Objektrelationale Datenbanken

Vortrag DB2 Vorbereitungsseminar WS 2006/2007

Christian Figura

Institut für Informatik
Martin Luther Universität Halle / Wittenberg

12. Januar 2007

Inhalt

Geschichte des objektrelationalen Ansatzes

Entwicklung

- Untersuchungen Anfang der 90’ Jahre
- federführend dabei Michael Stonebraker an der Universität von Berkeley
- Forschungen führten zu Postgres (auf dieser Basis entstand PostgreSQL)
- erstes kommerzielles Produkt Illustra (wurde von Informix gekauft)
- das objektrelationale Modell wurde in SQL 1999 und SQL 2003 standardisiert

Geschichte des objektrelationalen Ansatzes

Entwicklung

- Untersuchungen Anfang der 90’ Jahre
- federführend dabei Michael Stonebraker an der Universität von Berkeley
- Forschungen führten zu Postgres (auf dieser Basis entstand PostgreSQL)
- erstes kommerzielles Produkt Illustra (wurde von Informix gekauft)
- das objektrelationale Modell wurde in SQL 1999 und SQL 2003 standardisiert

Geschichte des objektrelationalen Ansatzes

Entwicklung

- Untersuchungen Anfang der 90’ Jahre
- federführend dabei Michael Stonebraker an der Universität von Berkeley
- Forschungen führten zu Postgres (auf dieser Basis entstand PostgreSQL)
- erstes kommerzielles Produkt Illustra (wurde von Informix gekauft)
- das objektrelationale Modell wurde in SQL 1999 und SQL 2003 standardisiert

Geschichte des objektrelationalen Ansatzes

Entwicklung

- Untersuchungen Anfang der 90’ Jahre
- federführend dabei Michael Stonebraker an der Universität von Berkeley
- Forschungen führten zu Postgres (auf dieser Basis entstand PostgreSQL)
- erstes kommerzielles Produkt Illustra (wurde von Informix gekauft)
- das objektrelationale Modell wurde in SQL 1999 und SQL 2003 standardisiert

Geschichte des objektrelationalen Ansatzes

Entwicklung

- Untersuchungen Anfang der 90’ Jahre
- federführend dabei Michael Stonebraker an der Universität von Berkeley
- Forschungen führten zu Postgres (auf dieser Basis entstand PostgreSQL)
- erstes kommerzielles Produkt Illustra (wurde von Informix gekauft)
- das objektrelationale Modell wurde in SQL 1999 und SQL 2003 standardisiert

Geschichte des objektrelationalen Ansatzes

Entwicklung

- Untersuchungen Anfang der 90’ Jahre
- federführend dabei Michael Stonebraker an der Universität von Berkeley
- Forschungen führten zu Postgres (auf dieser Basis entstand PostgreSQL)
- erstes kommerzielles Produkt Illustra (wurde von Informix gekauft)
- das objektrelationale Modell wurde in SQL 1999 und SQL 2003 standardisiert

Einordnung von ORDBs

Erweiterung des relationalen Ansatzes

- Erhaltung der Kompatibilität zum relationalen Modell
- Erweiterung zu einem objektbasiertem Modell
- echte Integration der Objektorientierung

Einordnung von ORDBs

Erweiterung des relationalen Ansatzes

- Erhaltung der Kompatibilität zum relationalen Modell
- Erweiterung zu einem objektbasiertem Modell
- echte Integration der Objektorientierung

Einordnung von ORDBs

Erweiterung des relationalen Ansatzes

- Erhaltung der Kompatibilität zum relationalen Modell
- Erweiterung zu einem objektbasiertem Modell
- echte Integration der Objektorientierung

Einordnung von ORDBs

Erweiterung des relationalen Ansatzes

- Erhaltung der Kompatibilität zum relationalen Modell
- Erweiterung zu einem objektbasiertem Modell
- echte Integration der Objektorientierung

Standardisierung ab SQL 1999

Überblick über die Erweiterungen

- Benutzerdefinierte Datentypen (UDT - User Defined Type)
- typisierte Tabellen und Sichten
- Hierarchien von Typen, Tabellen und Sichten
- OO-Konzepte
  - Kapselung
  - Vererbung
  - Substituierbarkeit
  - uvm.

