SQL Virtual Foreign Keys Simplify Hibernate Access to Your Existing CA IDMS™ Databases

Jean-Paul De Feyter
CA Technologies

IUA/CA IDMS™ Technical Conference
May 16-20, 2016

Abstract

- Users need to enhance developer productivity as they leverage and extend their investment in CA IDMS. Hibernate is a popular open source object to relational mapping framework for developing Java applications that access and store objects in relational databases. This session shows how you can use SQL Virtual Foreign Key feature introduced in CA IDMS 19.0 with CA IDMS Server to improve developer productivity by using Hibernate reverse engineering to generate Java objects that access and update records in your existing network databases.

Agenda

- Object-Relational Mapping Concepts
- Java Persistence API
- Relational-Network Mapping
- Sample JPA access to Employee database
- Hibernate HQL Demo

Object-Relational Mapping Concepts

What is object-relational mapping?

- From wikipedia.org:
  - "a programming technique for converting data between incompatible type systems in object oriented programming languages"
  - "virtual object database" used within the programming language
- Persistence
  - Objects stored, somewhere
  - Serialization
  - Database

Why use object-relational mapping?

- Programmer concentrates on business logic
- Programmer works with application objects
- Provider takes care of persistence
- No need to code database interface calls
- Limited or no need to know and code SQL

Object-relational mapping concepts

Object (Java)

- Class
- Object
- Attribute
- Relationship

Relational (SQL)

- Table
- Row
- Column
- Referential constraint

Object-relational mapping software components

- Provider run time
  - Generates SQL
  - Reflection
  - Mapping definitions
- Object definition tools
  - Schema definition
  - Reverse engineering

Schema generation

- Automatically generates DB schema from objects
- Most useful for prototyping DB
- Physical tuning always manual
- Over-reliance on ORM can lead to poor DB design
- DBA should do final design

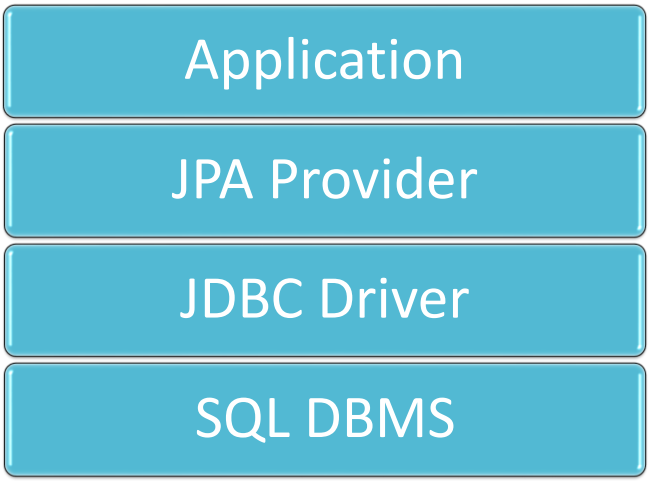
Reverse engineering

- Create object definitions from database
- Most application databases already exist
- Not biased toward a single application
- Most ORM frameworks provide reverse engineering tool
- Uses database metadata API's to discover
  - Entities
  - Attributes
  - Relationships

JPA Java Persistence API

- Application Programming Interface
  - Defined in Java 5 SE and EE
  - javax.persistence package
- JPA specification came as part of EJB 3.0
- Service Provider Interface (SPI)
- Providers
  - Hibernate (Jboss, Red Hat)
    - also has own API
  - OpenJPA (known as Kodo, BEA, Oracle)
  - TopLink (Oracle)
  - Others

JPA architecture



JPA entities

- Entity
  - Represents application object
  - May represent database table
  - POJO
- EntityManager
  - Manages state and life cycle of entity
    - Persist
    - CreateEntityGraph
    - Remove
    - Find (uses primary key)
    - Query
    - Transaction

Reflection and annotations
How JPA works

- Reflection
  - Discover classes, fields, methods in code
  - Depends on coding conventions (get, set, etc.)
- Annotations
  - Metadata in code about classes, fields, methods
  - Relate Java objects to database tables
  - Language feature introduced in J2SE 5
  - @<name>(optional arguments)
  - Extensive use of defaults
  - Alternative to XML descriptor files

