Kapitel 6: Objektrelationale Datenbankmodelle

	Relationales Datenmodell (RDM)	Netzwerk- und Hierarchisches Datenmodell (NDM, HDM)	Objekt-orientierte Datenmodelle (OODM)	Objekt-relationale Datenmodelle
Überblick über die Konzepte	**3.1**	**4.1 4.2**	**5.1**	**6.1**
Darstellung von Assoziationen				
Datendefinition				
Anfragen				
Aktualisierungs-operationen				
Spezifika	**3.2 SQL**		**5.2 ODMG**	**6.2, 6.3**

Kapitel 6: Objektrelationale Datenbankmodelle

Lernziele und Überblick

6.1 Klassifikation von Datenbanksystemen

- ❏ Anforderungen an Datenbanksysteme
- ❏ Einordnung *objektrelationaler* Datenbanksysteme

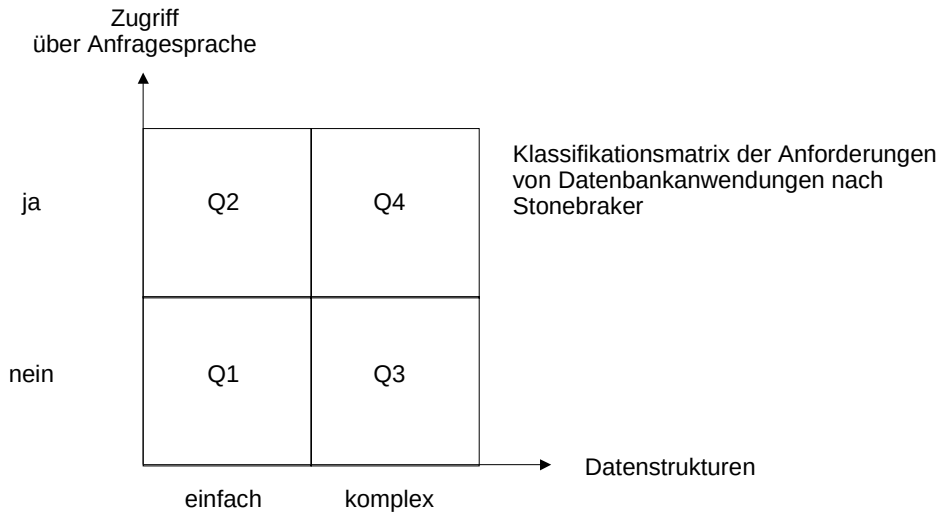
6.2 Erweiterte relationale Datenbanksysteme (Beispiel: Oracle 8)

6.3 Objektrelationale Middleware

Literatur zu 6.1

- Michael Stonebraker, Dorothy Moore:
  Object-Relational DBMSs - The Next Great Wave
  The Morgan Kaufmann Series in Data Management Systems, 1996
- C. J. Date, Hugh Darwen
  Foundation for Object/Relational Databases - The Third Manifesto
  Addison-Wesley, 1998
- Thomas M. Connolly, Carolyn E. Bragg:
  Database Systems (Chapter 23)
  A Practical Approach to Design, Implementation, and Management
  Addison-Wesley, 2nd edition, 1998

6.1 Klassifikation von Datenbanksystemen



Klassifikationsmatrix der Anforderungen von Datenbankanwendungen nach Stonebraker

Quadrant 1: Einfache Daten, keine Anfragen

- ❏ Typisches „Datenbankmanagementsystem“ dieser Klasse: Dateiverwaltungssystem des Betriebssystems (z.B. NTFS, FAT, NFS).
- ❏ Dateien werden nicht durch Anfragen bearbeitet, sondern
  - vom Anwendungsprogramm vollständig eingelesen,
  - innerhalb des Anwendungsprogramms bearbeitet und
  - nach Abschluß der Änderungen vollständig ins Dateisystem zurückgeschrieben.
- ❏ Das Dateiverwaltungssystem sorgt nur für die physikalische Speicherung der Daten.
- ❏ Einzige Datenstruktur: Die Datei - ein byteadressierbarer Adreßraum.
- ❏ Für die korrekte Interpretation und Benutzung der Inhalte einer Datei ist das jeweilige Anwendungsprogramm (z.B. Texteditor, Grafikprogramm) selbst verantwortlich.
- ❏ NDM und HDM würden auch in Q1 eingeordnet werden.

