EclipseLink: JPA 2.0 und noch mehr

Berthold Maier
Chef Architekt
Oracle Deutschland GmbH

Michael Bräuer
Systemberater
Oracle Deutschland GmbH

JAVAFORUM stuttgart

eclipse link

Ziele

- Was verbirgt sich hinter EclipseLink?
- Wie kann ich es einsetzen?

JAVAFORUM stuttgart

eclipse link

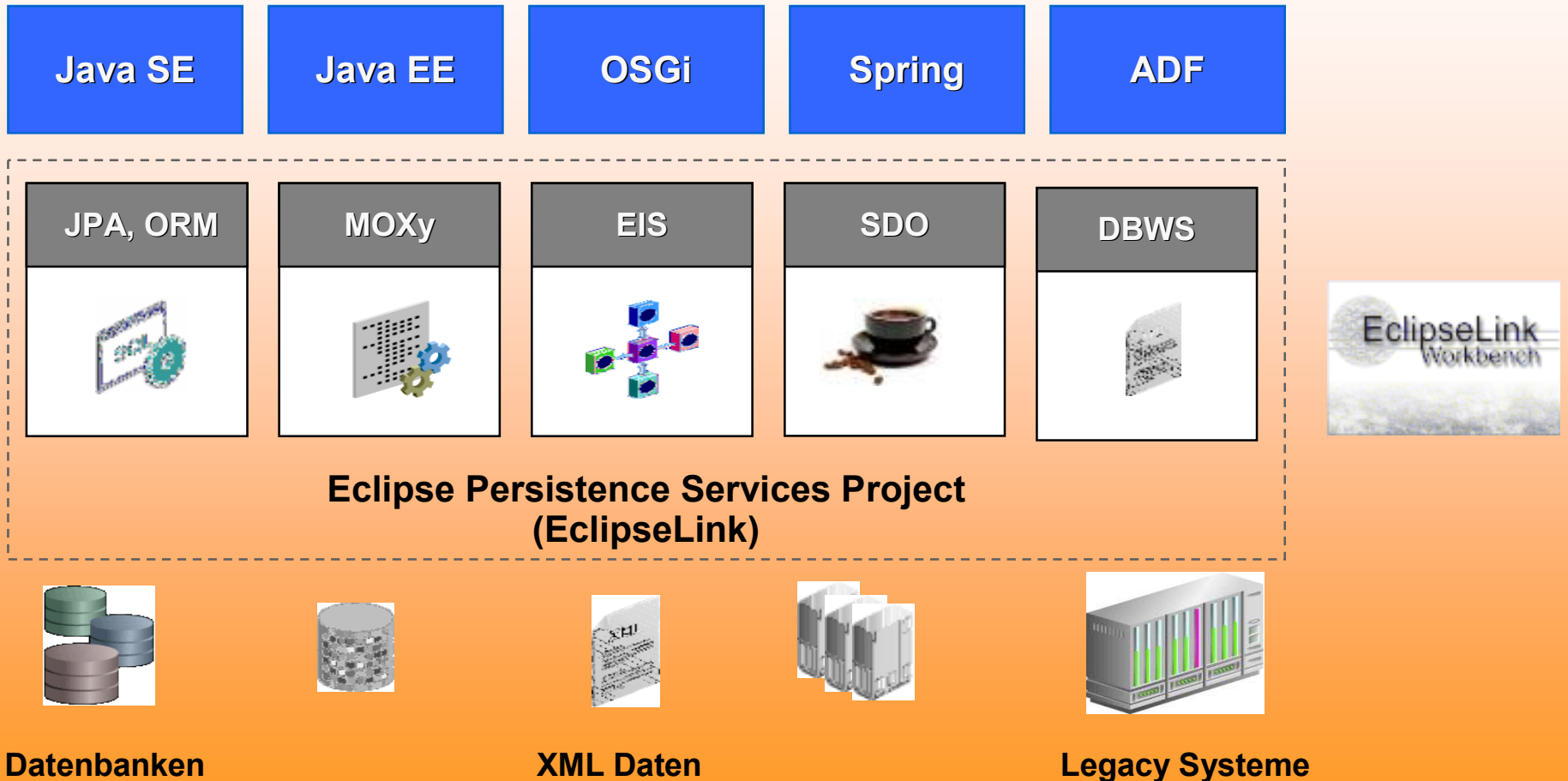
Agenda

- Was ist EclipseLink?
  - EclipseLink ORM
- JPA mit EclipseLink
  - EclipseLink JPA Extensions
  - Ausblick JPA 2.0
- EclipseLink – mehr als ein ORM
  - SDO
  - DBWS
- Fragen und Antworten

Was ist EclipseLink?

- Basiert auf 12 Jahren kommerzieller Benutzung: Oracle TopLink
- Eclipse Runtime Project: Eclipse Persistence Service Project == EclipseLink
- Projektziel: Umfassendes Persistenz-Framework – ORM ist ein Aspekt

Was ist EclipseLink?