JPA annotations

- @Entity
- @Table
- @Column
- @Id
- @OneToMany
- @ManyToMany
- @Inheritance
- Many more…

JPA and CA IDMS SQL databases

- CA IDMS is like most other relational databases
- Schema generation
  - With CA IDMS 19.0 IR3 or later
    - DDL standard, including referential constraints
- Most CA IDMS databases are not SQL defined
  - Reverse engineering also complete for network DBs with CA IDMS 19.0 IR3 Virtual Key schema

Relational-network mapping

Relational-network mapping

Relational (SQL)	Network (CA IDMS)
Table	Record definition
Row	Record occurrence
Column	Field
Referential constraint	Set

JPA and CA IDMS Network databases

- Includes most CA IDMS applications
- Access via SQL
- With CA IDMS 19.0 IR3 or later
  - Reverse engineering
  - Equivalent SQL Schema generation
- Before CA IDMS 19.0 IR3
  - Reverse engineering
    - requires customization
  - "Impedance mismatch"
    - Elements
    - Sets without primary/foreign key specification

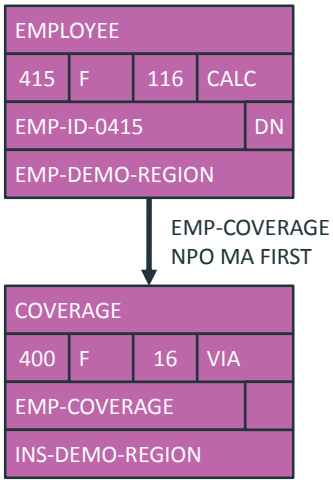
Relational-network mapping
Techniques to overcome impedance mismatch

- Not needed if using Virtual Foreign Keys in CA IDMS 19.0 IR3
  - Syntax extensions
  - Views
  - Table procedures
  - Embedded Virtual foreign keys

Sample JPA access to Employee database

JPA example on network defined DB using VFK schema

- Hibernate reverse engineering
  - Entity classes generated for all records & sets of EMPDEMO
    - EMPSCHM WITH VIRTUAL KEYS
    - No manual modifications
    - With annotations
- Employee database
  - EMPLOYEE
  - COVERAGE
- Entity classes
  - Employee
  - Coverage



Reverse engineered Employee class has primary key ROWID and set Coverage …

```
@Entity
@Table(name = "EMPLOYEE")
public class Employee implements java.io.Serializable
{
   private byte[] rowid;
   private short  empId;
   private String empFirstName;
   private Set<Coverage> coverages =
                              new HashSet<Coverage>(0);
   // remaining private member vars for each column
   public Employee() {}
   ...
   @Id
   @Column(name = "ROWID")
   public byte[] getRowid() {
         return this.rowid;
   }
```

… and is related to Coverage in a OneToMany relationship

```
    ...
    @OneToMany(fetch = FetchType.LAZY, mappedBy =
                                              "employee")
    public Set<Coverage> getCoverages() {
       return this.coverages;
    }

    public void setCoverages(Set<Coverage> coverages) {
       this.coverages = coverages;
    }

    // access methods for each member variable...

}
```


Reverse engineered Coverage class contains reference to owning Employee class …

```
@Entity
@Table(name = "COVERAGE")
public class Coverage implements
java.io.Serializable {
   private byte[] rowid;
   private Employee employee;
   private byte selectionYear0400;
   // private member variables for each column...

   public Coverage() {}
   @Id
   @Column(name = "ROWID")
   public byte[] getRowid() {return this.rowid;}
   public void setRowid(byte[] rowid) {
                            this.rowid = rowid;}
```

… and is related with employee in ManyToOne relationship

```
@ManyToOne(fetch = FetchType.LAZY)
   @JoinColumn(name = "FKEY_EMP_COVERAGE")
   public Employee getEmployee() {
      return this.employee;
   }

   public void setEmployee(Employee employee) {
      this.employee = employee;
   }

   @Column(name = "SELECTION_YEAR_0400"
       , nullable = false, precision = 2, scale = 0)
   public byte getSelectionYear0400() {
      return this.selectionYear0400;
   }
// access methods for each member variable...
}
```