Quadrant 2: Einfache Daten mit Anfragen

- ❏ Das DBMS verwaltet Daten mit Hilfe einfacher, vordefinierter Datenstrukturen-
- ❏ Gespeicherte Daten können mit Hilfe einer deklarativen Anfragesprache untersucht und manipuliert werden.
- ❏ Typische Vertreter dieser Klasse: Relationale Datenbanksysteme (z.B. Oracle, Adabas D, CA-Ingres)
  - Basisdatenstrukturen: Tabellen, Tabellenzeilen/Tupel
  - Anfragesprache: SQL (SQL-92)
    - Manipulation von Daten:

```
update angestellte
set gehalt = gehalt * 1,03
```

    - Lesen von Daten:

```
select nachname, vorname
from angestellte
where abteilung = ‘Verkauf‘
```

Beispiel zu Q2: Relationale Datenbanksysteme

- ❏ Tupel innerhalb einer Relation setzen sich aus Werten weniger vordefinierter Wertebereiche (*Domains*) zusammen, z.B. INTEGER, FLOAT, DATE, VARCHAR.
- ❏ Tupel sind flach, die Verwendung komplexer Datenstrukturen innerhalb eines Tupels ist nicht möglich.
- ❏ Anwendungsprogramme beschreiben durch Anfragen Eigenschaften der gesuchten Tupelmenge.
- ❏ Die Bestimmung der Anfrageergebnisse übernimmt das Datenbanksystem (Auswahl von Zugriffspfaden, Optimierung der Anfrage).
- ❏ Das Datenbanksystem
  - garantiert die Dauerhaftigkeit und Integrität des von ihm verwalteten Datenbestandes,
  - löst Zugriffskonflikte, wie sie im Mehrbenutzerbetrieb auftreten können und
  - sichert die Daten gegen unbefugten Zugriff.

Quadrant 3: Komplexe Daten ohne Anfragen

- ❏ Daten werden in Form komplexer Objekte gespeichert.
- ❏ Vorhandene Objekte lassen sich beliebig zu neuen Objekten aggregieren.
- ❏ Zwischen verschiedenen Objekten können komplexe Beziehungen bestehen, z.B. Nachbarschaftsbeziehung zwischen Polygonen in einer CAD-Anwendung.
- ❏ Typische Systeme dieser Klasse: persistente, objektorientierte Programmier-sprachen (z.B. Smalltalk, Tycoon) und objektorientierte Datenbanken (z.B. O_2, Objectivity).
- ❏ Objekte eines Anwendungsprogramms werden mit all ihren Attributen und Beziehungen dauerhaft gespeichert.
- ❏ Explizite Konvertierungen aus oder in eine externe Darstellung, wie bei der Speicherung in Dateisystemen oder relationalen Datenbanken sind nicht erforderlich.
- ❏ Außer der unterschiedlichen Lebensdauer „sieht“ der Anwendungsentwickler keinen Unterschied zwischen persistenten und transienten/temporären Objekten.

Quadrant 4: Anfragen auf komplexen Daten

Beispiele

- ❏ Suche in geographischen Daten
  („Alle Bushaltestellen im Umkreis von 500 m“)
- ❏ Volltextsuche
  („In welchen Texten steht das Wort *Objekt* in der Nähe von *relational* ?“)
- ❏ Bilddatenauswertung (z.B. Warburg Electronic Library): „Alle Bilder, auf denen Konrad Adenauer zu sehen ist“
- ❏ Audiodaten („Alle Vorlesungsmitschnitte, in denen das Wort *äh* weniger als zwanzig Mal ausgesprochen wird“)

Die zu speichernden Daten sind komplex strukturiert, z.B.

