbei handelt es sich um Datentypen, die es erlauben, auch sehr große Attributwerte für z.B. Multimedia-Daten zu speichern. Die Größe kann bis zu einigen Gigabyte betragen. Vielfach werden die Large Objects den objekt-relationalen Konzepten eines relationalen Datenbanksystems hinzugerechnet, obwohl es sich dabei eigentlich um ‚reine' Werte handelt.

- Mengenwertige Attribute (NF^2)
  - Einem Tupel (Objekt) wird in einem Attribut eine Menge von Werten zugeordnet
  - Damit ist es beispielsweise möglich, den Studenten ein mengenwertiges Attribut ProgrSprachenKenntnisse zuzuordnen.
  - Schachtelung / Entschachtelung in der Anfragesprache

Konzepte objekt-relationaler Datenbanken

- Geschachtelte Relationen
  - Bei geschachtelten Relationen geht man noch einen Schritt weiter als bei mengenwertigen Attributen und erlaubt Attribute, die selbst wiederum Relationen sind.
  - z.B. in einer Relation *Studenten* ein Attribut *absolviertePrüfungen*, unter dem die Menge von Prüfungen-Tupeln gespeichert ist.
  - Jedes Tupel dieser geschachtelten Relation besteht selbst wieder aus Attributen, wie z.B. Note und Prüfer.

- Typdeklarationen
  - Objekt-relationale Datenbanksysteme unterstützen die Definition von anwendungsspezifischen Typen – oft user-defined types (UDTs) genannt.
  - Oft unterscheidet man zwischen wert-basierten (Attribut-) und Objekt-Typen (Row-Typ).

Konzepte objekt-relationaler Datenbanken

- Referenzen
  - Attribute können direkte Referenzen auf Tupel/Objekte (derselben oder anderer Relationen) als Wert haben.
  - Dadurch ist man nicht mehr nur auf die Nutzung von Fremdschlüsseln zur Realisierung von Beziehungen beschränkt.
  - Insbesondere kann ein Attribut auch eine Menge von Referenzen als Wert haben, so dass man auch N:M-Beziehungen ohne separate Beziehungsrelation repräsentieren kann
  - Beispiel: Studenten.hört ist eine Menge von Referenzen auf Vorlesungen

- Objektidentität
  - Referenzen setzen natürlich voraus, dass man Objekte (Tupel) anhand einer unveränderlichen Objektidentität eindeutig identifizieren kann

- Pfadausdrücke
  - Referenzattribute führen zur Notwendigkeit, Pfadausdrücke in der Anfragesprache zu unterstützen.

Konzepte objekt-relationaler Datenbanken

- Vererbung
  - Die komplex strukturierten Typen können von einem Obertyp erben.
  - Weiterhin kann man Relationen als Unterrelation einer Oberrelation definieren.
  - Alle Tupel der Unter-Relation sind dann implizit auch in der Ober-Relation enthalten.
  - Damit wird das Konzept der Generalisierung/Spezialisierung realisiert.

- Operationen
  - Den Objekttypen zugeordnet (oder auch nicht)
  - Einfache Operationen können direkt in SQL implementiert werden
  - Komplexere werden in einer Wirtssprache „extern“ realisiert
    - Java, C, PLSQL (Oracle-spezifisch), C++, etc.

Standardisierung in SQL:1999

- SQL2 bzw. SQL:1992
  - Derzeit realisierter Standard der kommerziellen relationalen Datenbanksysteme
  - Vorsicht: verschiedene Stufen der Einhaltung
    - Entry level ist die schwächste Stufe

- SQL:1999
  - Objekt-relationale Erweiterungen
  - Trigger
  - Stored Procedures
  - Erweiterte Anfragesprache
- Leider haben viele Systeme schon ihre eigene proprietäre Syntax (und Semantik) realisiert
  - Anpassung an den Standard kann dauern

Große Objekte: Large Objects

- CLOB
  - In einem Character Large OBject werden lange Texte gespeichert.
  - Der Vorteil gegenüber entsprechend langen varchar(...) Datentypen liegt in der verbesserten Leistungsfähigkeit, da die Datenbanksysteme für den Zugriff vom Anwendungsprogramm auf die Datenbanksystem-LOBs spezielle Verfahren (sogenannte Locator) anbieten.

- BLOB
  - In den Binary Large Objects speichert man solche Anwendungsdaten, die vom Datenbanksystem gar nicht interpretiert sondern nur gespeichert bzw. archiviert werden sollen.