27

Not using VFK schema requires modifying Employee class …

- Add reference to member object Coverage
- Use set specification instead of foreign key

```
@SqlResultSetMapping(
   name = "EmpCoverageResult", entities = {
   @EntityResult(entityClass=Coverage.class)})

@NamedNativeQuery(
   name="GetEmpCoverage",
   query="SELECT c.ROWID, c.* FROM " +
   "EMPSCHM.EMPLOYEE e, EMPSCHM.COVERAGE c " +
   "WHERE EMP_ID_0415 = :empID " +
   "AND \"EMP-COVERAGE\"",
   resultSetMapping="EmpCoverageResult")
```

28 IUA/CA IDMS™ Technical Conference

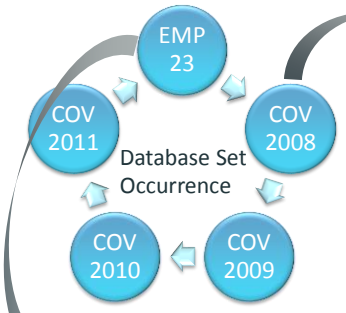
… and modifying Coverage class …

- Add ROWID as primary key
- Add reference to owner object

… and populating set occurrence objects in business logic

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("NonSqlJPA");
EntityManager em = emf.createEntityManager();

Employee e = em.getReference(Employee.class, 23);

Query q = em.createNamedQuery("GetEmpCoverage");
q = q.setParameter(1, 23);
List<Coverage> l = List<Coverage>)q.getResultList();
e.setCoverage(l);

Iterator<Coverage> ci = e.getCoverage().iterator();
while (ci.hasNext()){
   Coverage c = ci.next();
   c.setEmployee(e);
}
```

Set occurrence mapped to objects



Named Query Result Set

c.ROWID	c.YEAR	c.TYPE
08010202…08	2008	F
08010204…08	2009	F
08010210…08	2010	M
08010303…08	2011	F

Java Objects

```
Employee e = {0920…08, 23, Joe, …}
List<Coverage> coverage = {08010202…08, e, 2008, F, …}
                          {08010204…08, e, 2009, F, …}
                          {08010210…08, e, 2010, M, …}
                          {08010303…08, e, 2011, F, …}
```


Sample business logic
Update member set

```
Employee e;                 // e is an EMPLOYEE object
...                         // Code to retrieve e
...
EntityTransaction tx = em.getTransaction();
tx.begin();

Iterator<Coverage> ci = e.getCoverage().iterator();
while (ci.hasNext()) {
   Coverage c = ci.next();
   if (c.getType() == 'M') {
      em.lock(c, LockModeType.PESSIMISTIC_WRITE);
      c.setType0400('F');
   }
}
tx.commit();
```

Update implementation

- Varies by provider and DBMS
  - versioning
- Concurrency and locking
  - Optimistic
  - Pessimistic

Optimistic locking

- Supported by Hibernate
- Checks all columns for changes

```
@org.hibernate.annotations.Entity(
     dynamicUpdate = true,
     optimisticLock =
org.hibernate.annotations.OptimisticLockType.ALL)

UPDATE EMPSCHM.COVERAGE SET TYPE_0400=?
   where ROWID=?
      AND INS_PLAN_CODE_0400=?
      AND SELECTION_DAY_0400=?
      AND SELECTION_MONTH_0400=?
      AND …
```

Pessimistic locking

- Use with JPA
- Uses positioned update, sort of

```
em.lock(c, LockModeType.PESSIMISTIC_WRITE);
c.setType0400('M');         // update one column
tx.commit();

SELECT T0.ROWID FROM EMPSCHM.COVERAGE T0
   WHERE T0.ROWID = ? FOR UPDATE

UPDATE EMPSCHM.COVERAGE SET TYPE_0400 = ?
   WHERE ROWID = ?
```


Sample business logic
Delete

- The remove object method generates SQL to delete the row:

```
em.remove(c);

DELETE FROM EMPSCHM.COVERAGE WHERE ROWID = ?

em.remove(e);

