

6.2 Fallbeispiel: Oracle 8 mit „Objects Option“

- ❑ Oracle war bis zur Version 7 ein rein relationales Datenbanksystem.
- ❑ Die Version 8 nennt sich *objektrelational* und enthält gegenüber dem relationalen Vorgängersystem zahlreiche Erweiterungen:
 - benutzerdefinierte Datentypen,
 - Objekte und Objektreferenzen,
 - Aggregationen von Objekten in Form von geschachtelten Tabellen oder Feldern,
 - Methoden, die Objekte eines Typs um ein benutzerdefiniertes Verhalten erweitern und
 - Mechanismen, um Regeln zu formulieren und zu überwachen.

Aufgaben bei der Erweiterung relationaler DBMS

- ❑ Die vom RDBMS bekannte Funktionalität soll erhalten bleiben und ihre Leistung darf sich nicht verschlechtern.
- ❑ Zugriffsmechanismen für neue Datenstrukturen sollen gleichberechtigt neben den bekannten Zugriffsverfahren verwendbar sein.
- ❑ Bei der Überprüfung, Optimierung und Ausführung von Anfragen müssen benutzerdefinierte Datenstrukturen und Funktionen berücksichtigt werden:
 - **Anfrageanalyse:** Wo ist der Code einer benutzerdefinierten Funktion zu finden?
 - **Optimierung:** Wie teuer (zeit- und speicheraufwendig) wird die Anfrage, wenn ein benutzerdefiniertes Prädikat ausgewertet wird?
 - **Ausführung:** Welche Zugriffsrechte hat eine benutzerdefinierte Funktion (die des Anwenders, die des Administrators)?
- ❑ **Konsequenz:** Vormalig festverdrahtete Informationen über Datentypen, Funktionen, Selektivität von Prädikaten, müssen als erweiter- und änderbare Metadaten in der Datenbank gespeichert werden (→ *Data Dictionary*).

Benutzerdefinierte Typen

```
create type adresse_t as object (
  strasse varchar(20),
  hausnummer number(3),
  ortsname varchar(30)
);
```

Basisdatentypen
(varchar, date, number)

```
create type personal_t as object (
  nachname varchar(20),
  vorname varchar(20),
  geburtsdatum date,
  gehalt number(7,2),
  kinder number(5),
  adresse adresse_t
);
```

benutzerdefinierter Typ

```
create table personal
(angestellter personal_t);
```

Tabellenerzeugung

Tabelle fungiert als „Container“ für Objekte des angegebenen Typs

Einführung in Datenbanksysteme

Fallbeispiel – Objektrelationale Datenbanken 6.2.3

Verwendung benutzerdefinierter Typen

- ❑ Jeder benutzerdefinierte Typ besitzt einen Konstruktor gleichen Namens, mit dem Objekte bzw. „Instanzen“ dieses Typs erzeugt werden können, Beispiel:

```
insert into personal values (
  personal_t( 'Mustermann',
             'Gabi',
             '07-aug-1971',
             2500.00,
             2,
             adresse_t('Musterallee',1,'Musterstadt') ) );
```

Konstruktoraufruf

- ❑ Selektion von Attributen benutzerdefinierter Typen („kaskadierte Punktnotation“):

```
select p.angestellter.nachname,
       p.angestellter.adresse.ortsname
from personal p
where p.angestellter.gehalt > 2000.0;
```

Einführung in Datenbanksysteme

Fallbeispiel – Objektrelationale Datenbanken 6.2.4

Objektreferenzen (1)

- ❑ Zu jedem Objekt wird automatisch ein systemweit eindeutiger Identifikator erzeugt (OID), der z.B. zur Modellierung von Beziehungen herangezogen werden kann
- ❑ Anwendungsbeispiel 1:N-Beziehung

Relationale Modellierung mit Fremdschlüsseln

Deklaration

```
create table abteilungen (
  nummer integer primary key,
  bezeichnung varchar(20)
)

create table angestellte (
  nachname varchar(20),
  vorname varchar(20),
  abteilung integer,
  constraint fk_abteilung
foreign key (abteilung)
references abteilungen
)
```

Benutzung

```
select ang.nachname, abt.bezeichnung
from angestellte ang, abteilungen abt
where ang.abteilung = abt.nummer
```