- ❏ Nahverkehrsnetz (räumliche Lage von Haltestellen, Fahrpläne)
- ❏ Bilder: abgebildete Objekte und Personen, Beziehungen (Person *x* steht vor, neben, hinter Person *y*)
- ❏ Audio: Name des Sprechers, Stimmuster, Sprechgeschwindigkeit, Aufnahmeverfahren

Defizite des RDM und ODM (nach Stonebraker)

Die Struktur der Daten und ihre Beziehungen untereinander sind zu komplex, als daß herkömmliche Datenbanksysteme sie effizient bearbeiten könnten:

Relationale Datenbanken

- ❏ können den Zugriff über einzelne, einfache Attribute einer Tabelle beschleunigen, z.B. durch einen B-Baum oder eine Hashtabelle,
- ❏ erlauben die Definition solcher Zugriffsstrukturen aber nur für die Werte ihrer Basisdatentypen,
- ❏ können keine Indizes erzeugen, die Strukturen *innerhalb* eines einzelnen Wertes berücksichtigen (z.B. die Wörter innerhalb eines Strings) und
- ❏ bieten nur einfache Anfrageprädikate (z.B. =, >, <).

Objektorientierte Datenbanken

- ❏ bieten komplexe Strukturen und komplexe Methoden,
- ❏ unterstützen aber den navigierenden Objektzugriff besser als den anfrageorientierten Zugriff.

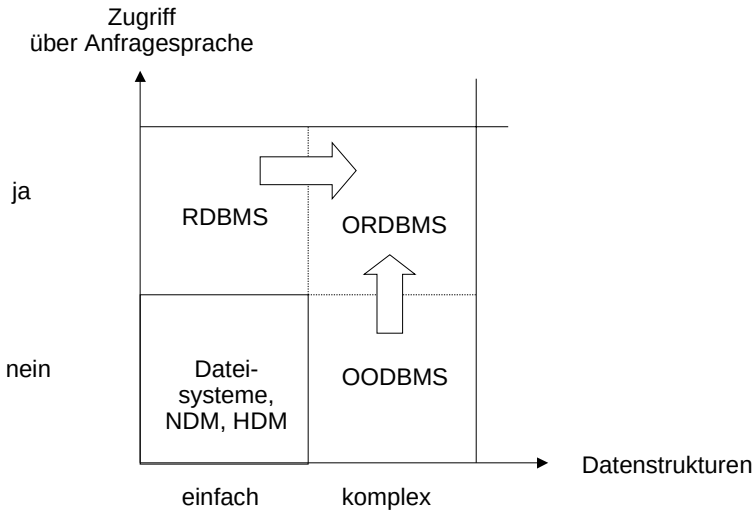
Anforderungen an ORDBMS (nach Stonebraker)

- ❏ Aus der relationalen Welt übernommene Konzepte:
  - deklarative Anfragesprache
  - Sicherung von Konsistenz, Integrität und Dauerhaftigkeit der Daten
- ❏ Aus der objektorientierten Welt übernommene Konzepte:
  - beliebige, orthogonale Kombinierbarkeit existierender Datenstrukturen zu neuen, komplexeren Strukturen
  - Vererbung
  - Objektreferenzen
  - Objekte kapseln nicht nur Daten, sondern auch Verhalten
- ❏ Erweiterbarkeit der Anfragesprache um
  - Funktionen und Operatoren für die Nutzung benutzerdefinierter Datenstrukturen

Diskussion

- ❏ Stonebrakers Klassifikation für Datenbanksysteme ist nur sehr grob:
  - Objektorientierte Datenbanksysteme erfüllen z.B. mit der *Object Query Language* (OQL) bereits viele Anforderungen, die Stonebraker an eine mächtige objektrelationale Anfragesprache stellt und
  - ehemals rein relationale Datenbanken erweitern ihren Funktionsumfang um objektorientierte Konzepte, wie Referenzen, Vererbung oder benutzerdefinierte Typen/Klassen (SQL 3).
- ❏ Bisher existiert *keine* geschlossene und allgemein akzeptierte objektrelationale Theorie oder entsprechende Standards (vgl. OQL vs. SQL 3).
- ❏ Bestehende „objektrelationale“ Datenbanken erweitern vorhandene, ehemals rein relationale oder objektorientierte Datenbanken um „nützliche“ Konzepte der jeweils anderen Systemklasse.

