Kapitel 6
Objektrelationale Datenbanken

Folien zum Datenbankpraktikum
Wintersemester 2010/11 LMU München

unter Verwendung der Folien des Datenbankpraktikums aus dem Wintersemester 2007/08 von Dr. Matthias Schubert

Übersicht

Was sind objektrelationale Datenbanken?

Alternativ zu relationalen DBs gibt es objektorientierte DBs für:

- Verwaltung komplexer Objekte mit Komponenten, die wiederum Komponenten besitzen.

  ***Beispiel**: Ein Motor besteht aus einem Motorblock und einem Zylinderkopf, in dem wiederum Ventile sitzen.*

- Verwaltung verschiedener Repräsentationen desselben Objekts die bei Updates alle miteinander zu ändern sind.

  ***Beispiele**: Komplexe Zahlen, Voxelisierung und Polygonzüge eines Bauteils.*

- Einhaltung von Konsistenzbedingungen aus der Anwendung.

  ***Beispiel**: Bei der Vergrößerung des Hubraums müssen Motorblock und Zylinder im genau richtigen Abstand verändert werden.*

- Wiederverwendung von vorhandenen Basis-Bausteinen die nicht immer wieder neu entworfen werden sollen.

  ***Beispiel**: Basis-Komponenten eines Zulieferers sollen beim Design eines neuen Autos verwendet werden.*

Nachteile angebotener objektorientierter DBMS

- meist keine deskriptive DML
- kein Einzelzugriff auf die Attribute aller Objekte

  ***Beispiel**: Die Summe aller Verkäufe eines bestimmten Artikels erfordert das Lesen aller entsprechender Auftragsobjekte statt nur der einzelnen Positionen.*

- wenig Unterstützung von Multiuser-Anwendungen
- weniger ausgereifte Transaktions- und Recoverykonzepte
- kein einheitlicher Standard

→ Die Vorteile von objektorientierten Datenbanken wurden in die etablierten relationalen Systeme übernommen. Daher Kombination beider Paradigmen in einem Produkt.

→ *ORACLE* ist ein objektrelationales Datenbanksystem.

Übersicht

Objekte in *ORACLE*

- In *ORACLE* können Objekttypen vom Benutzer definiert werden.
- Beispiel:

```
CREATE TYPE person AS OBJECT (
P_ID number,
Vorname Varchar2(128),
Nachname Varchar2(128),
Geburtsdatum date,
Job job_description_type
 );
```

- Constraints können, müssen aber nicht angegeben werden.
- Jeder Objekttyp wird im Data Dictionary verwaltet.
- Jeder Datentyp ist verwendbar als Tabellenattribut, in *Object Tables* und *Object Views* und in *PL/SQL*-Programmen.

Object Tables

- verwalten persistente Objekte
- Beispiel:

```
CREATE TABLE persons OF person (P_ID Primary Key);
```

- Verwendung:

```
INSERT INTO persons VALUES (
    person (1, 'Joseph', 'Kamel',
                to_date('11-11-1970', 'dd-mm-yyyy'),
                job_description_type('Programmer', '...'));

SELECT value(p) FROM persons p WHERE p.job.description =
                                                'Programmer';
    -- gibt Objekt zurück

SELECT * FROM persons p WHERE p.job.description = 'Programmer';
    -- gibt Tupel zurück
```


Object Views

- ermöglichen einen objektorientierten Zugriff auf vorhandene Relationen
- Beispiel:

```
CREATE TABLE person_tab(
    P_ID number Primary Key,
    Vorname Varchar2(128),
    Nachname Varchar2(128),
    Geburtsdatum date,
    Job job_description_type);
```

- Anlegen der Object View:

```
CREATE VIEW person_object_view OF person
    WITH OBJECT IDENTIFIER (P_ID)
    AS SELECT * FROM person_tab;
```

Referenzen

- Ermöglichen die Darstellung eines direkten Bezugs eines Objekts auf Objektattribute.
- Beispiel:

```
CREATE TYPE Department AS OBJECT (
    Beleg_ID number PRIMARY KEY,
      ...,
    chef REF person SCOPE IS Angestellte);
```

Der Zusatz `SCOPE IS` beschränkt die referenzierten Objekte auf die Tabelle `Angestellte`.

