



Hibernate

Objekt/Relationales Mapping
für Java



Wer bin ich?

- Stefan Wille
- Softwareentwickler / Architekt seit 1996
- Freelancer seit 2000
- Autor von „Goto JavaServer Pages“ in 2001



Wesentliche Themen

- Was ist ORM? Warum ORM?
- Was ist Hibernate?
- Grundlegende Features
- Queries
- Vererbung
- Fortgeschrittene Features, Tools
- Wie mit Hibernate anfangen?
- EJB3



Was ist ORM?
Warum ORM?



Was ist ORM?

- Objekt/Relationales Mapping verknüpft Objekte und OOP mit RDBMS
- Entwickler konstruieren ihre Software mit Objekten. ORM bildet dann API-Calls und Objekte auf SQL und Tabellen ab
- Freie und kommerzielle ORM verfügbar
- ORM war im Java-Umfeld zunächst nicht so populär, hat sich in den letzten Jahren geändert. Wichtiger Grund: Hibernate



Was ist ORM?

- Mapping im einfachsten Fall:
 - Klasse \Leftrightarrow Tabelle
 - Objekt \Leftrightarrow Tabellenzeile
 - Property \Leftrightarrow Tabellenattribut
 - Objektreferenz \Leftrightarrow Fremdschlüssel
- Moderne ORMs bieten wesentlich flexiblere Mappings



Warum ORM?

- Viel weniger Code als bei JDBC
- Strukturelles Mapping von Java auf RDBMS robuster
- Weniger fehleranfällig
- Performance-Optimierung jeder Zeit
- Portabilität (über RDBMS)



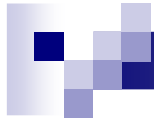
ORM ist nicht perfekt

- OOP und RDBMS haben widersprüchliche Konzepte:
 - Mengenverarbeitung / einzelne Objekte
 - Typ-Vererbung
 - Entity-Relationships sind bidirektional, Objektreferenzen nur unidirektional
 - Objektreferenzen, Collections / Joins
- Deshalb: ORM ist sehr hilfreich, aber das Verständnis für RDBMS und SQL bleibt nötig



Das Ziel

Die Vorteile von SQL-Datenbanken ausnutzen, ohne die Java-Welt von Klassen und Objekten zu verlassen.



Das wirkliche Ziel

Weniger Zeit und Arbeit für Persistenz
aufwenden und einen zufriedenen DBA
haben



Moderne ORM-Lösungen

- Transparente Persistenz (POJOs)
- Automatische Dirty-Erkennung
- Transitive Persistenz
- Verschiedene Ansätze für Vererbung
- Geschicktes Fetching und Caching
- Entwicklung-Tools



Definition: Transparente Persistenz

- Jede Klasse kann eine *persistente* Klasse sein
 - Keine Interfaces zu implementieren
 - Keine Superklasse zu subclassen
- Persistente Klassen außerhalb des Persistenz-Kontexts nutzbar (Unit-Tests, Web-Framework, XML Data Binding)



Was ist Hibernate?




Hibernate

- ORM Implementation
- Open Source (LGPL)
- Ausgereift, Entwicklung durch User-Requests getrieben
- Populär (3.000 Downloads / Tag)
- Die Messlatte, die es für andere Produkte zu schlagen gilt



Features

- Persistenz für POJOs (Java Beans)
- Flexibles Mapping (XML)
- Sehr leistungsfähige, schnelle Queries
- Ausgefeiltes Caching
- Toolset (hbm2java, hbm2ddl, ...)
- Unterstützung für Detached Objects (keine DTOs mehr)
- Standard-Collection-Klassen für Properties
- ...