Standardisierung ab SQL 1999

Überblick über die Erweiterungen

- Benutzerdefinierte Datentypen (UDT - User Defined Type)
- typisierte Tabellen und Sichten
- Hierarchien von Typen, Tabellen und Sichten
- OO-Konzepte
  - Kapselung
  - Vererbung
  - Substituierbarkeit
  - uvm.

Standardisierung ab SQL 1999

Überblick über die Erweiterungen

- Benutzerdefinierte Datentypen (UDT - User Defined Type)
- typisierte Tabellen und Sichten
- Hierarchien von Typen, Tabellen und Sichten
- OO-Konzepte
  - Kapselung
  - Vererbung
  - Substituierbarkeit
  - uvm.

Standardisierung ab SQL 1999

Überblick über die Erweiterungen

- Benutzerdefinierte Datentypen (UDT - User Defined Type)
- typisierte Tabellen und Sichten
- Hierarchien von Typen, Tabellen und Sichten
- OO-Konzepte
  - Kapselung
  - Vererbung
  - Substituierbarkeit
  - uvm.

Standardisierung ab SQL 1999

Überblick über die Erweiterungen

- Benutzerdefinierte Datentypen (UDT - User Defined Type)
- typisierte Tabellen und Sichten
- Hierarchien von Typen, Tabellen und Sichten
- OO-Konzepte
  - Kapselung
  - Vererbung
  - Substituierbarkeit
  - uvm.

Standardisierung ab SQL 1999

Überblick über die Erweiterungen

- Benutzerdefinierte Datentypen (UDT - User Defined Type)
- typisierte Tabellen und Sichten
- Hierarchien von Typen, Tabellen und Sichten
- OO-Konzepte
  - Kapselung
  - Vererbung
  - Substituierbarkeit
  - uvm.

Standardisierung ab SQL 1999

Überblick über die Erweiterungen

- Benutzerdefinierte Datentypen (UDT - User Defined Type)
- typisierte Tabellen und Sichten
- Hierarchien von Typen, Tabellen und Sichten
- OO-Konzepte
  - Kapselung
  - Vererbung
  - Substituierbarkeit
  - uvm.

Unterstützung des objektrelationalen Modells

Verfügbare Implementierungen

kommerziell

- DB2 Enterprise Server 9
- Oracle 9i
- Microsoft SQL Server 2005

frei verfügbar

- PostgreSQL (Vorreiter)
- teilweise Unterstützung auch durch andere Systeme

Implementierungen sind untereinander *nicht* kompatibel

Unterstützung des objektrelationalen Modells

Verfügbare Implementierungen

kommerziell

- DB2 Enterprise Server 9
- Oracle 9i
- Microsoft SQL Server 2005

frei verfügbar

- PostgreSQL (Vorreiter)
- teilweise Unterstützung auch durch andere Systeme

Implementierungen sind untereinander *nicht* kompatibel

Unterstützung des objektrelationalen Modells

Verfügbare Implementierungen

kommerziell

- DB2 Enterprise Server 9
- Oracle 9i
- Microsoft SQL Server 2005

frei verfügbar

- PostgreSQL (Vorreiter)
- teilweise Unterstützung auch durch andere Systeme

Implementierungen sind untereinander *nicht* kompatibel

Von relational nach objektrelational

Objektorientiertes Konzept

- Klassen mit
  - (Instanz- und Klassen-) Variablen
  - (Instanz- und Klassen-) Methoden
  - implizite Invariante
- Vererbung
- Substituierbarkeit

Von relational nach objektrelational

Objektorientiertes Konzept

- Klassen mit
  - (Instanz- und Klassen-) Variablen
  - (Instanz- und Klassen-) Methoden
  - implizite Invariante
- Vererbung
- Substituierbarkeit

Von relational nach objektrelational

Objektorientiertes Konzept

- Klassen mit
  - (Instanz- und Klassen-) Variablen
  - (Instanz- und Klassen-) Methoden
  - implizite Invariante
- Vererbung
- Substituierbarkeit

Von relational nach objektrelational

Objektorientiertes Konzept

- Klassen mit
  - (Instanz- und Klassen-) Variablen
  - (Instanz- und Klassen-) Methoden
  - implizite Invariante
- Vererbung
- Substituierbarkeit