Klassifikation bekannter Datenbanksysteme



Notwendigkeit objektrelationaler Systeme

- ❏ Objektorientierte und relationale Systeme unterscheiden sich hinsichtlich
  - Terminologie,
  - theoretischen Grundlagen und
  - typischen Anwendungsbereichen.
- ❏ Diese historisch bedingte Unterscheidung ist bei der Entwicklung vieler datenintensiver Anwendungen nicht (mehr) möglich.
- ❏ Verbilligung von Massenspeichern und Beschleunigung der Rechnerkommunikation erlauben neben einfachen, strukturierten Daten auch die Speicherung und den Transport großer, heterogen strukturierter Datenmengen.
- ❏ Anwendungsentwickler müssen sich in zwei verschiedenen Systemwelten auskennen, wenn sie sowohl auf die Konzepte relationaler wie objektorientierter Technik angewiesen sind.
  - Beispiel: Zugriff auf eine historisch gewachsene relationale Datenbank mit Hilfe eines (objektorientierten) Java-Programms.

Aktuelle Entwicklung objektrelationaler Software

Ziel: Softwaresysteme, welche

- ❏ die Vorteile von relationaler und objektorientierter Technik in sich vereinigen,
- ❏ Brüche zwischen den ehemals getrennten Datenmodellen überwinden,
- ❏ den Entwurf komplexer Anwendungen und Datenstrukturen innerhalb eines einheitlichen theoretischen und terminologischen Rahmens erlauben.

Entwicklungslinien objektrelationaler Systeme:

- ❏ Alternative 1: Erweiterung relationaler Datenbanken um objektorientierte Konzepte
  - Oracle (Oracle 8 mit „*Objects Option*“)
  - Informix (nach dem Aufkauf von Illustra), Postgres
  - UniSQL
- ❏ Alternative 2: Integration relationaler Datenbanken in objektorientierte Systeme (s. Abschnitt 6.3)

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 als erstes ausgewertet wird?
  - **Ausführung**: Welche Zugriffsrechte hat eine benutzerdefinierte Funktion (die des Anwenders, die des Administrators)?
- **Konsequenz**: Vormals 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)
);

create type personal_t as object (
       nachname varchar2(20),
       vorname varchar2(20),
       geburtsdatum date,
       gehalt number(7,2),
       kinder number(5),
       adresse adresse_t
);

create table personal
(angestellter personal_t);
```

Basisdatentypen (varchar, date, number)

benutzerdefinierter Typ

Tabellenerzeugung

Tabelle fungiert als „Container“ für Objekte des angegebenen Typs

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;
```


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.abteilung)
from angestellte a
```

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

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

Anfrage mit Objektreferenzen

```
select ang.nachname
from angestellte ang
where ang.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

❑ **VARRAY**s

- repräsentieren Felder mit einer festen Maximalkapazität,
- 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
);
```

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'
```

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.