DELETE FROM EMPSCHM.EMPLOYEE WHERE ROWID = ?
```

Insert member

- Usually need (virtual) foreign keys
- Alternative is use of a procedure

Sample business logic
Insert Coverage member

```
em.getTransaction().begin();
Query query = em.createQuery("select emp FROM
      Employee emp where emp.empId415 = :empID");
query.setParameter("empID", 23);
List<Employee> arr_cov = (List<Employee>)
                              query.getResultList();
Employee emp = (Employee)arr_emp.iterator().next();

Coverage coverage = new Coverage();
byte[] Rowid00 = new byte[]
            {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
coverage.setRowid(Rowid00);
coverage.setType0400('F');
coverage.setselectionYear(2016);
...                                 // Other attributes
coverage.setEmployee(emp);
em.persist(coverage);
em.getTransaction().commit();
```

Relational to network mapping Tradeoffs

SQL Extensions	Views	Table Procedures	Foreign Keys	Virtual Foreign Keys
– Non-standard SQL – No new programs – No application changes – No restructure – Set support limited to SELECT, UPDATE, DELETE	– Use standard SQL – No new programs – No application changes – No restructure – Set support limited to SELECT, UPDATE, DELETE	– Use standard SQL – New programs required to implement procedures – No application changes – No restructure – Full set support encapsulated in procedure DML statements	– Use standard SQL – No new programs – Limited application changes usually required – Targeted restructure usually required – Full set support as referential constraints in SQL statements	– Use standard SQL – No new programs – No application changes – No restructure – Full set support as referential constraints in SQL statements – Use of ROWID as primary key

Tips for using JPA

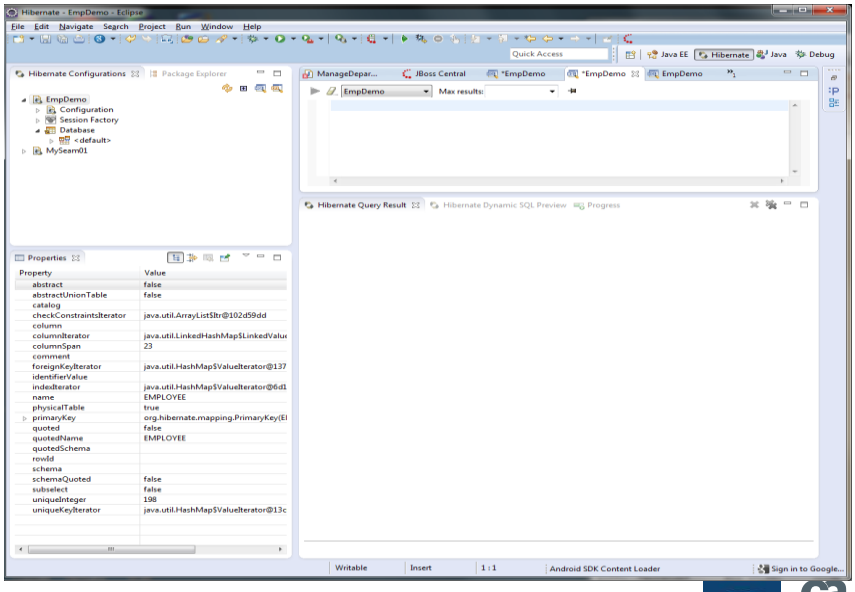
- Exceptions can be vague, use logging for details
- Use SLF4J (Simple Logging Facade for Java) logging api
  - See sample log4j.properties
- For JPA on Hibernate
  - Add *hibernate.show_sql=true* to properties file
  - SQL statements and parameter bindings are logged
- Alternatively use IDMS log facilities
  - Use Type 2 driver for debugging
  - Can use Type 4 for production
  - Enable SQL trace
  - Use ODBC Administrator

Tips for using JPA
Sample log4j.properties

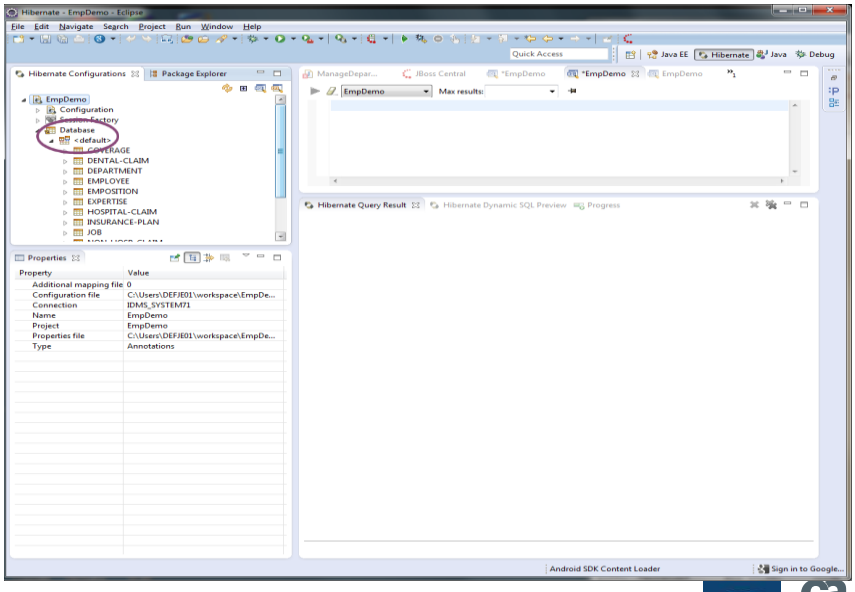
```
# Direct log messages to a log file
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=C:/tmp/Hibernate.log
log4j.appender.file.MaxFileSize=100MB
log4j.appender.file.MaxBackupIndex=1
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Direct log messages to stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