Von relational nach objektrelational

Objektrelationales Konzept

- Erstellung von benutzerdefinierten Typen (UDTs) mit
  - (Instanz-) Variablen
  - (Instanz- und Type-) Methoden
  - (implizite Invariante)
- Erstellung von typisierten Tabellen (durch UDTs)
- Erstellung von Objekten als Zeilen in typisierten Tabellen

Von relational nach objektrelational

Objektrelationales Konzept

- Erstellung von benutzerdefinierten Typen (UDTs) mit
  - (Instanz-) Variablen
  - (Instanz- und Type-) Methoden
  - (implizite Invariante)
- Erstellung von typisierten Tabellen (durch UDTs)
- Erstellung von Objekten als Zeilen in typisierten Tabellen

Von relational nach objektrelational

Objektrelationales Konzept

- Erstellung von benutzerdefinierten Typen (UDTs) mit
  - (Instanz-) Variablen
  - (Instanz- und Type-) Methoden
  - (implizite Invariante)
- Erstellung von typisierten Tabellen (durch UDTs)
- Erstellung von Objekten als Zeilen in typisierten Tabellen

Von relational nach objektrelational

Objektrelationales Konzept

- Erstellung von benutzerdefinierten Typen (UDTs) mit
  - (Instanz-) Variablen
  - (Instanz- und Type-) Methoden
  - (implizite Invariante)
- Erstellung von typisierten Tabellen (durch UDTs)
- Erstellung von Objekten als Zeilen in typisierten Tabellen

Von relational nach objektrelational

Beispiel UDT

```
CREATE TYPE tAddress AS
  (Strasse VARCHAR(200), Ort VARCHAR(200),...)
  MODE DB2SQL
```

Von relational nach objektrelational

Neuer SQL-Typ, die Referenz

- mittels `ref(UDT-NAME)`
- kann als Spaltentyp in *allen* Tabellen benutzt werden
- Verhalten wie Fremdschlüssel
- Existenz von verschiedenen Referenzen möglich

Von relational nach objektrelational

Neuer SQL-Typ, die Referenz

- mittels `ref(UDT-NAME)`
- kann als Spaltentyp in *allen* Tabellen benutzt werden
- Verhalten wie Fremdschlüssel
- Existenz von verschiedenen Referenzen möglich

Von relational nach objektrelational

Neuer SQL-Typ, die Referenz

- mittels `ref(UDT-NAME)`
- kann als Spaltentyp in *allen* Tabellen benutzt werden
- Verhalten wie Fremdschlüssel
- Existenz von verschiedenen Referenzen möglich

Von relational nach objektrelational

Neuer SQL-Typ, die Referenz

- mittels `ref(UDT-NAME)`
- kann als Spaltentyp in *allen* Tabellen benutzt werden
- Verhalten wie Fremdschlüssel
- Existenz von verschiedenen Referenzen möglich

Von relational nach objektrelational

Neuer SQL-Typ, die Referenz

- mittels `ref(UDT-NAME)`
- kann als Spaltentyp in *allen* Tabellen benutzt werden
- Verhalten wie Fremdschlüssel
- Existenz von verschiedenen Referenzen möglich

Von relational nach objektrelational

Typisierte Tabellen

- enthalten Instanzen von UDTs
- zusammengesetzt aus
  - OID - 'Primärschlüssel' der Objekte
  - je eine Spalte pro UDT-Attribut
- je eine Zeile pro Instanz einer UDT
- Subtabellen durch Vererbung
  - jede Zeile einer Subtabelle auch in Supertabelle
  - UDT der Subtabelle muss vom UDT der Supertabelle abgeleitet sein
  - OID wird *ausschließlich* im Wurzeltyp generiert

Von relational nach objektrelational

Typisierte Tabellen

- enthalten Instanzen von UDTs
- zusammengesetzt aus
  - OID - 'Primärschlüssel' der Objekte
  - je eine Spalte pro UDT-Attribut
- je eine Zeile pro Instanz einer UDT
- Subtabellen durch Vererbung
  - jede Zeile einer Subtabelle auch in Supertabelle
  - UDT der Subtabelle muss vom UDT der Supertabelle abgeleitet sein
  - OID wird *ausschließlich* im Wurzeltyp generiert