- Vorteile gegenüber Fremdschlüsselbeziehung:
  - leichter Zugriff über Punktnotation (z.B. dept.chef.P_ID = 12434)
  - direkter Zugriff auf Row-Objekt (effizienterer Zugriff)
- Dangling REFs: Referenziertes Objekt kann ungültig werden. Test mit `IS DANGLING`.

Übersicht

Methoden

- Objekte können auch über Methoden verfügen.
- Prozeduren und Funktionen können als `MEMBER` oder `STATIC` definiert werden.
  - Aufruf von `STATIC` Methoden über Punktnotation auf Datentyp.
  - Aufruf von `MEMBER` Methoden über Punktnotation auf Datenobjekt.
- Methoden werden in Datentypdeklaration definiert und im ***Type Body*** deklariert.
- Als Programmiersprache dient ebenfalls *PL/SQL*. Der Platzhalter `SELF` steht für Selbstreferenzen.
- Beispiel:

```
SELECT a.normalize() FROM rational_tab a;
```

- Achtung: Anders als bei *PL/SQL* Prozeduren müssen hier immer Klammern nach dem Methodenaufruf gesetzt werden.

- Beispiel:

```
CREATE TYPE Rational AS OBJECT (
  num INTEGER,
  den INTEGER,
  MEMBER PROCEDURE normalize,
     ...
 );

CREATE TYPE BODY Rational AS
   MEMBER PROCEDURE normalize IS
      g INTEGER;
   BEGIN
      g := gcd(SELF.num, SELF.den);
      g := gcd(num, den);   -- äquivalent zur vorigen Zeile
      num := num / g;
      den := den / g;
   END normalize;
    ...
END;
```

Vergleichsmechanismen für Objekte

- `Map()`-Methode:

  Die `Map()`-Methode kann implementiert werden, um jedem Objekt einen Wert in einem elementaren Datentyp zuzuorden. Vergleiche werden dann auf Basis dieser Abbildung durchgeführt. `object.map()` kann also `number`, `char`, `float`, ... zurückliefern.

- `Order()`-Methode:

  Die `Order()`-Methode kann implementiert werden, um zwei Objekte direkt zu vergleichen. Sie muss so implementiert werden, dass sie ein weiteres Objekt vom gleichen Typ als Eingabe bekommt und dann einen Integer-Wert zurückgibt.

  Bedeutungen:

```
a.order(b) = 0      -- Gleichheit
a.order(b) < 0      -- a < b
a.order(b) > 0      -- a > b
```

Beide Mechanismen sind Alternativen. D.h. es muss keine aber höchstens eine Methode implementiert werden. Ist eine der beiden Methoden implementiert, funktionieren Vergleiche mit den Standardrelationen <,>, = , <= ,>=.

Übersicht

Vererbung

- *ORACLE* kennt nur einfache Vererbung.
- Das Schlüsselwort `UNDER` gibt die Oberklasse an, von der abgeleitet wird.
- `NOT FINAL` erlaubt das Anlegen eines Untertypen. `FINAL` verbietet es.
- Object Tables können auch Objekte aller Untertypen speichern.
- Mit dem Zusatz `NOT INSTANTIABLE` lassen sich abstrakte Klassen erzeugen, die nur als Oberklasse ohne eigene Instanzen fungieren.
- Ein Obertyp vererbt alle seine Methoden an die Untertypen.
- Methoden können überladen und in Subtypen überschrieben werden.
- Beim Aufruf von überschriebenen Methoden verwendet *ORACLE* dynamisches Binden.