(liefert zu jedem Angestellten den Nachnamen und die Abteilung, in der er arbeitet)

Objektreferenzen (2)

Objektorientierte Modellierung mit OID

Deklaration

```
create type abteilung_t as object(
  nummer integer,
  bezeichnung varchar(20)
)
create table abteilungen (
  abteilung abteilung_t
)

create type angestellter_t as
object(
  nachname varchar(20),
  vorname varchar(20),
  abteilung ref abteilung_t
)
create table angestellte (
  angestellter angestellter_t
)
```

Erzeugen von Objekten

```
insert into abteilungen values(
  abteilung_t(1,'Lohnbuchhaltung')
)
```

Benutzung von Referenzen

```
insert into angestellte
select angestellter_t(
  'Mustermann',
  'Max',
  ref (abt)
)
from abteilungen abt
where abt.abteilung.nummer = 1
```

Anfragen mit Objektreferenzen (Beispiel)

Auswahl des kompletten referenzierten Objekts durch explizite Dereferenzierung

```
select a.angestellter.nachname, deref(a.angestellter.abteilung)
from angestellte a
```

Implizite Dereferenzierung durch kaskadierte Punktnotation (Beachte: Kein Join mit zweiter Tabelle nötig !)

```
select a.angestellter.nachname, a.angestellter.abteilung.bezeichnung
from angestellte a
```

Anfrage mit Objektreferenzen

```
select a.angestellter.nachname
from angestellte a
where a.angestellter.abteilung.bezeichnung = 'Lohnbuchhaltung'
```

statt

```
select ang.nachname
from angestellte ang, abteilungen abt
where ang.abteilung = abt.nummer
and abt.bezeichnung = 'Lohnbuchhaltung'
```

Objektreferenzen

□ Vorteile

- Beziehungen können durch Referenzen „natürlicher“ dargestellt werden, als durch künstlich eingeführte Fremdschlüsselattribute
- Entlang von Referenzen kann direkt zwischen verschiedenen Objekten navigiert werden.
- Komplexe Abfragen, die im relationalen Modell Joins über mehrere Tabellen erfordern, vereinfachen sich u.U. erheblich, wenn die Beziehung durch Referenzen modelliert wird (weniger Tabellen in der *from*-Klausel, besser lesbare Anfragebedingungen)

□ Nachteile

- Die Konstruktion neuer Referenzen zwischen existierenden Objekten ist aufwendig (referenziertes Objekt muß anhand eines eindeutigen Attributes identifiziert werden)
- Referenzen und Fremdschlüssel können parallel verwendet werden und führen zu einer erhöhten Komplexität im Datenmodell

Aggregationstypen

□ VARRAYs

- repräsentieren Felder mit einer festen Maximalkardinalität,
- werden „*inline*“, d.h. als Teil ihrer Tabelle gespeichert,
- können nicht innerhalb von SQL-Anfrageprädikaten verwendet werden,
- können in SQL nur als ganzes gelesen oder überschrieben werden (gilt nicht für PL/SQL),
- sind daher nur für kleine, einfach strukturierte Mengen einsetzbar.

□ Nested Tables

- können eine beliebige Menge von Datensätzen speichern und können wie VARRAYs als Attributtyp verwendet werden,
- können als Bestandteil von Anfragen verwendet werden, allerdings nicht an beliebigen Stellen im Anfrageausdruck,
- werden in Form separater, aber nicht öffentlich zugänglicher Tabellen gespeichert.

Nested Tables (Beispiel)

Tabellendefinition

```
create type telefonliste_t as table of varchar(20)
create table personal (
  nachname    varchar(20),
  vorname     varchar(20),
  telefonliste telefonliste_t
);
```

Geschachtelte Tabelle muß durch eine Anfrage selektiert werden, die genau ein Element liefert. Anfragen, die gleichzeitig mehr als eine geschachtelte Tabelle lesen oder verändern, sind **nicht** möglich.