Von relational nach objektrelational

Typisierte Tabellen

- enthalten Instanzen von UDTs
- zusammengesetzt aus
  - OID - 'Primärschlüssel' der Objekte
  - je eine Spalte pro UDT-Attribut
- je eine Zeile pro Instanz einer UDT
- Subtabellen durch Vererbung
  - jede Zeile einer Subtabelle auch in Supertabelle
  - UDT der Subtabelle muss vom UDT der Supertabelle abgeleitet sein
  - OID wird *ausschließlich* im Wurzeltyp generiert

Von relational nach objektrelational

Typisierte Tabellen

- enthalten Instanzen von UDTs
- zusammengesetzt aus
  - OID - 'Primärschlüssel' der Objekte
  - je eine Spalte pro UDT-Attribut
- je eine Zeile pro Instanz einer UDT
- Subtabellen durch Vererbung
  - jede Zeile einer Subtabelle auch in Supertabelle
  - UDT der Subtabelle muss vom UDT der Supertabelle abgeleitet sein
  - OID wird *ausschließlich* im Wurzeltyp generiert

Von relational nach objektrelational

Typisierte Tabellen

- enthalten Instanzen von UDTs
- zusammengesetzt aus
  - OID - 'Primärschlüssel' der Objekte
  - je eine Spalte pro UDT-Attribut
- je eine Zeile pro Instanz einer UDT
- Subtabellen durch Vererbung
  - jede Zeile einer Subtabelle auch in Supertabelle
  - UDT der Subtabelle muss vom UDT der Supertabelle abgeleitet sein
  - OID wird *ausschließlich* im Wurzeltyp generiert

Von relational nach objektrelational

Beispiel Typisierte Tabelle

```
CREATE TYPE tAddress AS
(Strasse VARCHAR(200), Ort VARCHAR(200),...)
MODE DB2SQL

CREATE TABLE Addresses OF tAddress
(REF IS OID USER GENERATED)
```

Von relational nach objektrelational

Objektrelationale Vererbung

- für UDTs zu definieren
- Anwendung auf typisierten Tabellen
- Überschreiben von Methoden durch `overriding`
- abstrakte UDTs durch `not instantiable`
- SQL '99 fordert `not final` zum Vererben
- SQL '03 erlaubt `final` zum Verbieten der Vererbung

Von relational nach objektrelational

Objektrelationale Vererbung

- für UDTs zu definieren
- Anwendung auf typisierten Tabellen
- Überschreiben von Methoden durch `overriding`
- abstrakte UDTs durch `not instantiable`
- SQL '99 fordert `not final` zum Vererben
- SQL '03 erlaubt `final` zum Verbieten der Vererbung

Von relational nach objektrelational

Objektrelationale Vererbung

- für UDTs zu definieren
- Anwendung auf typisierten Tabellen
- Überschreiben von Methoden durch `overriding`
- abstrakte UDTs durch `not instantiable`
- SQL '99 fordert `not final` zum Vererben
- SQL '03 erlaubt `final` zum Verbieten der Vererbung

Von relational nach objektrelational

Objektrelationale Vererbung

- für UDTs zu definieren
- Anwendung auf typisierten Tabellen
- Überschreiben von Methoden durch `overriding`
- abstrakte UDTs durch `not instantiable`
- SQL '99 fordert `not final` zum Vererben
- SQL '03 erlaubt `final` zum Verbieten der Vererbung

Von relational nach objektrelational

Objektrelationale Vererbung

- für UDTs zu definieren
- Anwendung auf typisierten Tabellen
- Überschreiben von Methoden durch `overriding`
- abstrakte UDTs durch `not instantiable`
- SQL '99 fordert `not final` zum Vererben
- SQL '03 erlaubt `final` zum Verbieten der Vererbung

Von relational nach objektrelational

Objektrelationale Vererbung

- für UDTs zu definieren
- Anwendung auf typisierten Tabellen
- Überschreiben von Methoden durch `overriding`
- abstrakte UDTs durch `not instantiable`
- SQL '99 fordert `not final` zum Vererben
- SQL '03 erlaubt `final` zum Verbieten der Vererbung

Von relational nach objektrelational

Objektrelationale Vererbung

- für UDTs zu definieren
- Anwendung auf typisierten Tabellen
- Überschreiben von Methoden durch `overriding`
- abstrakte UDTs durch `not instantiable`
- SQL '99 fordert `not final` zum Vererben
- SQL '03 erlaubt `final` zum Verbieten der Vererbung