- Beispiel:

```
CREATE TYPE Person_typ AS OBJECT (
    ssn NUMBER,
    name VARCHAR2(30),
    address VARCHAR2(100)
) NOT FINAL;

CREATE TYPE Student_typ UNDER Person_typ (
    deptid NUMBER,
    major VARCHAR2(30)
) NOT FINAL;

INSERT INTO PERSON_TAB (
    Student_typ(1, 'Hans Mustermann', '...',
                12, 'Informatik') ); -- hier Attribute
                                        des Untertyps
```

Übersicht

Mengenartige Datentypen

- VARRAY:
  - Array mit geordneten Einträgen und direktem Zugriff über Positionszähler.
  - Beim Anlegen wird Initialgröße spezifiziert.
  - Bei Bedarf kann `VARRAY` vergrößert werden.
  - Die Physikalische Speicherung erfolgt z.B. als BLOB.
- `NESTED TABLE`:
  - Dynamische Datenstruktur beliebiger Größe.
  - Daten in `NESTED TABLES` werden nicht ordnungserhaltend gespeichert.
  - Der Zugriff erfolgt über `SELECT`, `INSERT`, `DELETE` und `UPDATE`.
  - Die physikalische Speicherung erfolgt in sog. *Storage Tables* (Datenbanktabellen).
- Verwendung beider Typen in *PL/SQL*-Programmen, in Mengen-Typen und als mengenartige Attribute in Objekten.

- Beispiel:

Erstellen eigener Datentypen:

```
CREATE TYPE satellite_t AS OBJECT (
   name VARCHAR2(20),
   diameter NUMBER
 );

CREATE TYPE nt_sat_t AS TABLE OF satellite_t;

CREATE TYPE va_sat_t AS VARRAY(100) OF satellite_t;

CREATE TYPE planet_t AS OBJECT (
  name VARCHAR2(20),
  mass NUMBER,
  satellites1 va_sat_t,
  satellites2 nt_sat_t
 );

CREATE TYPE nt_pl_t AS TABLE OF planet_t;
```


Erstellen einer Tabelle mit mengenartigen Attributen:

```
CREATE TABLE stars (name VARCHAR2(20), age NUMBER, planets nt_pl_t)
   NESTED TABLE planets STORE AS planets_tab
     (NESTED TABLE satellites STORE AS satellites_tab);
```

Zugriff auf mengenartige Attribute:

```
INSERT INTO stars VALUES ( 'Sun', 23,
   nt_pl_t(
      planet_t( 'Neptune', 10, null,
          nt_sat_t(satellite_t('Proteus‘ ,67), satellite_t('Triton', 82) )
      ),
      planet_t( 'Jupiter', 189, null,
          nt_sat_t(satellite_t('Callisto', 97), satellite_t('Ganymede', 22) )
      )
   )
);
```

Anfrage:

```
SELECT p.name
FROM stars s,
     TABLE(s.planets) p,
     TABLE(p.satellites) t
WHERE t.name = 'Proteus';
```

Ergebnis:

```
Name
------------------------------
Neptune
```


Übersicht

Wichtige Funktionen

- **VALUE**: Erzeugt Instanzen aus Datensätzen (vgl. Folie 7).

  Beispiel: `select value(p) from person_table p;`

- **REF**: Erzeugt eine Referenz auf das mitgegebene Objekt.
- **DEREF**: Gibt Objekt der angegebenen Referenz zurück.
- **TREAT**: Methode zum Spezialisieren auf Subtypen. Konvertiert ein Objekt zu einen angegebenen Objekttyp, falls möglich.

  Beispiel:
  ```
  select treat(Value(p) as student_typ) from
                                          persons p;
  ```

- **IS OF TYPE**: Gibt an, ob Objekt vom angegebenen Objekttyp ist.
- **SYS_TYPEID**: Gibt die systeminterne ID des speziellsten Objekttyps zurück, dem das Objekt angehört.