Eigenschaften von persistenten Klassen in Hibernate

- Java Beans-Spezifikation (POJOs)
- Property xyz → Getter/Setter getXyz() / setXyz()
- NoArg-Konstruktor
- Collection-Properties als Interface



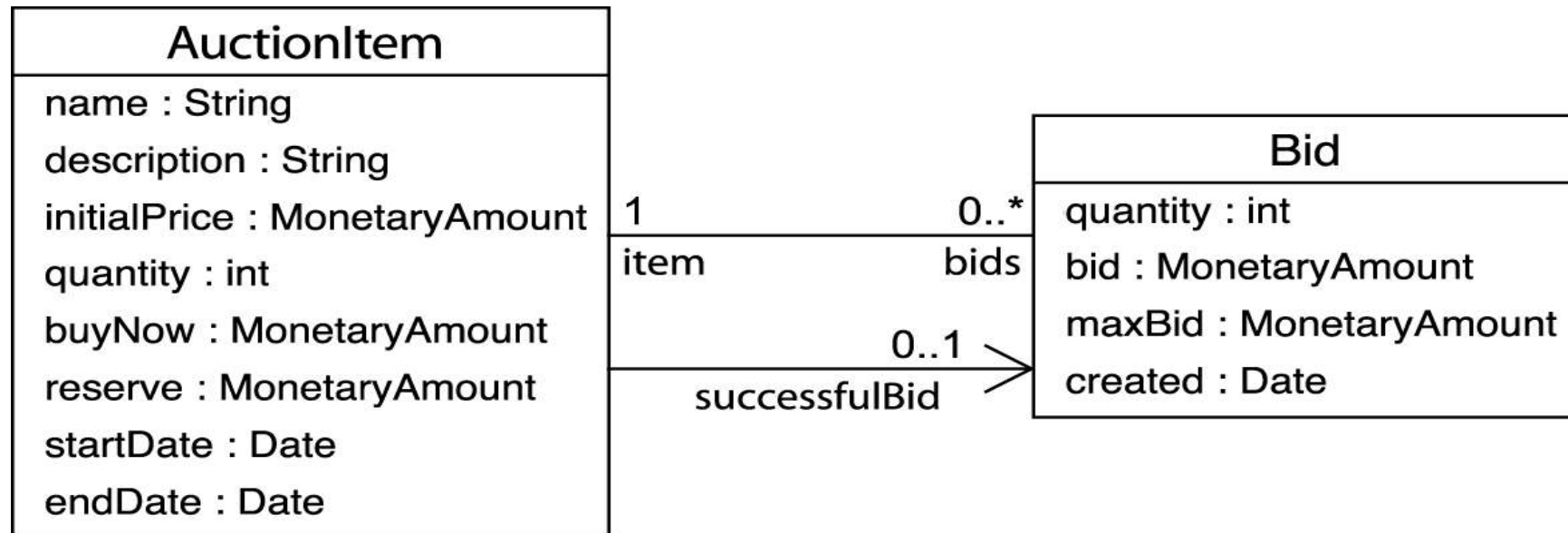
Unterstützte Datenbanken

- Hibernate unterstützt die SQL-Dialekte von mehr als 20 Datenbanken, darunter
 - Oracle, DB2, MS SQL Server, Sybase
 - PostgreSQL, MySQL
 - HypersonicSQL, Mckoi SQL, SAP DB, Interbase, Pointbase, Progress, FrontBase, Ingres, Informix, Firebird



Hibernate in einem Beispiel

Beispiel: Objekt-Modell





Beispiel: Klasse AuctionItem

```
public class AuctionItem {  
    public AuctionItem()...  
  
    public void setDescription(String dsptn) ...  
    public String getDescription() ...  
  
    public void setBids(java.util.Set bids) ...  
    public java.util.Set getBids()...  
    ...  
}
```



XML Mapping-Dokument

```
<class name="AuctionItem" table="AUCTION_ITEM">
  <id name="id" column="ITEM_ID">
    <generator class="native"/>
  </id>

  <property name="description" column="DESCR"/>

  <set name="bids" cascade="all" lazy="true">
    <key column="ITEM_ID"/>
    <one-to-many class="Bid"/>
  </set>

  <many-to-one name="successfulBid"
    column="SUCCESSFUL_BID_ID"/>

  ...
</class>
```



Automatisches Dirty-Checking

Lese ein `AuctionItem` und ändere `description`:

```
Session session = sessionFactory.openSession();  
Transaction tx = session.beginTransaction();  
AuctionItem item =  
    (AuctionItem) session.get(AuctionItem.class, itemId);  
item.setDescription(newDescription);  
tx.commit();  
session.close();
```




Transitive Persistenz (Objektgraphen)

Lese ein `AuctionItem` und erzeuge einen neuen, persistententen `Bid`:

```
Bid bid = new Bid()
bid.setAmount(bidAmount);
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
AuctionItem item =(AuctionItem)
    session.get(AuctionItem.class, itemId);
bid.setItem(item);
item.getBids().add(bid);
tx.commit();
session.close();
```

Assoziation durch die
Applikation gemanaged!