Von relational nach objektrelational

Vererbung für Tabellen

```
CREATE TYPE tAddress AS (Strasse VARCHAR(200),
  Ort VARCHAR(200),...) MODE DB2SQL

CREATE TABLE GlobalAdd OF tAddress
  (REF IS OID USER GENERATED)

CREATE TYPE tlocalAdd UNDER tAddress AS
  (newAtt VARCHAR(10)) MODE DB2SQL

CREATE TABLE localAdd OF tlocalAdd
  UNDER GlobalAdd INHERIT SELECT PRIVILEGES
```

Von relational nach objektrelational

Objektrelationale Methoden

- Implementierung von Deklaration getrennt
- Überladen möglich
- Automatisch generierte Methoden
  - leerer Konstruktor
  - Lesemethoden für alle Attribute
  - Schreibmethoden (copy on write)

Von relational nach objektrelational

Objektrelationale Methoden

- Implementierung von Deklaration getrennt
- Überladen möglich
- Automatisch generierte Methoden
  - leerer Konstruktor
  - Lesemethoden für alle Attribute
  - Schreibmethoden (copy on write)

Von relational nach objektrelational

Objektrelationale Methoden

- Implementierung von Deklaration getrennt
- Überladen möglich
- Automatisch generierte Methoden
  - leerer Konstruktor
  - Lesemethoden für alle Attribute
  - Schreibmethoden (copy on write)

Von relational nach objektrelational

Objektrelationale Methoden

- Implementierung von Deklaration getrennt
- Überladen möglich
- Automatisch generierte Methoden
  - leerer Konstruktor
  - Lesemethoden für alle Attribute
  - Schreibmethoden (copy on write)

Von relational nach objektrelational

Methode auf UDTs

```
CREATE METHOD testPLZ (addr taddress)
  RETURNS INTEGER FOR address_t
  RETURN
   (CASE WHEN (self..plz = addr..plz)
    THEN 1
    ELSE 0
   END)
```

Von relational nach objektrelational

Methode auf UDTs (mit Deklaration)

```
CREATE TYPE tAddress AS (Strasse VARCHAR(200),
  ...) MODE DB2SQL
  METHOD method-name (addr taddress)
  RETURNS INTEGER LANGUAGE SQL

CREATE METHOD testPLZ (addr taddress)
  RETURNS INTEGER FOR address_t
  RETURN
   (CASE WHEN (self..plz = addr..plz)
    THEN 1 ELSE 0
   END)
```

Von relational nach objektrelational

Typumwandlung

- Test auf Typzugehörigkeit mit `OBJEKT is of UDT`
- 'down cast' mit `treat(OBJEKT as OBJEKT)`
- 'up cast' mit `(OBJEKT as OBJEKT)`
- Referenzumwandlung `cast(REFERENZ as REFERENZ)`

Von relational nach objektrelational

Typumwandlung

- Test auf Typzugehörigkeit mit `OBJEKT is of UDT`
- 'down cast' mit `treat(OBJEKT as OBJEKT)`
- 'up cast' mit `(OBJEKT as OBJEKT)`
- Referenzumwandlung `cast(REFERENZ as REFERENZ)`

Von relational nach objektrelational

Typumwandlung

- Test auf Typzugehörigkeit mit `OBJEKT is of UDT`
- 'down cast' mit `treat(OBJEKT as OBJEKT)`
- 'up cast' mit `(OBJEKT as OBJEKT)`
- Referenzumwandlung `cast(REFERENZ as REFERENZ)`

Von relational nach objektrelational

Typumwandlung

- Test auf Typzugehörigkeit mit `OBJEKT is of UDT`
- 'down cast' mit `treat(OBJEKT as OBJEKT)`
- 'up cast' mit `(OBJEKT as OBJEKT)`
- Referenzumwandlung `cast(REFERENZ as REFERENZ)`

Von relational nach objektrelational

Typumwandlung

- Test auf Typzugehörigkeit mit `OBJEKT is of UDT`
- 'down cast' mit `treat(OBJEKT as OBJEKT)`
- 'up cast' mit `(OBJEKT as OBJEKT)`
- Referenzumwandlung `cast(REFERENZ as REFERENZ)`