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
)
```

- Methodenrümpfe 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, werden automatisch Änderungen 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, Alarme).

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
- „Verhalten“ von Objekten kann nur in Form von Funktionen erfolgen, Änderungen des Objektzustandes durch Aufrufe von *member functions* sind nicht möglich

Vererbung: nicht unterstützt

Definition und automatische Überwachung von Regeln

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

Oracle 8: http://www.oracle.com/st/products/uds/oracle8/html/oracle8.html
Informix Dynamic Server: http://www.informix.com/informix/products/ids/

6.3 Objektrelationale Middleware

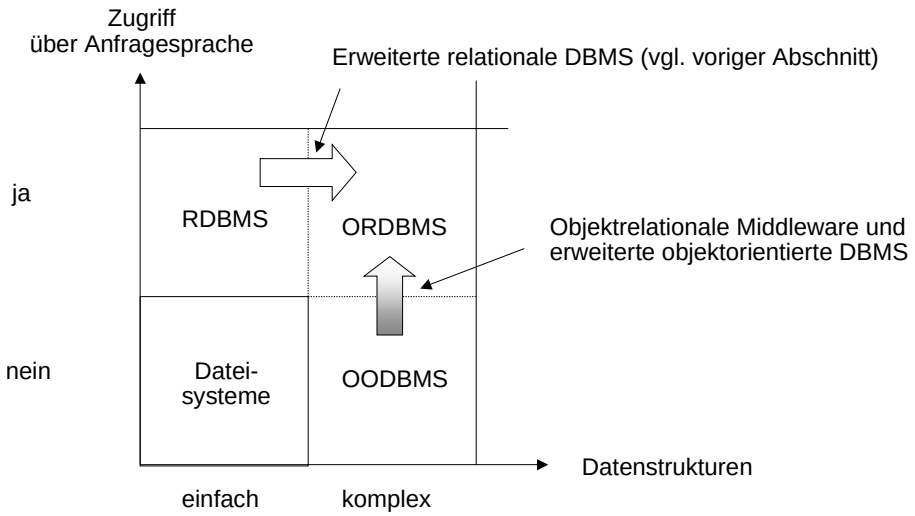
Lernziele und Überblick:

- ❏ Abgrenzung gegen objektrelationale Datenbanken
- ❏ Unterstützte Entwurfprozesse
- ❏ Leistungsumfang typischer objektrelationaler Middleware

Literatur

- ❏ Birgit Demuth: *Zweckgemeinschaft - Verschmelzung von objektorientierten und relationalen Systemen, i*X 5/1996, S. 140-148
- ❏ A. M. Keller, R. Jensen, S.Agarwal: *Persistence Software: Bridging object-oriented programming and relational databases*, SIGMOD Record, Vol. 22, No. 2, p. 523-528
- ❏ Java Blend: http://java.sun.com/products/java-blend
- ❏ Persistence Power Tier: http://www.persistence.com/products/PTejb.html
- ❏ POET SQL Object Factory: http://www.poet.de/d_prodkt/d_o_sql4/d_o_sql4.htm

Klassifikation bekannter Datenbanksysteme



Integration relationaler Dienste in OO-Systeme

- ❏ Erweiterung objektorientierter Datenbanksysteme um eine leistungsfähige Anfragesprache:
  - OQL: Obermenge von SQL, mit der sich Eigenschaften objektorientierter Datenbanksysteme (Objektnavigation, Vererbung, Orthogonalität) nutzen lassen und auch „relationale“ Anfragen möglich sind.
  - Die Anfragesprache ist Teil der Wirtsprogrammiersprache.

- ❏ Nachteile bei vollständiger Implementierung der Anfragefunktionalität innerhalb des OODBMS:
  - Durch das OO-Modell mögliche Komplexität der Datenstrukturen erschwert die Nutzung von Zugriffsstrukturen (Hashtabellen, B-Bäume), die die hohe Verarbeitungsleistung von relationalen Systemen ausmachen.
  - Um Funktionen einer relationalen Datenbank nachzubilden, muß „das Rad noch einmal neu erfunden werden“, sprich die Funktionalität eines RDBMS im objektorientierten System nachgebaut werden.

Objektrelationale Middleware

Bibliotheken zur Nutzung von Diensten relationaler Datenbanken innerhalb eines objektorientierten Systems (z.B. Java Programmiersprache).

Vorteile

- ❏ Einfach strukturierte Daten können in relationale DB ausgelagert werden und von deren Fähigkeiten (Indizierung, Konsistenzsicherung, hohe Verfügbarkeit, Mehrbenutzerbetrieb) profitieren.
- ❏ Der Benutzer „sieht“ trotzdem nur Strukturen des objektorientierten Systems, die Umsetzung von OO-Operationen auf relationale Operationen erfolgt transparent.
- ❏ Historisch gewachsene und in relationalen Datenbanken gespeicherte Daten können mit den Mitteln des objektorientierten Systems weiterverwendet werden.
- ❏ Dienstleistungen, die bereits von der relationalen Datenbank erbracht werden können, brauchen nicht im objektorientierten System erneut realisiert zu werden

Forward Engineering

Das objektorientierte System verwendet die relationale Datenbank als alternativen Objektspeicher entweder für alle seine Objekte oder eine Menge besonders gekennzeichneter Objekte.

Vorteil