Änderungsoperationen

```
insert into personal values
('Mustermann','Max',
telefonliste_t('040-1111', '040-2222'))
```

```
update the (select telefonliste
from personal
where nachname = 'Mustermann')
set column_value = '040-3333'
where column_value = '040-2222'
```

Methoden eines Objekttyps

- ❑ Oracle-Terminus: „member functions“

- ❑ Die Methodensignatur wird bei der Typdefinition angegeben, z.B.:

```
create type point_t as object (
  x float,
  y float,
  member function distance (p point_t) return float
)
```

- ❑ Methodenrumpfe werden durch eine getrennte Anweisung erzeugt:

```
create or replace type body point_t as
  member function distance (p point_t) return float is
  begin
    return sqrt(power(self.x-p.x,2)+power(self.y-p.y,2));
  end;
end;
```

- ❑ Member functions dürfen *keine* Seiteneffekte haben, insbesondere dürfen sie nicht den Zustand der Datenbank verändern.

Definition und Überwachung von Regeln

- ❑ Eine Regel besteht in objektrelationalen DBMS aus zwei Komponenten,

- einem Ereignis und
- einer Aktion, die beim Eintreten des Ereignisses ausgeführt wird.

- ❑ Szenarien für den Einsatz von Regeln:

- **update-update:** Sobald ein Element einer Relation verändert wird (Ereignis), werden automatisch Änderungen (Aktionen) in anderen Relationen ausgeführt (vergleichbar mit Triggern in relationalen Systemen).
- **query-update:** Sobald Inhalte einer Relation gelesen werden, wird eine Änderung in anderen Relationen ausgeführt (z.B. Protokollierung von Lesezugriffen auf sensible Daten).
- **query-query:** Wird eine bestimmte Anfrage gestellt, wird an ihrer Stelle eine andere Anfrage ausgeführt (Anwendung: Anfrage liefert nur dann das korrekte Ergebnis, wenn der Benutzer berechtigt ist, diese Anfrage zu stellen).
- **update-query:** Eine Änderungsoperation bewirkt für denselben oder einen anderen Datenbankbenutzer eine Zustandsmeldung (Ausnahmen, Alarmer).

Tücken eines komplexen Regelsystems

- In welcher Reihenfolge werden mehrere Regeln abgearbeitet, die sich auf dasselbe Ereignis beziehen ?
- Wie reagiert das DBMS, wenn eine Regelanwendung Folgeänderungen nach sich zieht, die zu Endlosschleifen führen ?
- Angenommen, eine Transaktion wird abgebrochen, sollen dann auch alle innerhalb der Transaktion erfolgten Regelanwendungen zurückgenommen werden?
- Darf der Effekt einer Regelanwendung sofort eintreten oder erst am Ende der Transaktion (z.B. weil bei sofortiger Regelanwendung Daten verloren gehen, die bis zum Ende der Transaktion noch benötigt werden)

Bei der Definition von Regeln müssen nicht nur die unmittelbaren Effekte einer Regelanwendung, sondern auch die möglichen Effekte von Folgeoperationen beachtet werden.

Unterstützung von Regeln in Oracle 8

- Nicht alle der von Stonebraker geforderten Fälle werden unterstützt**
- update-update:** unterstützt durch Trigger, also PL/SQL-Prozeduren, die unmittelbar vor, nach oder anstelle einer Änderungsoperation weitere Änderungen vornehmen können.
- query-update:** nicht unterstützt, Trigger werden nur durch Änderungsoperation ausgelöst (insert, update, delete).
- query-query:** abgesehen von den in relationalen Datenbanken üblichen Views, keine weitergehende Unterstützung.
- update-query:** unterstützt durch eine Kombination aus Triggern und Funktionen eines speziellen PL/SQL-Pakets (DBMS_ALERT)

Oracle 8: Zusammenfassung

Erzeugung neuer Typen

- selbstdefinierte Typen auf Grundlage vorhandener Basistypen
- keine Definition neuer atomarer Basistypen

Unterstützung komplexer Objekte

- über selbstdefinierte Typen
- einfache Aggregationstypen (VARRAY, nested tables) mit beschränkt leistungsfähigen Zugriffsmethoden
- Änderungen des Objektzustandes durch Aufrufe von *member functions* sind nicht möglich, lediglich Berechnungen über Inhalten von Objekten (read only).

Vererbung: nicht unterstützt

Definition und automatische Überwachung von Regeln

- Update-update-* und *update-query-*Regeln sind möglich

Oracle 8: <http://www.oracle.com/database/oracle8i/>
Informix Dynamic Server: <http://www.informix.com/informix/products/ids/>