Von relational nach objektrelational

Zugriff auf Objektattribute und Methoden

- Ursprungsform `deref(REFERENZ).ATTRIBUT`
- Kurzform `REFERENZ->ATTRIBUT`
- Beide Formen auch für Methoden
  - `deref(REFERENZ).METHODE(PARAMETER)`
  - `REFERENZ->METHODE(PARAMETER)`
- In DB2 als Besonderheit mit .. statt ->

Von relational nach objektrelational

Zugriff auf Objektattribute und Methoden

- Ursprungsform `deref(REFERENZ).ATTRIBUT`
- Kurzform `REFERENZ->ATTRIBUT`
- Beide Formen auch für Methoden
  - `deref(REFERENZ).METHODE(PARAMETER)`
  - `REFERENZ->METHODE(PARAMETER)`
- In DB2 als Besonderheit mit .. statt ->

Von relational nach objektrelational

Zugriff auf Objektattribute und Methoden

- Ursprungsform `deref(REFERENZ).ATTRIBUT`
- Kurzform `REFERENZ->ATTRIBUT`
- Beide Formen auch für Methoden
  - `deref(REFERENZ).METHODE(PARAMETER)`
  - `REFERENZ->METHODE(PARAMETER)`
- In DB2 als Besonderheit mit .. statt ->

Von relational nach objektrelational

Zugriff auf Objektattribute und Methoden

- Ursprungsform `deref(REFERENZ).ATTRIBUT`
- Kurzform `REFERENZ->ATTRIBUT`
- Beide Formen auch für Methoden
  - `deref(REFERENZ).METHODE(PARAMETER)`
  - `REFERENZ->METHODE(PARAMETER)`
- In DB2 als Besonderheit mit .. statt ->

Von relational nach objektrelational

Zugriff auf Objektattribute und Methoden

- Ursprungsform `deref(REFERENZ).ATTRIBUT`
- Kurzform `REFERENZ->ATTRIBUT`
- Beide Formen auch für Methoden
  - `deref(REFERENZ).METHODE(PARAMETER)`
  - `REFERENZ->METHODE(PARAMETER)`
- In DB2 als Besonderheit mit .. statt ->

Ein Beispiel

Person-Adresse OR-Anwendung

Person(pid, Name, aid)
Adresse(aid, Strasse, Ort)

```
SELECT * FROM Person A, Adresse B
  WHERE A.aid=B.aid AND B.Ort='Halle'
```

Adresse(Strasse, Ort)
Person(pid, Name, Adresse ref(Adresse))

```
SELECT * FROM Person A
  WHERE A.adresse..Ort='Halle'
```

Ein Beispiel

Person-Adresse OR-Anwendung

Person(pid, Name, aid)
Adresse(aid, Strasse, Ort)

```
SELECT * FROM Person A, Adresse B
  WHERE A.aid=B.aid AND B.Ort='Halle'
```

Adresse(Strasse, Ort)
Person(pid, Name, Adresse ref(Adresse))

```
SELECT * FROM Person A
  WHERE A.adresse..Ort='Halle'
```

Ein Beispiel

Ort-Vergleich Datenbank

Adresse(Strasse, Ortsname, PLZ)

besser

Adresse(Strasse, OrtID)

Orte(OrtID, Ortsname, PLZ)

noch besser

Orte(Ortsname, PLZ) Adresse(Strasse, Ort ref(Orte))

```
SELECT * FROM Adresse A
  WHERE A.Ort..Ortsname='Halle'
```

Ein Beispiel

Ort-Vergleich Datenbank

Adresse(Strasse, Ortsname, PLZ)
besser
Adresse(Strasse, OrtID)
Orte(OrtID, Ortsname, PLZ)
noch besser
Orte(Ortsname, PLZ) Adresse(Strasse, Ort ref(Orte))

```
SELECT * FROM Adresse A
  WHERE A.Ort..Ortsname='Halle'
```

Ein Beispiel

Ort-Vergleich Datenbank

Adresse(Strasse, Ortsname, PLZ)
besser
Adresse(Strasse, OrtID)
Orte(OrtID, Ortsname, PLZ)
noch besser
Orte(Ortsname, PLZ) Adresse(Strasse, Ort ref(Orte))

```
SELECT * FROM Adresse A
  WHERE A.Ort..Ortsname='Halle'
```