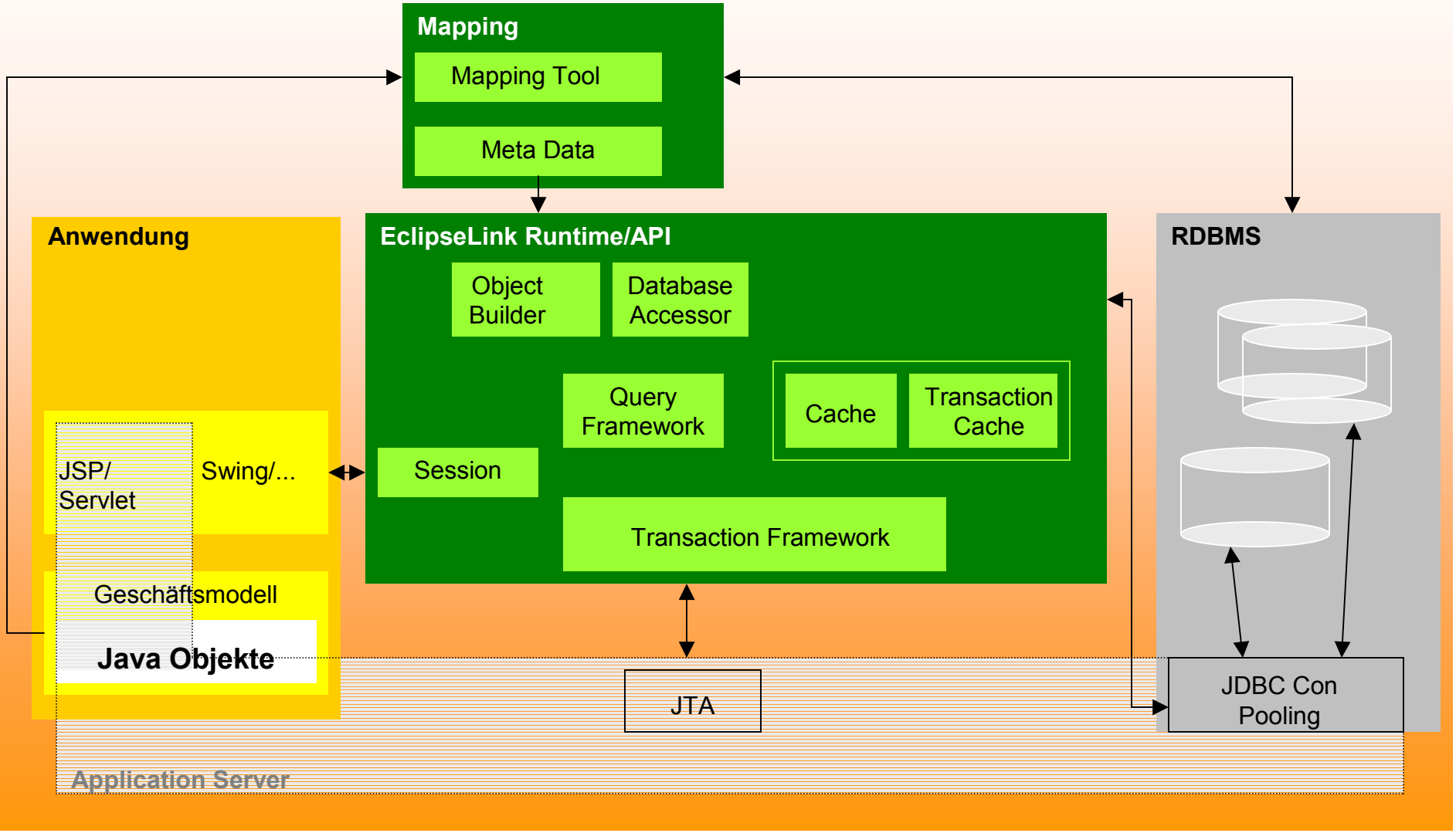
JAVAFORUM stuttgart

eclipse link

Was ist EclipseLink?

- Vollständige Open Source Persistenzlösung:
  - EclipseLink JPA/ORM: Objekt-Relational
  - EclipseLink MOXy: Objekt-XML
  - EclipseLink SDO: Service Data Objects
  - EclipseLink DBWS: Datenbank Web Services
  - EclipseLink EIS: Nicht-Relationale Persistenz, benutzt JCA
- Ausnutzung von Synergien, z.B. gemeinsames Domänenmodell für die Nutzung unterschiedlicher Persistenztechnologien