# Root logger option
log4j.rootLogger=INFO, file, stdout

# Log everything. Good for troubleshooting
log4j.logger.org.hibernate=INFO

# Log all JDBC parameters
log4j.logger.org.hibernate.type=ALL
```


Hibernate HQL Demo

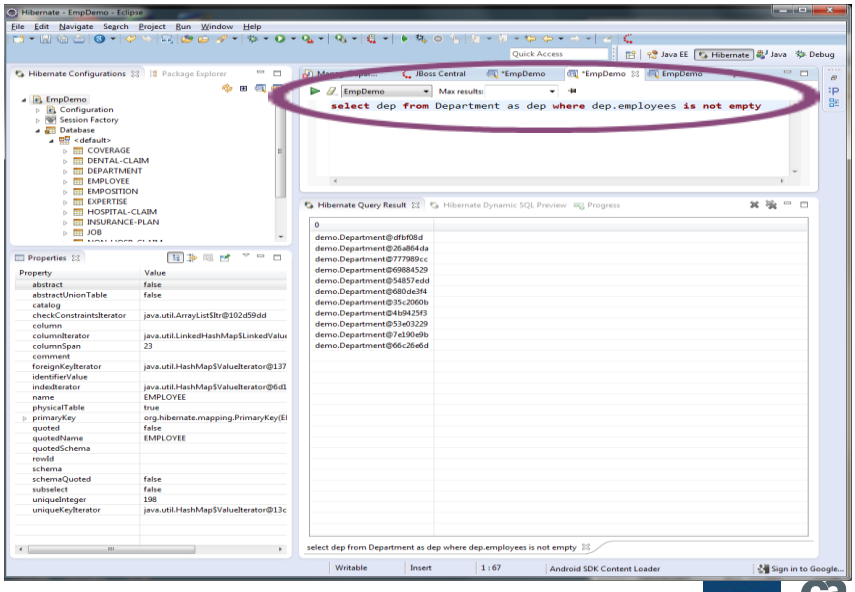
Hibernate HQL Editor – Sample EmpDemo session



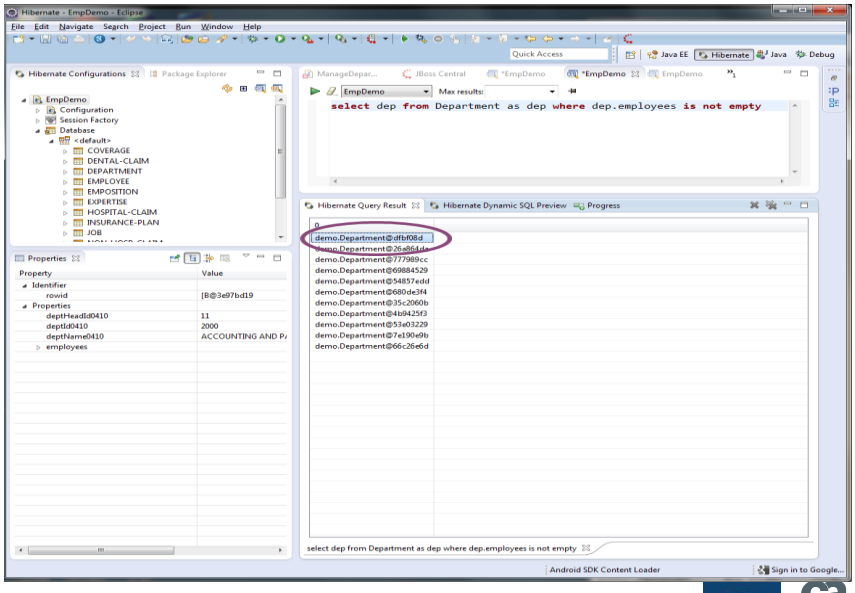
HQL Expand Database



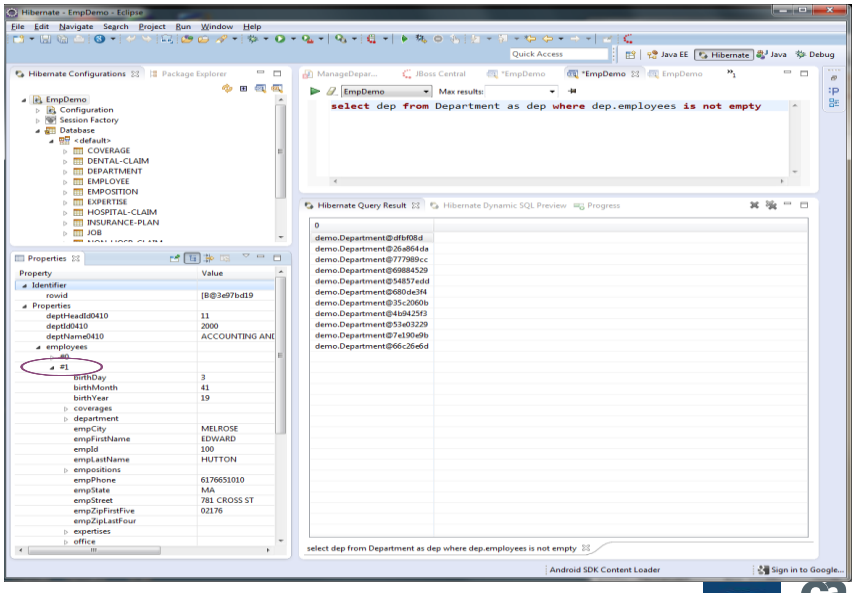
HQL Enter & Run HQL query



HQL Select Department

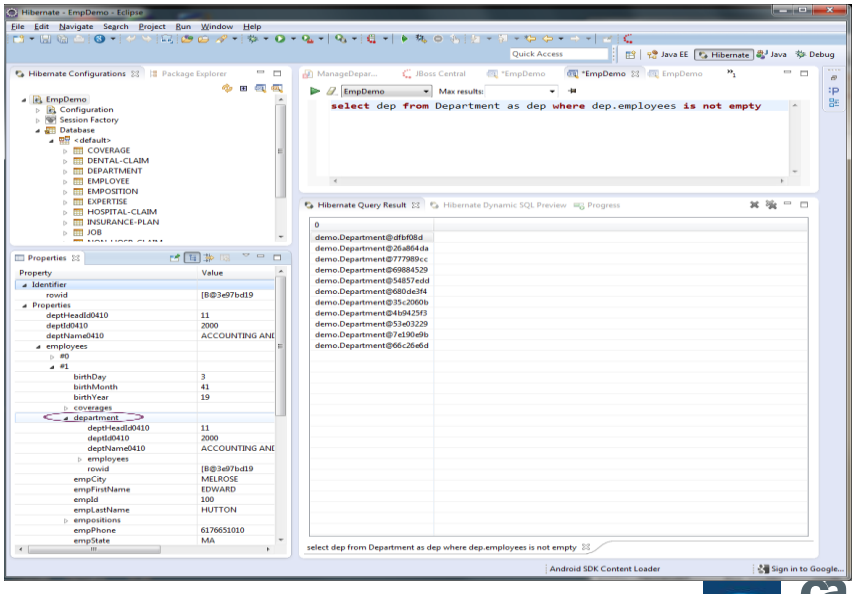


HQL Explore Department property



47 IUA/CA IDMS™ Technical Conference

HQL Explore Employee property



48 IUA/CA IDMS™ Technical Conference

Summary

- Object-Relational Mapping Concepts
- Java Persistence API
- Relational-Network Mapping
  - Use Virtual Foreign Key feature of CA IDMS 19.0
- Sample JPA access to Employee database
- Hibernate HQL Demo

FOR INFORMATION PURPOSES ONLY

Terms of this Presentation

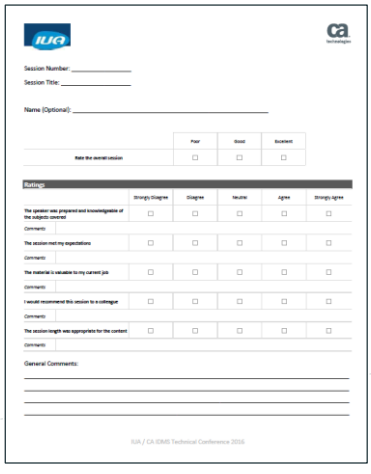
This presentation was based on current information and resource allocations as of May 2016 and is subject to change or withdrawal by CA at any time without notice. Notwithstanding anything in this presentation to the contrary, this presentation shall not serve to (i) affect the rights and/or obligations of CA or its licensees under any existing or future written license agreement or services