Ein Beispiel

Ort-Vergleich Datenbank

Adresse(Strasse, Ortsname, PLZ)
besser
Adresse(Strasse, OrtID)
Orte(OrtID, Ortsname, PLZ)
noch besser
Orte(Ortsname, PLZ) Adresse(Strasse, Ort ref(Orte))

```
SELECT * FROM Adresse A
  WHERE A.Ort..Ortsname='Halle'
```

Eigentliches Problem von RDBs

Impedance Mismatch

- Problem
  - objektorientierte Sprache
  - relationales Datenbankmodell
- zwei Datenmodelle müssen entwickelt werden
- oder OO-R Mapping muss vollzogen werden

Eigentliches Problem von RDBs

Impedance Mismatch

- Problem
  - objektorientierte Sprache
  - relationales Datenbankmodell
- zwei Datenmodelle müssen entwickelt werden
- oder OO-R Mapping muss vollzogen werden

Eigentliches Problem von RDBs

Impedance Mismatch

- Problem
  - objektorientierte Sprache
  - relationales Datenbankmodell
- zwei Datenmodelle müssen entwickelt werden
- oder OO-R Mapping muss vollzogen werden

Eigentliches Problem von RDBs

Impedance Mismatch

- Problem
  - objektorientierte Sprache
  - relationales Datenbankmodell
- zwei Datenmodelle müssen entwickelt werden
- oder OO-R Mapping muss vollzogen werden

Impedance Mismatch verringern

ORDBMSs

- Objekte und Methode können in der DB abgebildet bzw. hinterlegt werden
- kein separates Datenmodell nötig
- Zusammenrücken von Java und SQL
- Verringerung der Komplexität von SQL-Anfragen

Impedance Mismatch verringern

ORDBMSs

- Objekte und Methode können in der DB abgebildet bzw. hinterlegt werden
- kein separates Datenmodell nötig
- Zusammenrücken von Java und SQL
- Verringerung der Komplexität von SQL-Anfragen

Impedance Mismatch verringern

ORDBMSs

- Objekte und Methode können in der DB abgebildet bzw. hinterlegt werden
- kein separates Datenmodell nötig
- Zusammenrücken von Java und SQL
- Verringerung der Komplexität von SQL-Anfragen

Impedance Mismatch verringern

ORDBMSs

- Objekte und Methode können in der DB abgebildet bzw. hinterlegt werden
- kein separates Datenmodell nötig
- Zusammenrücken von Java und SQL
- Verringerung der Komplexität von SQL-Anfragen

Impedance Mismatch verringern

ORDBMSs

- Objekte und Methode können in der DB abgebildet bzw. hinterlegt werden
- kein separates Datenmodell nötig
- Zusammenrücken von Java und SQL
- Verringerung der Komplexität von SQL-Anfragen

Probleme von ORDBs

Schwierigkeiten durch OR

- Probleme mit Arrays und Listen
- Unterobjekte nicht referenzierbar
- Restriktionen beim OO-Model
- *keine* Portabilität des SQL-Codes

Probleme von ORDBs

Schwierigkeiten durch OR

- Probleme mit Arrays und Listen
- Unterobjekte nicht referenzierbar
- Restriktionen beim OO-Model
- *keine* Portabilität des SQL-Codes

Probleme von ORDBs

Schwierigkeiten durch OR

- Probleme mit Arrays und Listen
- Unterobjekte nicht referenzierbar
- Restriktionen beim OO-Model
- *keine* Portabilität des SQL-Codes

Probleme von ORDBs

Schwierigkeiten durch OR

- Probleme mit Arrays und Listen
- Unterobjekte nicht referenzierbar
- Restriktionen beim OO-Model
- *keine* Portabilität des SQL-Codes

Probleme von ORDBs

Schwierigkeiten durch OR

- Probleme mit Arrays und Listen
- Unterobjekte nicht referenzierbar
- Restriktionen beim OO-Model
- *keine* Portabilität des SQL-Codes

Vielen Dank für Ihre Aufmerksamkeit

Fragen ?

Quellenverzeichnis

Verwendete Literatur

- dpunkt.verlag, Can Türker und Gunter Saake, Objektrelationale Datenbanken Ein Lehrbuch
- FH Braunschweig/Wolfenbüttel, Holger Märtens, Objektbasierte Datenbanken SoS 2005
- Wikipedia.org, 'Object-relational database', Letzter Zugriff: 11.01.2007
- Wikipedia.org, 'Object-relational impedance mismatch', Letzter Zugriff: 11.01.2007