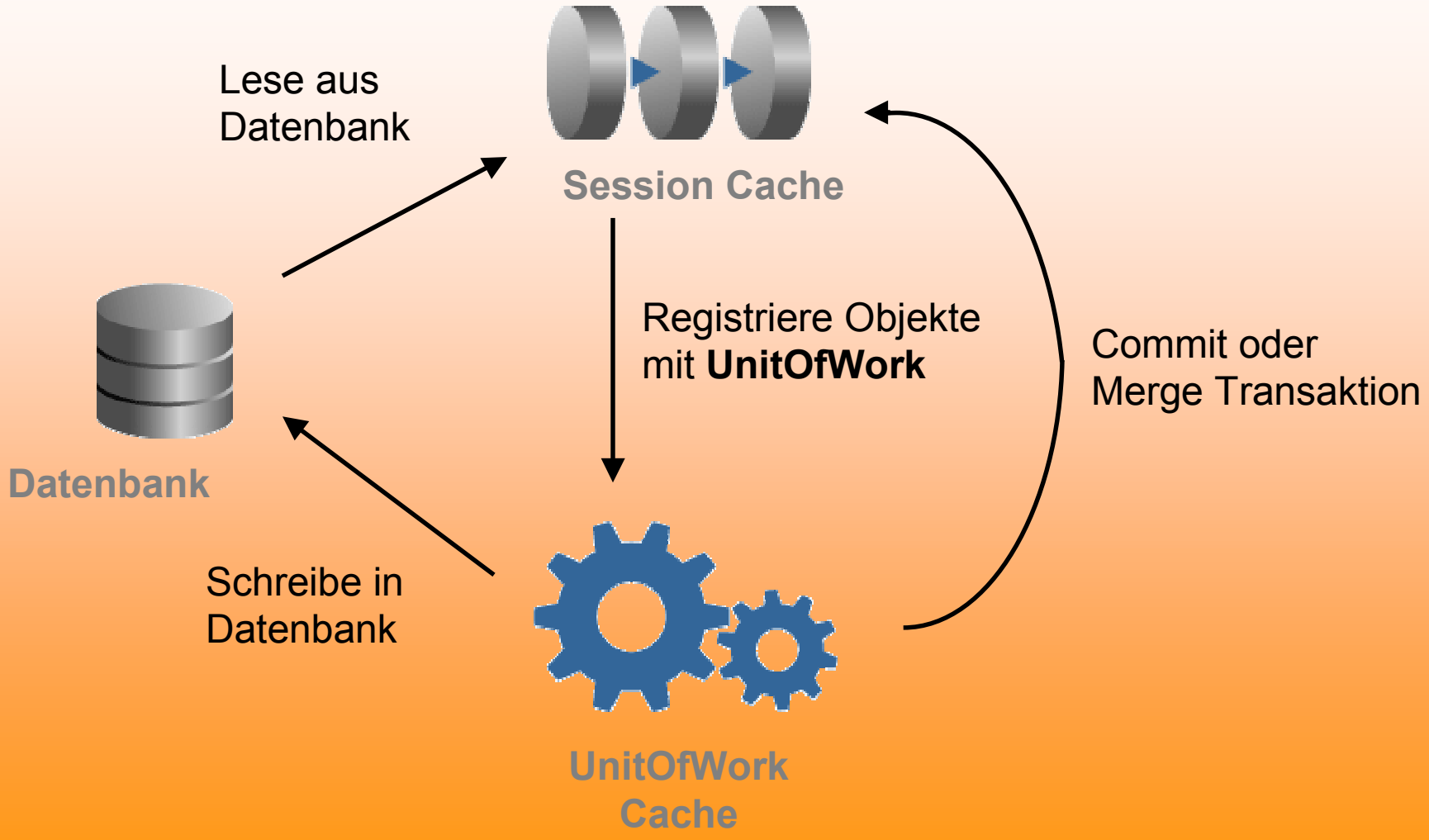
EclipseLink ORM



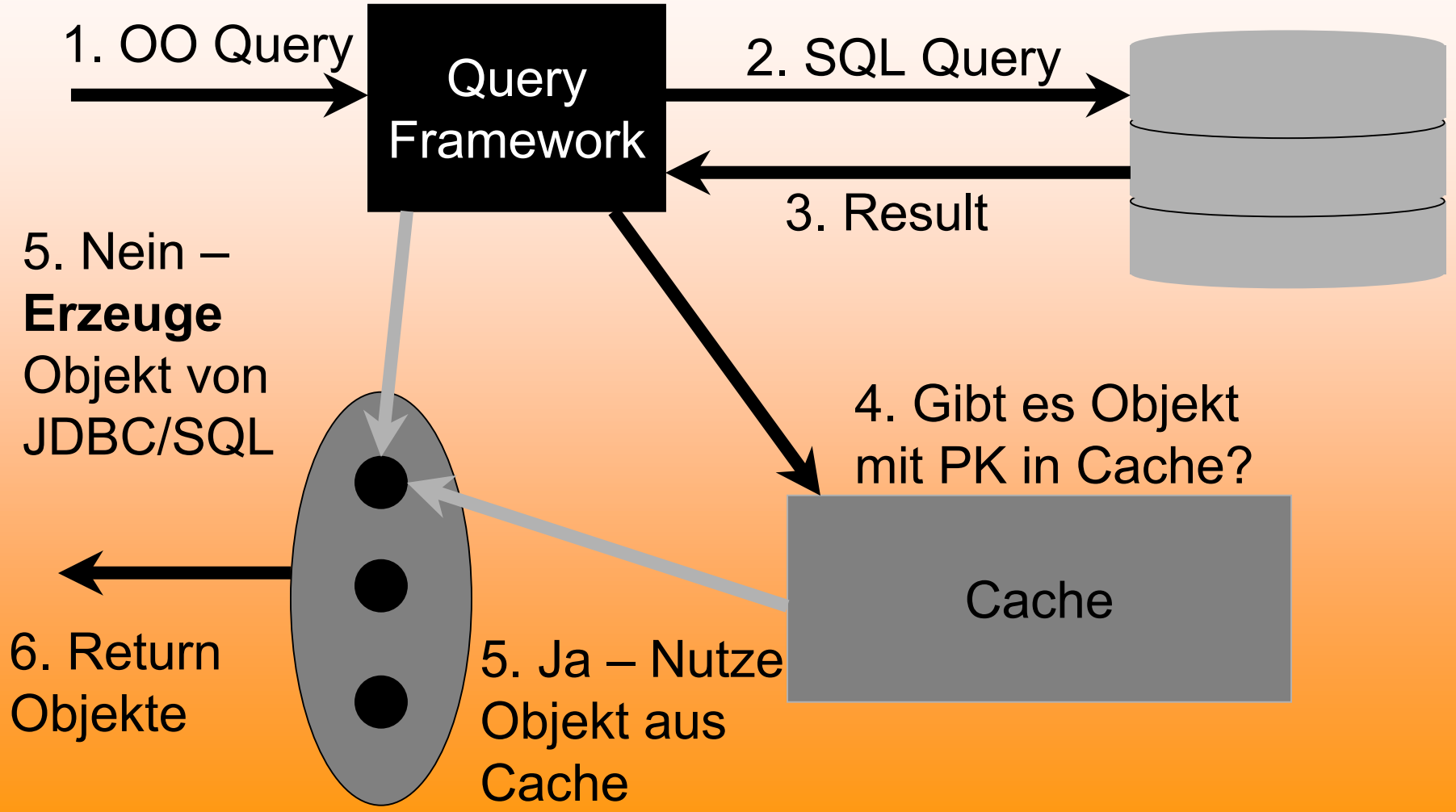
EclipseLink ORM

- Zugriff auf API via Sessions:
  - Server Session
  - Client Session → UnitOfWork Session
- Session haben Caches
  - Shared/Isoliert
- UnitOfWork Session:
  - Transaktionale Einheit
  - EclipseLink ermittelt minimale Änderungsmenge (Change Tracking)