Der Speicher für alle persistenten Daten eines Unternehmens liegt in Form einer einzigen, zentral administrierbaren, relationalen Datenbank vor. Für objektorientierte Daten fallen keine zusätzlichen Verwaltungsaufgaben (z.B. Datensicherung) an.

Nachteile

- ❏ Speicherung der Objekte erfolgt oft in Form von BLOBs, wodurch Stärken des RDBMS, wie schneller Schlüsselzugriff oder komplexe Anfragen nur unzureichend genutzt werden.
- ❏ Strukturiert gespeicherte Objekte können nur von der konkreten objektorientierten Anwendung genutzt werden, für die sie entwickelt wurden.

Forward Engineering Prozeß

- ❏ Objektorientierter Entwurf der Anwendungsklassen mit Hilfe eines Modellierungswerkzeugs.
- ❏ Eingabe (des relevanten Teils) der Entwurfsdaten in einen Klassengenerator, der dann
  - zu den gegebenen Klassen entsprechende Code-Fragmente (Klassenrümpfe) in der Zielprogrammiersprache (z.B. C++, Smalltalk, Java) erzeugt und
  - SQL-Befehle zur Erzeugung von Datenstrukturen zur persistenten Speicherung der Anwendungsdaten generiert.
- ❏ Programmieren der Anwendungslogik innerhalb der erzeugten Klassenstrukturen.
- ❏ Start des fertigen Programms zusammen mit einem Laufzeitsystem, das Zugriffe auf persistente Objekte in Zugriffe auf die relationale Datenbank umsetzt.

Kommerzielle Produkte zum Forward Engineering

- ❏ Modellierungswerkzeuge
  - Rational Rose
  - StP/OMT
  - Together
- ❏ Klassengeneratoren für die Erzeugung des Rumpfprogrammcodes und der relationalen Datenstrukturen
  - Rational Rose (C++, Java)
  - Persistence
  - ONTOS*Integrator (mit eigener grafischer Modellierungsschnittstelle)
  - O_2 Java Relational Binding
- ❏ Laufzeitsysteme, Objektmanager (in der Regel abgestimmt auf Klassengenerator desselben Herstellers)
  - Persistence Power Tier, O_2, ONTOS

Reverse Engineering

Das objektorientierte System kann vorhandene relationale Daten und Hilfsinformationen nutzen und in objektorientierte Strukturen übersetzen.

Vorteile

- ❏ Vorhandene relationale Daten können weiterbenutzt werden.
- ❏ Daten können über das zentrale relationale Datenbanksystem mit anderen Anwendungsprogrammen geteilt werden.
- ❏ (Teil-)Anfragen und zugehörige Optimierungen können an das relationale Datenbanksystem delegiert werden
- ❏ Beziehungen zwischen Relationen über Fremdschlüssel können in Objektreferenzen übersetzt werden.

Nachteile

- ❏ Objekte, deren Struktur die Mächtigkeit des relationalen Modells übersteigt, lassen sich nur unvollständig in der relationalen DB speichern.
- ❏ Effizienz der Nutzung vorhandener relationaler Daten hängt von der Qualität der verfügbaren Metainformationen über ein Datenbankschema ab.

Reverse Engineering Prozeß

Ausgangssituation

- ❏ Die Anwendungsprogrammiersprache ist objektorientiert.
- ❏ Es existieren bereits Anwendungsdaten, die u.U. auch von anderen, nicht-objektorientierten Anwendungsprogrammen genutzt werden.
- ❏ Die Daten sind in einer relationalen Datenbank gespeichert.

Prozeß

- ❏ Automatische oder halbautomatische Analyse des vorhandenen Datenbankschemas.
- ❏ Bereitstellung von Standardschnittstellen für den Datenbankzugriff oder Erzeugung von an das Datenbankschema angepaßten Klassen.
- ❏ Entwurf und Implementierung der Anwendungslogik unter Berücksichtigung der verfügbaren Zugriffsmethoden.
- ❏ Start des Programms, ggf. mit einem speziellen Laufzeitsystem, das die Funktionalität der DB-Zugriffe ergänzt (z.B. *Caching*).

Abbildung des Datenbankschemas

Soll ein Anwendungsprogramm ein existierendes Datenbankschema weiterbenutzen, sollte der Anwendungsentwickler es ohne tiefgreifende Kenntnisse des relationalen Modells benutzen können.