- NCLOB
  - CLOBs sind auf Texte mit 1-Byte Character-Daten beschränkt. Für die Speicherung von Texten mit Sonderzeichen, z.B. Unicode-Texten müssen deshalb sogenannte National Character Large Objects (NCLOBs) verwendet werden
  - In DB2 heißt dieser Datentyp (anders als im SSQL:1999 Standard) DBCLOB (Double Byte Character Large OBject)

Beispiel-Anwendung von LOBs

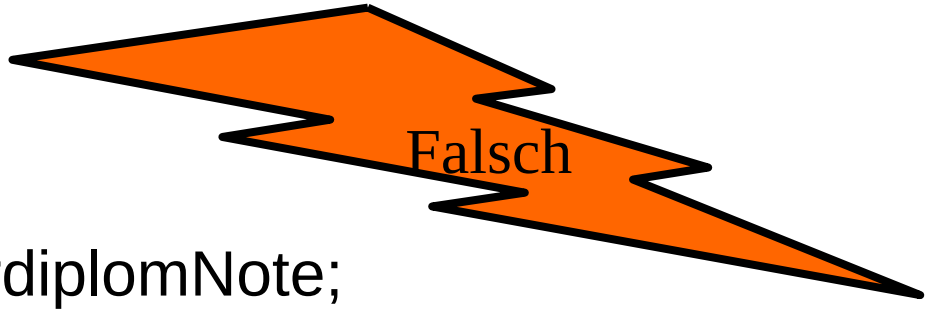
```
create table Professoren
        ( PersNr integer primary key,
        Name varchar(30) not null,
        Rang character(2) check (Rang in ('C2', 'C3', 'C4')),
        Raum integer unique,
        Passfoto BLOB(2M),
        Lebenslauf CLOB(75K) );

LOB (Lebenslauf) store as
        ( tablespace Lebensläufe
          storage (initial 50M next 50M) );
```

Der Speicherbereich kann explizit angegeben werden (um für bessere Performanz die LOBs von den ‚normalen' Daten zu trennen).

Einfache Benutzer-definierte Typen: Distinct Types

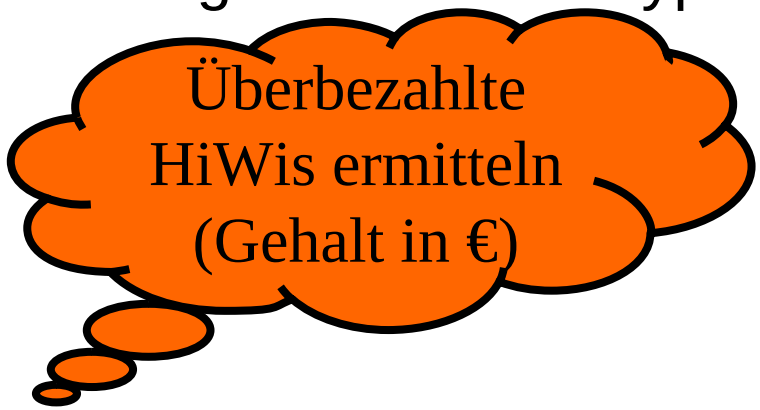
```
CREATE DISTINCT TYPE NotenTyp AS DECIMAL (3,2) WITH COMPARISONS;

CREATE FUNCTION NotenDurchschnitt(NotenTyp) RETURNS NotenTyp
    Source avg(Decimal());

Create Table Pruefen (
    MatrNr INT,
    VorlNr INT,
    PersNr INT,
    Note NotenTyp);

Insert into Pruefen Values (28106,5001,2126,NotenTyp(1.00));
Insert into Pruefen Values (25403,5041,2125,NotenTyp(2.00));
Insert into Pruefen Values (27550,4630,2137,NotenTyp(2.00));

select NotenDurchschnitt(Note) as UniSchnitt
from Pruefen;
```

Einfache Benutzer-definierte Typen: Distinct Types




```
select  *
from Studenten s
where s.Stundenlohn > s.VordiplomNote;
```

- Geht nicht: Scheitert an dem unzulässigen Vergleich zweier unterschiedlicher Datentypen NotenTyp vs. decimal
- Um unterschiedliche Datentypen miteinander zu vergleichen, muss man sie zunächst zu einem gleichen Datentyp transformieren (casting).




```
select  *
from Studenten s
where s.Stundenlohn >
                (9.99 - cast(s.VordiplomNote as decimal(3,2)));
```

Konvertierungen zwischen NotenTypen

```
CREATE DISTINCT TYPE US_NotenTyp AS DECIMAL (3,2) WITH
   COMPARISONS;


CREATE FUNCTION UsnachD_SQL(us US_NotenTyp) RETURNS
   NotenTyp
      Return (case when Decimal(us) < 1.0 then NotenTyp(5.0)
                   when Decimal(us) < 1.5 then NotenTyp(4.0)
                   when Decimal(us) < 2.5 then NotenTyp(3.0)
                   when Decimal(us) < 3.5 then NotenTyp(2.0)
                   else NotenTyp(1.0) end);


Create Table TransferVonAmerika (
    MatrNr INT,
    VorlNr INT,
    Universitaet Varchar(30),
    Note US_NotenTyp);
```

Anwendung der Konvertierung in einer Anfrage

```
Insert into TransferVonAmerika Values (28106,5041,
                'Univ. Southern California', US_NotenTyp(4.00));


select MatrNr, NotenDurchschnitt(Note)
from
      (
            (select Note, MatrNr from Pruefen) union
            (select USnachD_SQL(Note) as Note, MatrNr
             from TransferVonAmerika)
      ) as AllePruefungen
group by MatrNr
```