EclipseLink ORM



EclipseLink ORM



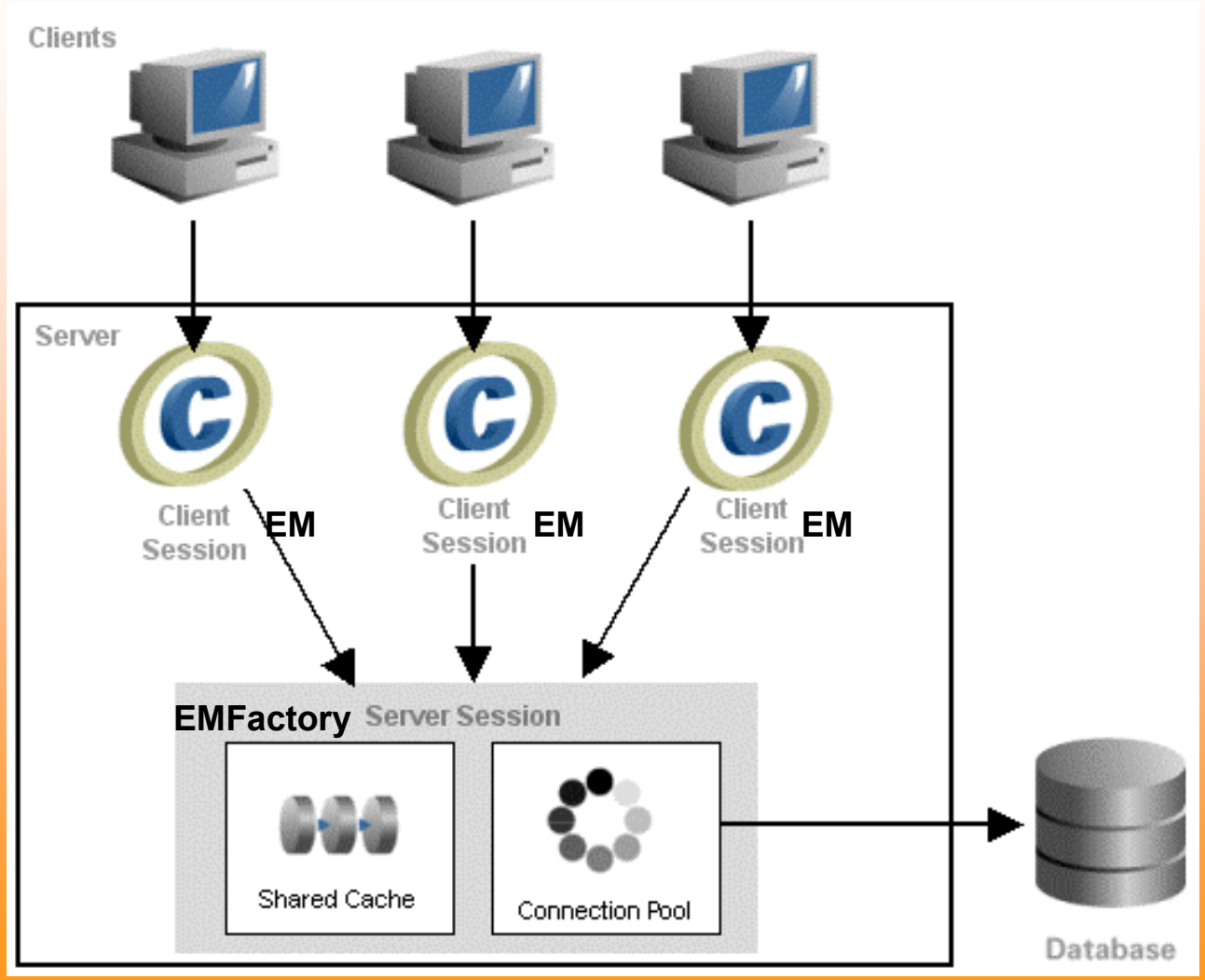
JAVAFORUM stuttgart

eclipse link

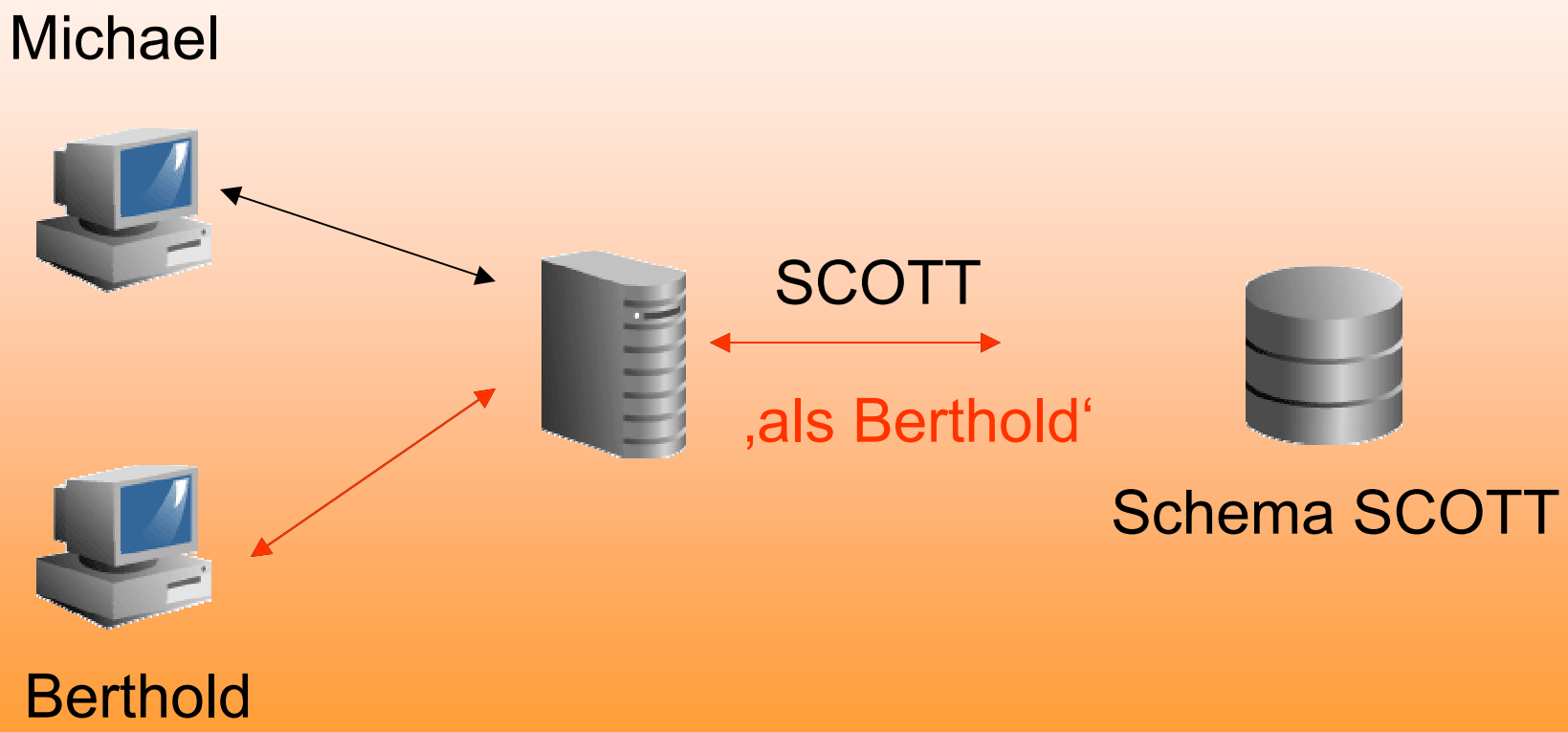
JPA mit EclipseLink

- JPA 1.0 Implementierung
  - Vielfältige Plattformunterstützung: Datenbank, Application Server, Frameworks
  - Basiert auf EclipseLink ORM: EclipseLink spezifische API, Expression/Query Language, Mappings
- Erweiterungen: EclipseLink JPA Extensions
  - jetzt verfügbar
- JPA 2.0 RI

Zusammenhänge: ORM - JPA



Exkurs: Proxy Authentifizierung



Exkurs: Proxy Authentifizierung

```
Map emProperties = new HashMap();
emProperties.put("eclipselink.oracle.proxy-type",
   OracleConnection.PROXYTYPE_USER_NAME);
emProperties.put(OracleConnection.PROXY_USER_NAME, "BERTHOLD");

EntityManager em = getEMF().createEntityManager(emProperties);
```

- Funktionaler Benutzer verbindet sich als Endbenutzer Berthold
- Unterstützt End-zu-End Sicherheitsaspekte (z.B. Auditing)
- Neben Oracle Proxy Authentifizierung auch Erweiterungen für Oracle VPD möglich