agreement relating to any CA software product; or (ii) amend any product documentation or specifications for any CA software product. The development, release and timing of any features or functionality described in this presentation remain at CA's sole discretion. Notwithstanding anything in this presentation to the contrary, upon the general availability of any future CA product release referenced in this presentation, CA will make such release available (i) for sale to new licensees of such product; and (ii) to existing licensees of such product on a when and if-available basis as part of CA maintenance and support, and in the form of a regularly scheduled major product release. Such releases may be made available to current licensees of such product who are current subscribers to CA maintenance and support on a when and if-available basis. In the event of a conflict between the terms of this paragraph and any other information contained in this presentation, the terms of this paragraph shall govern.

Certain information in this presentation may outline CA's general product direction. All information in this presentation is for your informational purposes only and may not be incorporated into any contract. CA assumes no responsibility for the accuracy or completeness of the information. To the extent permitted by applicable law, CA provides this presentation "as is" without warranty of any kind, including without limitation, any implied warranties or merchantability, fitness for a particular purpose, or non-infringement. In no event will CA be liable for any loss or damage, direct or indirect, from the use of this document, including, without limitation, lost profits, lost investment, business interruption, goodwill, or lost data, even if CA is expressly advised in advance of the possibility of such damages. CA confidential and proprietary. No unauthorized copying or distribution permitted.

Questions and Answers

Please Complete a Session Evaluation Form

- The number for this session is A07
- After completing your session evaluation form, place it in the envelope at the front of the room



Session Number: _____
Session Title: _____
Name (Optional): _____

	Poor	Good	Excellent
Rate the overall session	☐	☐	☐

Ratings	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The speaker was prepared and knowledgeable of the subjects covered	☐	☐	☐	☐	☐
Comments:					
The session met my expectations	☐	☐	☐	☐	☐
Comments:					
The material is valuable to my current job	☐	☐	☐	☐	☐
Comments:					
I would recommend this session to a colleague	☐	☐	☐	☐	☐
Comments:					
The session length was appropriate for the content	☐	☐	☐	☐	☐
Comments:					

General Comments:

IUA / CA IDMS Technical Conference 2016