Detached Objects

Lese ein `AuctionItem` und ändere `description`:

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
AuctionItem item =
(AuctionItem) session.get(AuctionItem.class, itemId);
tx.commit();
session.close();
```

```
item.setDescription(newDescription);
```

← Kein automatisches
Dirty-Checking

```
Session session2 = sessionFactory.openSession();
Transaction tx = session2.beginTransaction();
session2.update(item);
tx.commit();
session2.close();
```




Hibernate Query- Features



Hibernate Query-Features

- **Hibernate Query Language (HQL)**
 - objektorientierter Dialekt von ANSI SQL
- **Query by Criteria (QBC)**
 - erweiterb. Framework von Query-Objekten
 - darunter: Query by Example (QBE)
- **Native SQL Queries**
 - Query wird an DB durchgereicht
 - Automatisches Mapping auf Objekte



Hibernate Query Language

- „Objektorientiertes SQL“
 - Klassen und Properties statt Tabellen und Attribute
 - Unterstützt Polymorphismus
 - Automatisches Joining für Assoziationen
 - *wesentlich* kürzere Queries als in SQL
- Relationale Operationen voll unterstützt
 - inner/outer/full joins
 - Projektion, Sortieren, Aggregation, Grouping
 - Subqueries, SQL Funktionen



Die einfachste HQL-Query

```
from AuctionItem
```

d.h. liefere alle AuctionItems.

```
List allAuctionItems =  
    session.createQuery("from AuctionItem").list()
```



Eine kompliziertere HQL-Query

```
select item
from AuctionItem item
join item.bids as bid
where item.description like "Hibernate%"
and bid.amount > 100
```

d.h. lese alle `AuctionItems` mit einem `Bid` höher als 100 und einer `description`, die mit "Hibernate" anfängt.



Eine Criteria-Query

```
List auctionItems =
    session.createCriteria(AuctionItem.class)
        .setFetchMode("bids", FetchMode.EAGER)
        .add( Expression.like("description", desc) )
        .createCriteria("successfulBid")
        .add( Expression.gt("amount", minAmount) )
        .list();
```

Entspricht in HQL:

```
from AuctionItem item
    left join fetch item.bids
where item.description like :description
    and item.successfulbid.amount > :minAmount
```



Query By Example

```
Bid exampleBid = new Bid();
exampleBid.setAmount(100);
List auctionItems =
session.createCriteria(AuctionItem.class)
    .add( Example.create(exampleBid) )
    .createCriteria("bid")
    .add( Expression("created", yesterday)
    .list();
```

Entsprechend in HQL:

```
from AuctionItem item
    join item.bids bid
where bid.amount = 100
    and bid.created = :yesterday
```



Vererbung in Hibernate



Vererbung in Hibernate

- Deutlichstes Beispiel für OO/RDBMS-Mismatch
- Mühsam mit reinem JDBC zu implementieren
- Hibernate: drei Strategien
 - Tabelle pro Klassenhierarchie
 - Tabelle pro konkreter Klasse
 - Tabelle pro Klasse
- Mix innerhalb einer Hierarchie möglich

Beispiel: Klassendiagramm

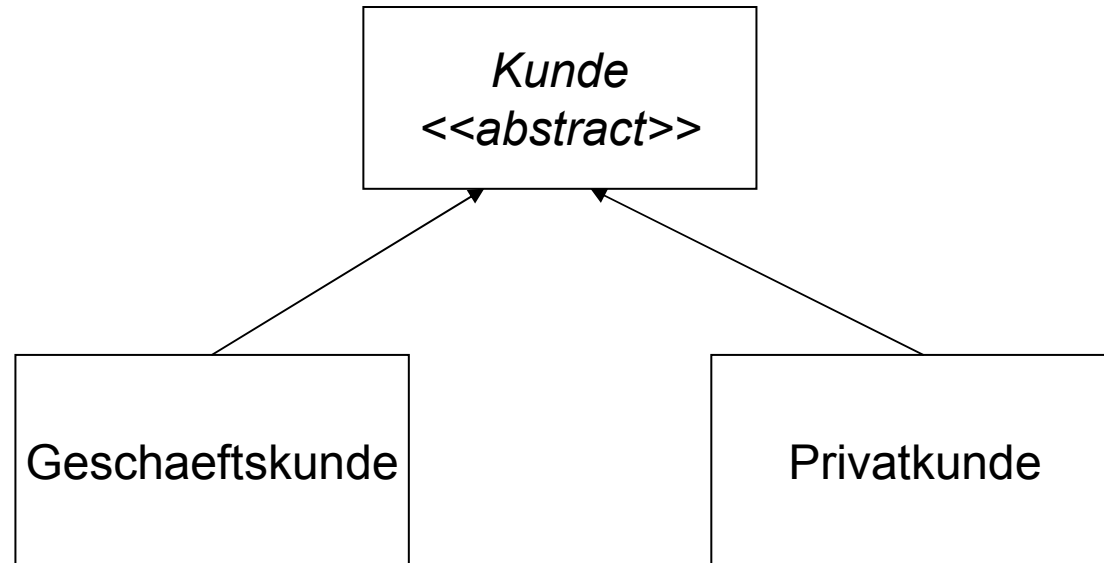


Tabelle pro Klasse

- Natürliches Mapping
- Performance etwas schlechter als bei Tabelle pro Klassenhierarchie

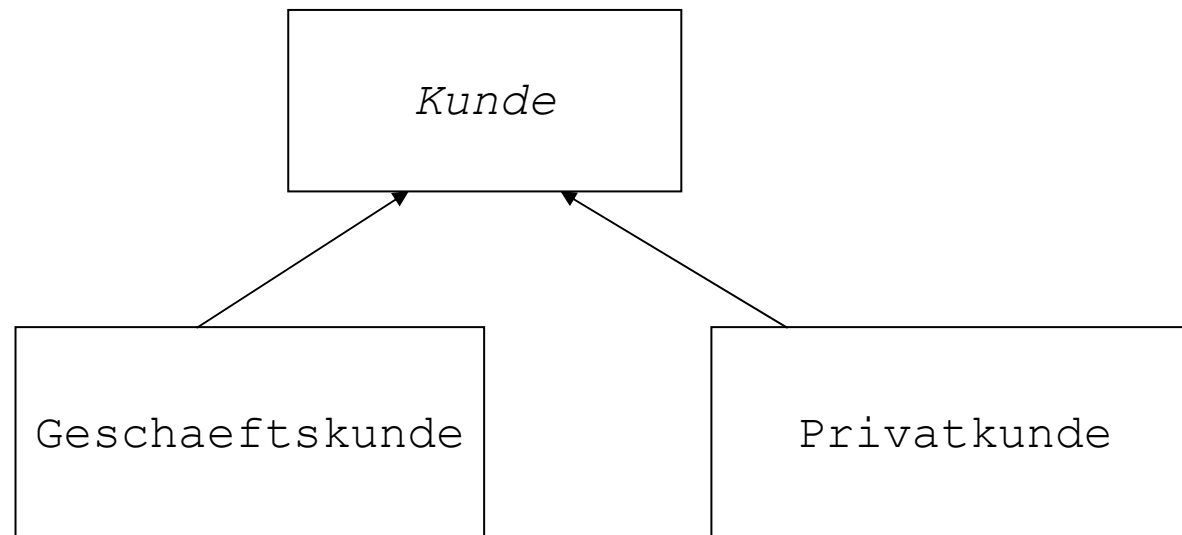




Tabelle pro Klasse - 2

- Generiertes SQL deutlich besseres SQL als bei handgeschriebenem JDBC
- **Hibernate:**