EclipseLink JPA Extensions

- Zusätzliche Mappings
  - @BasicCollection (→ JPA 2.0 ElementCollection)
  - @BasicMap (→ JPA 2.0 ElementCollection)
  - @PrivateOwned
  - @JoinFetch
  - @Converter
  - @Transformation, …

JAVAFORUM stuttgart

eclipse link

EclipseLink JPA Extensions

- Erweiterungen für Anpassungen und Optimierungen
  - @Customizer, DescriptorCustomizer, SessionCustomizer
  - Properties für weaving, change tracking, read only
- Returning Policy: Rückgabe von Werten aus der DB, z.B. bei Verwendung von DB Triggern, welche Werte ändern
  - @ReturnInsert, @ReturnUpdate

JAVAFORUM stuttgart

eclipse link

EclipseLink JPA Extensions

- Erweiterungen für Caching
  - Shared (L2)/Isoliert and Persistence Context Caching
    - Entity Cache — kein Datencache
  - Cache Invalidation/Expiration
    - Time to live
    - Fixed Times
    - Programmierbar (Externe Benachrichtigung)
  - Cache Coordination
  - Konfigurierbar per **Entität**

EclipseLink JPA Extensions

- Erweiterungen für Locking:
  - Optimistic Locking mittels @OptimisticLocking Annotation (u.a. ALL_COLUMNS, CHANGED_COLUMNS, SELECTED_COLUMNS)
  - Pessimistisches Locking (→ JPA 2.0)

EclipseLink JPA Extensions

- Kombination von EclipseLink nativen Eigenschaften, z.B. gemeinsame Benutzung von Expressions und JPQL:

```
ExpressionBuilder prt = new ExpressionBuilder();
Expression expr =
    prt.get("Event").get("Name").equal("JFS");

Query query =
((org.eclipse.persistence.jpa.JpaEntityManager)
em.getDelegate()).
          createQuery(expr, Teilnahme.class);
```

- Konsequenz: 3 Arten von Queries (JPQL, Mix mit Expression API, Criteria API (→ JPA 2.0) m/o Typsicherheit)

EclipseLink JPA Extensions

- Query Hints:
  - eclipselink.cache-usage
  - eclipselink.jdbc.bind-parameters
  - eclipselink.pessimistic-lock
  - eclipselink.refresh
  - eclipselink.batch
  - eclipselink.join-fetch
  - eclipselink.jdbc.timeout
  - eclipselink.jdbc.fetch-size
  - eclipselink.result-collection-type
  - eclipselink.query-type
  - ...

JAVAFORUM stuttgart

eclipse link

EclipseLink JPA Extensions

- Erweiterungen für Stored Procedures (@NamedStoredProcedure)
- DDL Generierung
- eclipselink-orm.xml

JAVAFORUM stuttgart

eclipse link

Ausblick JPA 2.0

- EL ist RI
- Neben den hier aufgezählten Features existieren noch andere
- Status:http://wiki.eclipse.org/EclipseLink/Development/JPA_2.0

Ausblick JPA 2.0

- Mapping
  - Access Type
  - Nested Embeddable
  - Flexible Maps und Collections
    - Collections (Basic, Embeddable)
    - Maps mit Key/Value (Basic Types), Entity oder Embeddable
  - Derived Identifiers
  - Orphan Removal
  - Unidirektionales 1:M
  - Persistent Ordered Lists

Ausblick JPA 2.0

- Pessimistisches Locking
- Cache API
- Query
  - JP QL Erweiterungen
    - Inheritance Filtering (durch Typ)
    - Collection Parameters für IN
    - CASE
- Andere
  - TABLE_PER_CLASS Strategie (JPA 1.0 – optional) für Vererbung

JAVAFORUM stuttgart

eclipse link

Ausblick JPA 2.0

- Mapping
  - Embeddable Association Override
- Query
  - JP QL Erweiterungen
    - Map Zugriff (Key, Value oder Eintrag)
    - …
  - Criteria API
- Meta Model API
- Validation
- Standard Properties
- Cache Usage Settings

JAVAFORUM stuttgart

eclipse link

Ausblick JPA 2.0

- Typesafe MetaModel Generation
- 1:1 JoinTable
- Schema/Catalog in SequenceGenerator
- Delimited Identifier

JAVAFORUM stuttgart

eclipse link

EclipseLink SDO

- Service Data Object – SDO 2.1
- Was kann man damit machen?
  - Generierung von Java SDO Klassen (statisch) auf Basis eines XML Schemas
  - Generierung von SDO Typen und Properties zur Laufzeit auf Basis eines XML Schemas (dynamisch)
  - Generierung von XML Schema, welches den Typen und Properties eines SDO Models entspricht

EclipseLink DBWS

- Was kann man damit machen?
  - Erzeugung von JAX-WS 2.0 Web Services basierend auf:
    - CR(findByPK,findAll)UD auf Tabellen
    - SQL
    - Stored Procedure
- Verwendung innerhalb von SOA