Übersetzung der Konzepte des relationalen Modells in analoge objektorientierte, z.B.

- ❏ Abbildung der Attribute einer Tabelle auf Objektattribute.
- ❏ Abbildung von Datenbankoperationen, die die Änderung einzelner Datensätze betreffen (Einfügen, Ändern, Löschen) auf Methoden der zu den Datensätzen korrespondierenden Objekte (*new*, Zuweisung, *delete*)
- ❏ Abbildung von Fremdschlüsselbeziehungen auf Objektreferenzen

relationale Datenbanken	Objektorientierte Programmierumgebung
Datensatz	Objekt
Tabelle	Klasse
Attribut	Objektattribut
Primärschlüssel	Objektidentität
Fremdschlüssel	Objektreferenz

Unterstützung des Abbildungsprozesses

- ❏ Die meisten Datenbanken speichern Informationen über Tabellenstrukturen, verwendete Typen, Primär- und Fremdschlüssel in ihrem *Data Dictionary*.
- ❏ Spezielle Entwicklungswerkzeuge können diese Metadaten lesen und daraus automatisch korrespondierende objektorientierte Strukturen generieren.
- ❏ Häufig muß der Anwendungsentwickler dem Entwicklungswerkzeug fehlende Metainformation nachträglich mitteilen (inverse Beziehungen, Kardinalitätsbeschränkungen, Zugriffsmuster, ...).
- ❏ Kommerzielle Produkte, die sowohl die automatische, als auch die vom Benutzer geführte Abbildung von relationalen Strukturen auf objektorientierte Strukturen erlauben sind z.B.
  - ONTOS*Integrator (1998)
  - Java Blend (1999)
- ❏ Aus den gegebenen Metadaten erzeugen diese Entwicklungswerkzeuge dann Klassen, mit deren Hilfe die relationalen Daten manipuliert werden können.

Standardschnittstellen zum DBMS

- ❏ Anwendungsprogramme können mit jeder relationalen Datenbank zumindest über die Sprache SQL kommunizieren („*intergalactic dataspeak*“).
- ❏ Unterschiedliche Softwarehersteller bieten Bibliotheken an, mit deren Hilfe SQL-Befehle an die Datenbank gesendet und die Ergebnisse übertragen und untersucht werden können:
  - Rogue Wave: DBTools für C++ und Java
  - JDBC (Java-SQL-Schnittstelle)
- ❏ Klassengeneratoren erzeugen Code, der auf solchen standardisierten Schnittstellen aufsetzt.
- ❏ Damit können Programme, die auf einer Standardschnittstelle aufsetzen, laufen mit jedem DBMS, für das entsprechende Treiberprogramme existieren.

Leistungen der Middleware (1)

- ❏ Umsetzung von Objektmanipulationen in Datenbankmanipulationen
  - Änderung von Attributwerten - SQL-update-Befehl
  - Erzeugung neuer Objekte - SQL-insert-Befehl
  - Lesen vorhandener Daten - SQL-select
  - Navigation mit Hilfe von Objektreferenzen
    - direkt im Objektspeicher
    - ggf. durch Formulierung einer geeigneten SQL-Anfrage (Join), wenn die Dereferenzierung zum ersten Mal erfolgt
- ❏ Konvertierung von atomaren Werten in eine korrespondierende objektorientierte Repräsentation
- ❏ Sicherung der Objektidentität
  - keine zwei Primärschlüsselattribute derselben Tabelle dürfen denselben Wert aufweisen
  - jedes Objekt korrespondiert mit genau einem Datensatz in der Datenbank

Leistungen der Middleware (2)

- ❏ Beschleunigung der Datenbankzugriffe durch einen Cache:

  Daten werden einmal aus der Datenbank gelesen und erst dann durch eine (vergleichsweise teure) SQL-Operation geändert, wenn
  - ein Anwendungsprogramm bzw. seine Änderungstransaktion beendet wird,
  - kein Anwendungsprogramm die Daten mehr benutzt und der Datensatz aus dem Zwischenspeicher entfernt wird (vgl. *Garbage Collection*) oder
  - das Anwendungsprogramm dies ausdrücklich verlangt.
- ❏ Verbergen der Komplexität, die durch die Mehrbenutzerfähigkeit der Datenbank und/oder des Cache entsteht:
  - Isolation gegen parallele Änderungsoperationen anderer Datenbanktransaktionen.