```
select kunde0_.id as id, kunde0_.name as name0_, kunde0_.remark as remark0_, kunde0_1_.revenue as revenue1_, case when kunde0_1_.id is not null then 1 when kunde0_2_.id is not null then 2 when kunde0_.id is not null then 0 end as clazz_ from kunde_ kunde0_ left outer join geschaeftskunde_ kunde0_1_ on kunde0_.id=kunde0_1_.id left outer join privatkunde_ kunde0_2_ on kunde0_.id=kunde0_2_.id
```



Weitere Features



Mehr Features

- Automatische, optimistische Currency Control
- Batch Insert/Update (1 Statement / n Records)
- Delete kann Cascaded Delete nutzen
- Exotische Mappings
- Named Queries
- Load/Insert/Update über handgeschriebenes SQL / Stored Procedures



Noch mehr Features

- Statistik-API
- Filter (Mandantenfähigkeit, Instance based Security ...)
- Bulk Update/Delete ("update ... where ...")
- Mapping-Metadaten per API zugänglich
- Entities können nicht nur Java-Objekte, sondern auch XML (DOM4J) oder HashMaps sein



Was kommt an neuen Features

- Alles noch Beta!
- Stateless Sessions
- Metadaten mit Annotations
- Validation-Framework (über Annotations)
- Lucene-Indizierung (über Annotations)
- HibernateTools – Plugin für Eclipse
- EJB3-API



AuctionItem im Annotation-Stil

```
@Entity
@Table(name = „auction_item“)
public class AuctionItem {
    public AuctionItem()...

    @Basic
    @Column(name = „desc“)
    public void setDescription(String description) ...
    public String getDescription () ...

    @OneToMany(mappedBy=„item“)
    public void setBids(java.util.Set<Bid> bids) ...
    public java.util.Set<Bid> getBids()...
    ...
}
```



Tools



Tools 1

- hbm2java (HBM → Java)
- hbm2ddl (HBM → DDL)
- XDoclet
- Middlegen
- AndroMDA
- Spring-Integration (kein Tool, aber...)



Tools 2

- HibernateTools Plugin für Eclipse
 - Java Code-Generierung
 - DB Schema Reverse-Engineering
 - HBM-Editor
 - Query-Editor mit SQL-Preview
 - Grafische Modellansicht

HibernateTools – HBM-Editor

```
<!-- Mapping for the component class Address. -->
<component name="address" class="Address">
  <property name="street"
    type="
    column
    length
  >
    <property name="
      type="
      column
      length= 16 />
</hibernate-mapping package="org.hibernate.auction.model"
<class name="User" table="">
  <id name="id"
    type="long"
    column="USER_ID"
    access="field">
    <generator class="
  </id>
```



- city String - Address
- street String - Address
- zipcode String - Address

- BANK_ACCOUNT
- BID
- CATEGORY
- CATEGORY_ITEM
- CREDIT_CARD
- DUAL_HIBERNATE_SEQUENCE
- ITEM
- ITEM_IMAGES
- USERS

HibernateTools – HQL-Editor

The screenshot displays the HibernateTools HQL-Editor interface. The main editor window contains the following HQL query:

```
select description, price, numberavailable
from Product p
where description like :description
and price between :minPrice and :maxPrice
```

To the right of the editor is the 'Query Parameters' panel, which lists the parameters used in the query:

Name	Type	Value
minPrice	double	10.0
description	string	%Mouse%
maxPrice	double	1000.0

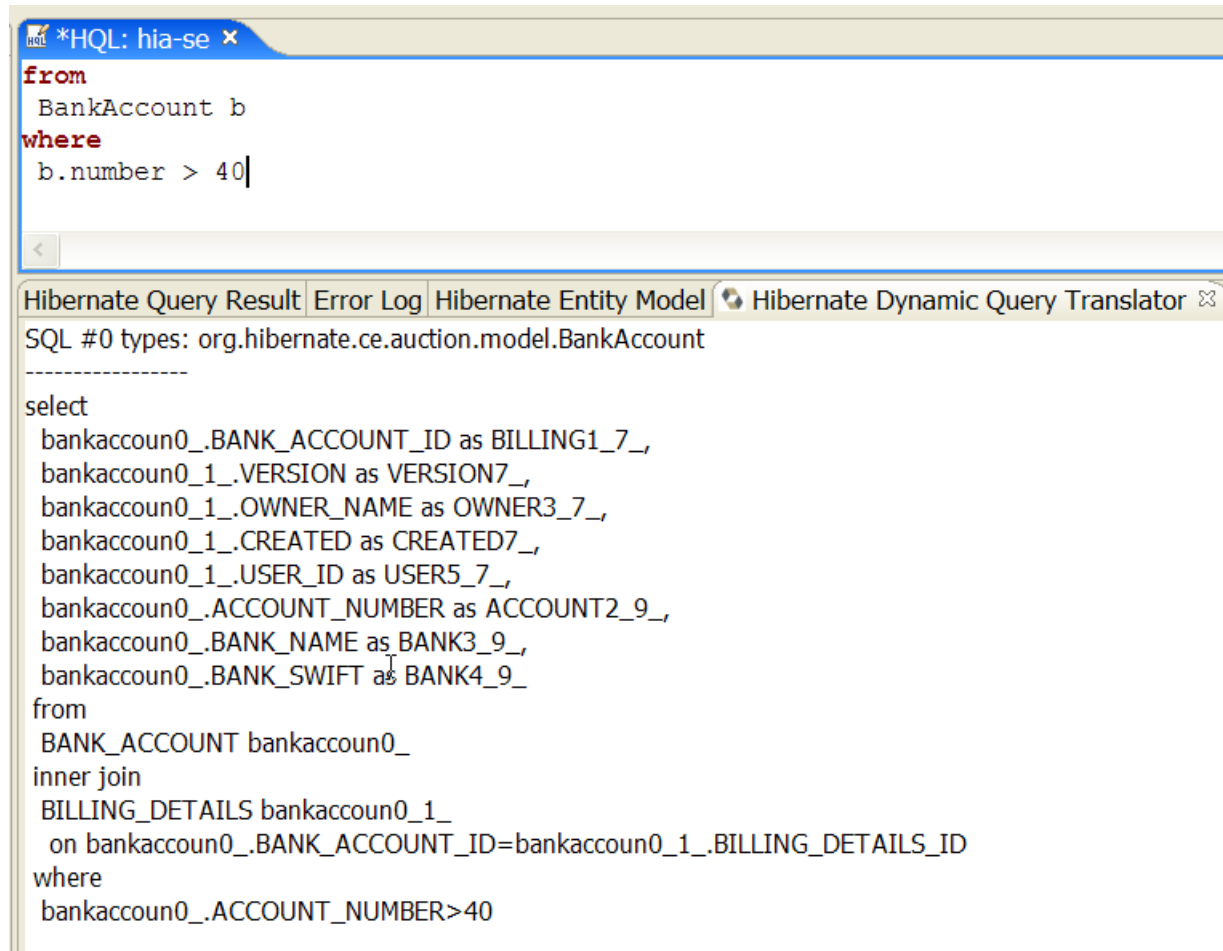
Below the parameters panel, the 'Format' is set to 42.42.

At the bottom of the interface, the 'Hibernate Query Result' panel shows the results of the query:

```
select description, price, numberavailable from Product p where description like :description and price between :n
```

0	1	2
My Mouse	101.0	23.0

SQL-Preview im HQL-Editor



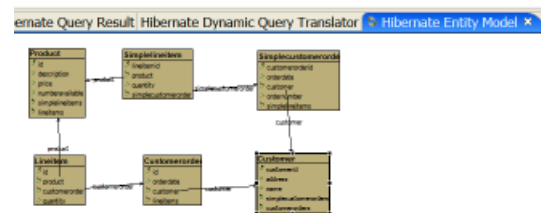
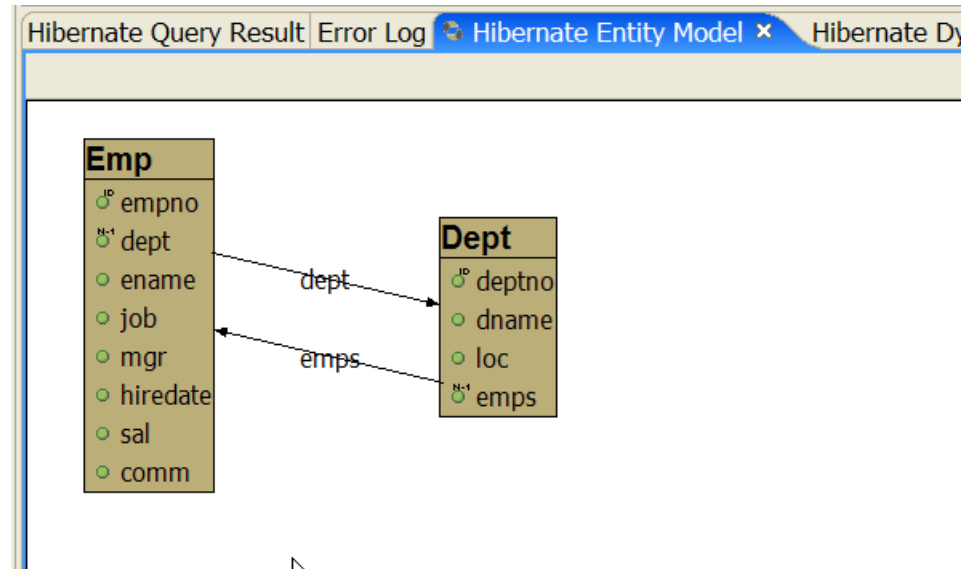
The screenshot shows a software interface with a top tab labeled '*HQL: hia-se'. The main text area contains the following HQL query:

```
from
  BankAccount b
where
  b.number > 40
```

Below the editor, there are tabs for 'Hibernate Query Result', 'Error Log', 'Hibernate Entity Model', and 'Hibernate Dynamic Query Translator'. The 'Hibernate Dynamic Query Translator' tab is active and displays the following SQL query:

```
SQL #0 types: org.hibernate.ce.auction.model.BankAccount
-----
select
  bankaccoun0_.BANK_ACCOUNT_ID as BILLING1_7_,
  bankaccoun0_1_.VERSION as VERSION7_,
  bankaccoun0_1_.OWNER_NAME as OWNER3_7_,
  bankaccoun0_1_.CREATED as CREATED7_,
  bankaccoun0_1_.USER_ID as USER5_7_,
  bankaccoun0_.ACCOUNT_NUMBER as ACCOUNT2_9_,
  bankaccoun0_.BANK_NAME as BANK3_9_,
  bankaccoun0_.BANK_SWIFT as BANK4_9_
from
  BANK_ACCOUNT bankaccoun0_
inner join
  BILLING_DETAILS bankaccoun0_1_
  on bankaccoun0_.BANK_ACCOUNT_ID=bankaccoun0_1_.BILLING_DETAILS_ID
where
  bankaccoun0_.ACCOUNT_NUMBER>40
```

HibernateTools – Grafische Modellansicht





Wie mit Hibernate anfangen



Literatur

Buch sehr sinnvoll, Ref.doku knapp, wenig Beispiele

- Pro Hibernate 3 – Apress
- Hibernate, A Developers Notebook – O'Reilly
- Hibernate in Action – Wiley
- Better, Faster, Lighter Java – O'Reilly



Lernkurve

- Anfangs steil
- Später flach
- An Beispielen orientieren
- Beispiel-Applikation ansehen
- Ein Hibernate-Kenner in der Nähe spart Zeit und Nerven



Typische Stolpersteine

- Legacy-Schemas (ungewöhnliche Konstruktionen)
- Zusammengesetzte, natürliche Schlüssel (vs. technische Schlüssel)
- Detached Objects
- Session-Lebenszeit



EJB3



EJB3

- In EJB3 ist die Persistenz auch ohne App.Server nutzbar - javax.persistence
- EJB3 orientiert sich bei der Persistenz an Hibernate
- Gavin King sitzt in der Expert Group
- Hibernate 3 implementiert EJB3 Draft API
- Hibernate 3 lernen hilft später bei EJB3



Fragen...

Kontakt

post@stefanwille.com

www.stefanwille.com