EclipseLink Distributionen

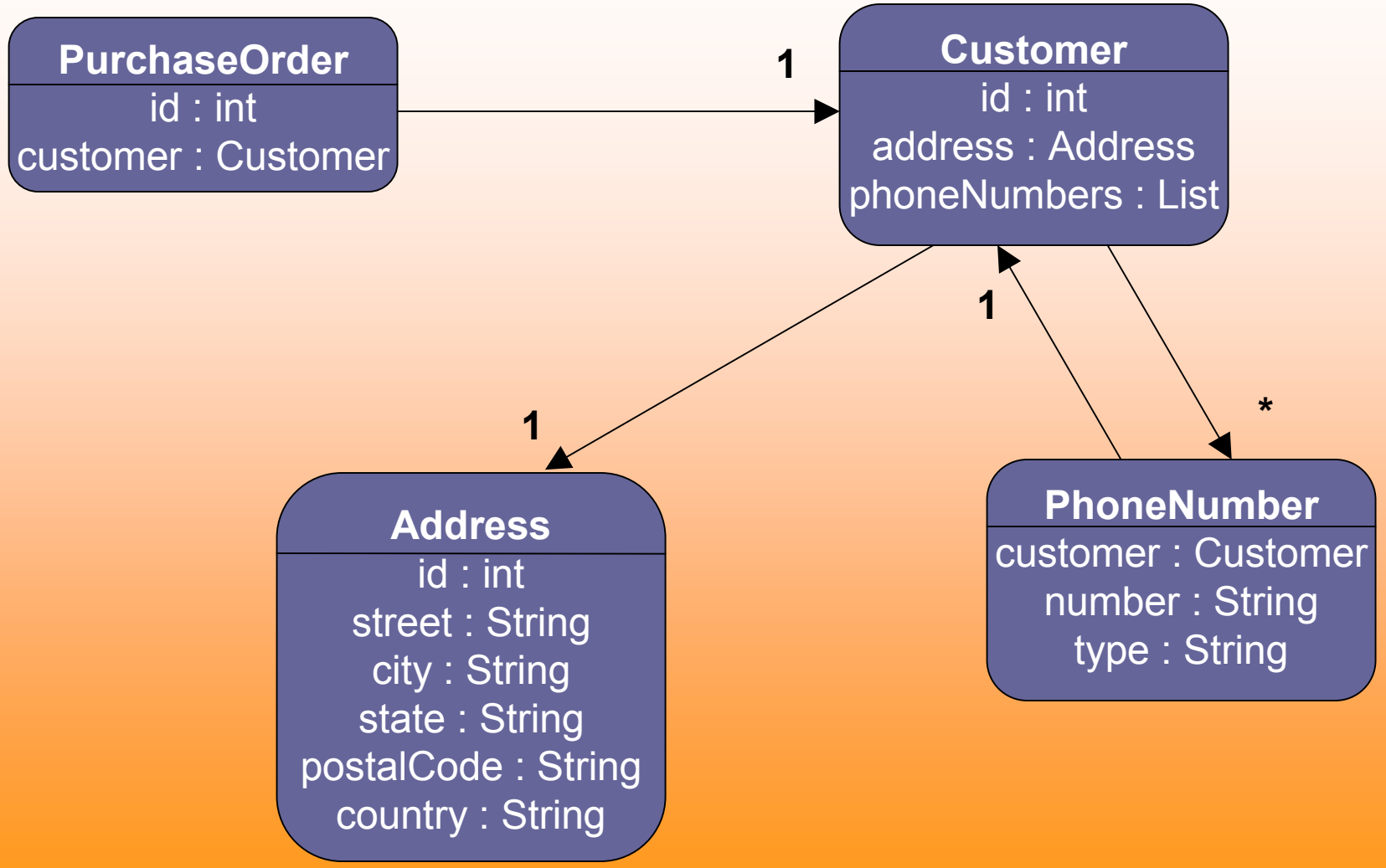
- Eclipse.org:
  - Installer ZIP/Source ZIP
  - OSGi Bundles
  - Maven Repository
  - SVN: http://dev.eclipse.org/svnroot/rt/org.eclipse.persistence/trunk
- Oracle: Oracle TopLink, Oracle WebLogic
- GlassFishV3:
  - EclipseLink ersetzt TopLink Essentials
- Spring Source:
  - Spring Framework
  - Spring OSGi Bundle Repository

JAVAFORUM stuttgart

eclipse link

Anhang

Laden des Objektgraphen – Bsp.



Aufgabe

- Suche alle aktiven Bestellungen eines Kunden aus San Francisco
- Hole dann zu jeder PO zusätzliche Informationen, wie Kundenadresse, Telefon

```
Query query =
    em.createQuery("SELECT po FROM PurchaseOrder po " +
                   "WHERE po.status = 'ACTIVE' AND " +
                   "po.customer.address.city = 'SFO'");
List<PurchaseOrder> pos = query.getResultList();

// Force graph load simulating presentation tier logic
for (PurchaseOrder po: pos) {
    po.getCustomer().getAddress();
    po.getCustomer().getPhone().size();
}
```

Ohne Optimierung

```
SELECT PO.* FROM PO, CUST, ADDR WHERE PO.STATUS =
  'ACTIVE' AND PO.CUST_ID = CUST.ID AND
  CUST.ADDR_ID = ADDR.ID AND ADDR.CITY = 'SFO'
  {Returns N Purchase Orders}                    {1}
SELECT * FROM CUST WHERE CUST.ID = 1 ...         {N}
SELECT * FROM ADDR WHERE ADDR.ID = 100 ...       {N}
SELECT * FROM PHONE WHERE PHONE.CUST_ID = 1      {N}
```