  - Behandlung von Konflikten, falls mehrere Anwendungsprogramme dieselben Objekte aus einem gemeinsamen Cache benutzen wollen.

Isolation durch Sperren

Vorteile

- ❏ Einmal aus der Datenbank gelesene Objekte können nur durch den Prozeß verändert werden, der sie auch gelesen hat.
- ❏ Änderungen an Objekten brauchen nur einmal am Ende einer Transaktion gesammelt in die Datenbank zurückgeschrieben werden.

Nachteile

- ❏ Mit jedem neuen Datenbankzugriff (Anfragen) werden neue Datensätze gesperrt, zusätzlich u.U. alle Datensätze, die von ihnen abhängen.
- ❏ Folge: Andere Datenbankbenutzer werden mit der Zeit buchstäblich „ausgesperrt“.
- ❏ Am Ende einer Datenbanktransaktion werden grundsätzlich alle Sperren freigegeben. Findet danach noch ein Zugriff auf Objekte statt, müssen die Sperren neu angefordert werden (zusätzliche DB-Kommunikation).

"Pessimistische Sperren"

Isolation durch Zeitstempel oder Versionen

Jedes Datum ist mit einer Versionsnummer oder einem Zeitstempel versehen. Die Middleware liest Daten ohne Sperren anzufordern. Änderungen erhöhen die Versionsnummer oder ändern den Zeitstempel. Beim Zurückschreiben der Daten wird die Versionsnummer im Cache mit der Versionsnummer in der DB verglichen. Bei Konflikten wird die Transaktion zurückgesetzt.

Vorteile

- ❏ Daten können von vielen Datenbankbenutzern gleichzeitig verwendet werden.
- ❏ Wartezeiten auf die Freigabe gesperrter Datensätze entfallen.

Nachteile

- ❏ Änderungen durch andere Datenbankbenutzer werden u.U. nicht oder zu spät bemerkt. Beispiel: Cache liefert noch Objekte, die in der Datenbank bereits gelöscht wurden.
- ❏ Transaktionen müssen evtl. zurückgesetzt werden, weil eine parallel laufende Transaktion dieselben Daten verändert hat und erfolgreich beendet wurde.

"Optimistische Sperren"

Kommerzielle Lösungen zur Isolation

- ❏ Das Sperren von Daten liegt in der Verantwortung des Anwendungsprogrammierers - typisch für SQL-Programmierschnittstellen, wie JDBC oder DBTools.h++
- ❏ Ein in die Anwendung integrierter Objektmanager
  - sperrt nur pessimistisch, sperrt nur optimistisch oder kann vom Anwendungsprogrammierer seinen Anforderungen entsprechend konfiguriert werden (z.B. ONTOS);
  - ist Teil der Anwendung, d.h. jede Anwendung benutzt einen eigenen Objektmanger und erscheint für die Datenbank als eigener Client.
- ❏ Ein zentraler Objektmanager für *n* verschiedene Anwendungen
  - tritt gegenüber der relationalen Datenbank als einzige Client-Anwendung auf und implementiert u.U. ein von der Datenbank völlig unabhängiges Sperrprotokoll (Beispiel: Persistence);
  - beliefert als sogenannter „Application Server“ („Middle Tier“) *n* Anwendungsprogrammen mit Daten.

Bewertung: Zentrale Objektmanager

Vorteile

- ❏ Eine Menge von Objekten, die mit Inhalten einer relationalen Datenbank korrespondieren, können vielen Anwendungen gleichzeitig zur Verfügung gestellt werden (auch sprachunabhängig als CORBA-Dienst).
- ❏ Die Einzelanwendungen brauchen kein eigenes Cache- oder Sperrprotokoll zu implementieren.
- ❏ Objektnavigation kann direkt im Objektmanager erfolgen.
- ❏ Objektmanager kann alternative Sperrprotokolle implementieren, z.B. durch Vergabe von Versionsnummern an Objekte.

Nachteile

- ❏ Alternative/leistungsfähigere Zugriffsmethoden können nur sinnvoll genutzt werden, wenn *alle* Anwendungen über *denselben* Objektmanager mit der Datenbank kommunizieren.
- ❏ Wie reagiert der Objektmanager auf Änderungen, die durch *direkte* Datenbankzugriffe erfolgen (z.B. durch ältere Datenbankanwendungen)?