Ergebnis: 3N+1 Abfrage (100 PO's = 301)

"N+1 Query Problem"

Join Reading FK Beziehungen

- Join Reading of M:1 and 1:1

```
Query query = em.createQuery("…");
query.setHint(QueryHints.FETCH, "po.customer");
query.setHint(QueryHints.FETCH, "po.customer.address");
List<PurchaseOrder> pos = query.getResultList();
```

- SQL:

```
SELECT PO.*,CUST.*,ADDR.* FROM PO, CUST, ADDR
  WHERE PO.STATUS = 'ACTIVE' AND PO.CUST_ID =
  CUST.ID AND CUST.ADDR_ID = ADDR.ID AND
  ADDR.CITY = 'SFO'
  {1 Aufruf: N Purchase Orders mit Kunden &
  Adressen}
SELECT * FROM PHONE WHERE PHONE.CUST_ID = 1
  ...{N Aufrufe}
```

Ergebnis: N+1 Abfragen (100 PO's = 101 SQL)

Join Reading aller Beziehungen

- Join Reading

```
Query query = em.createQuery("…");
query.setHint(QueryHints.FETCH, "po.customer");
query.setHint(QueryHints.FETCH, "po.customer.address");
query.setHint(QueryHints.FETCH, "po.customer.phones");
List<PurchaseOrder> pos = query.getResultList();
```

- SQL:

```
SELECT PO.*,CUST.*,ADDR.*, PHONE.*
FROM PO, CUST, ADDR, PHONE
WHERE PO.STATUS = 'ACTIVE' AND PO.CUST_ID =
  CUST.ID AND CUST.ADDR_ID = ADDR.ID AND
  ADDR.CITY = 'SFO' AND PHONE.CUST_ID = CUST.ID
{1: Liefert N Purchase Orders mit Kunden,
  Addressen und Telefonnummern}
```

Ergebnis: 1 Abfrage (100 PO's = 1 SQL)

Batch und Join Reading

- Auswahl pro Kind mit Join zur ursprüngl. Klausel

```
Query query = em.createQuery("…");
query.setHint(QueryHints.FETCH, "po.customer");
query.setHint(QueryHints.FETCH, "po.customer.address");
query.setHint(QueryHints.BATCH, "po.customer.phones");
List<PurchaseOrder> pos = query.getResultList();
```

- SQL:

```
SELECT PO.*,CUST.*,ADDR.* FROM PO, CUST, ADDR WHERE PO.STATUS =
   'ACTIVE' AND PO.CUST_ID = CUST.ID AND CUST.ADDR_ID = ADDR.ID
   AND ADDR.CITY = 'SFO'
   {Liefert N Purchase Orders mit Kunden & Adressen}

SELECT PHONE.* FROM PHONE, PO, CUST, ADDR WHERE PO.STATUS =
   'ACTIVE' AND PO.CUST_ID = CUST.ID AND CUST.ADDR_ID = ADDR.ID
   AND ADDR.CITY = 'SFO' AND PHONE.CUST_ID = CUST.ID
```

Ergebnis: 2 Abfragen (100 PO's = 2 SQL)

Nur Batch

- Query:

```
Query query = em.createQuery("…");
query.setHint(QueryHints.BATCH, "po.customer");
query.setHint(QueryHints.BATCH, "po.customer.address");
query.setHint(QueryHints.BATCH, "po.customer.phones");
List<PurchaseOrder> pos = query.getResultList();
```

```
SELECT PO.* FROM PO, CUST, ADDR WHERE PO.STATUS = 'ACTIVE' AND PO.CUST_ID
   = CUST.ID AND CUST.ADDR_ID = ADDR.ID AND ADDR.CITY = 'SFO'
SELECT CUST.* FROM PO, CUST, ADDR WHERE PO.STATUS = 'ACTIVE' AND
   PO.CUST_ID = CUST.ID AND CUST.ADDR_ID = ADDR.ID AND ADDR.CITY = 'SFO'
SELECT ADDR.* FROM PO, CUST, ADDR WHERE PO.STATUS = 'ACTIVE' AND
   PO.CUST_ID = CUST.ID AND CUST.ADDR_ID = ADDR.ID AND ADDR.CITY = 'SFO'
SELECT PHONE.* FROM PHONE, PO, CUST, ADDR WHERE PO.STATUS = 'ACTIVE' AND
   PO.CUST_ID = CUST.ID AND CUST.ADDR_ID = ADDR.ID AND ADDR.CITY = 'SFO'
   AND PHONE.CUST_ID = CUST.ID
```

- SQL:

  Ergebnis: 4 Abfragen (100 PO's = 4 SQL)

Ergebnisse

	# SQL	Time
Default	301	200 ms
Join 1:1's	101	150 ms
Join All	1	60 ms
Batch All	4	40 ms
Batch & Join	2	20 ms

Quelle: Einfache Testumgebung, Ergebnisse u.a. abhängig von #Attributen und #Sätzen

JAVAFORUM stuttgart

eclipse link

Fragen und Antworten