
ca world®'11
IT at the speed of business
modern application development with CA IDMS™
agility made possible™
ca technologies

Next-Generation Mainframe Management

MI340SN

modern application development with CA IDMS™

Dave Ross
CA Technologies

ca world®'11

agility made possible™

ca technologies

abstract

Dave Ross

CA Technologies, Manager, Software Engineering

Can you teach an old dog new tricks? Can you connect the next-generation technology with what's running today? This interactive session explores ways that you can leverage the value of your CA IDMS systems by integrating them with modern applications. Using the example of the Java Persistence Architecture and Hibernate as a starting point, this session discusses approaches that customers have used to access CA IDMS data and applications from web-based applications.

agenda

- Object-Relational Mapping Concepts
- Java Persistence API
- Relational-Network Mapping
- Sample JPA access to Employee database
- Enhancements discussion

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 need to 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

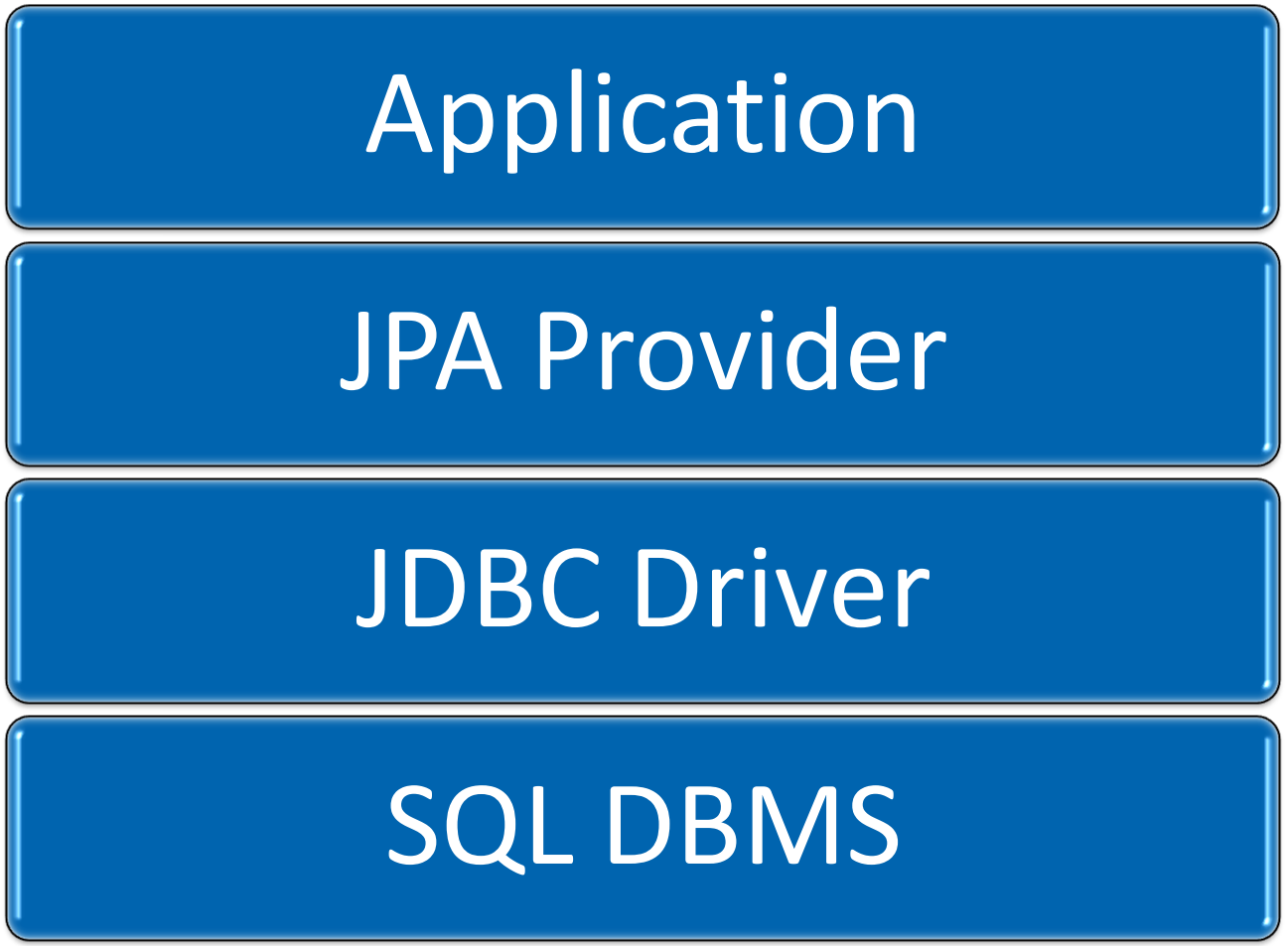
reverse engineering

- 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
- EJB 3.0
- Service Provider Interface (SPI)
- Providers
  - Hibernate (also has own API)
  - OpenJPA
  - Others

JPA architecture



JPA entities

- Entity
  - Represents application object
  - May represent database table
  - POJO
- EntityManager
  - Manages state and life cycle of entity
  - Create
  - Remove
  - Find
  - 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
  - DDL not quite standard
  - Add referential constraints manually
- Reverse engineering relatively complete
- Most CA IDMS databases are not SQL defined

JPA and CA IDMS Network databases

- Includes most CA IDMS applications
- Access via SQL
- No schema generation
- Reverse engineering with customization
- “Impedance mismatch”
  - Elements
  - Sets
  - Foreign Keys

relational-network mapping

Relational (SQL)

- Table
- Row
- Column
- Referential constraint

Network (CA IDMS)

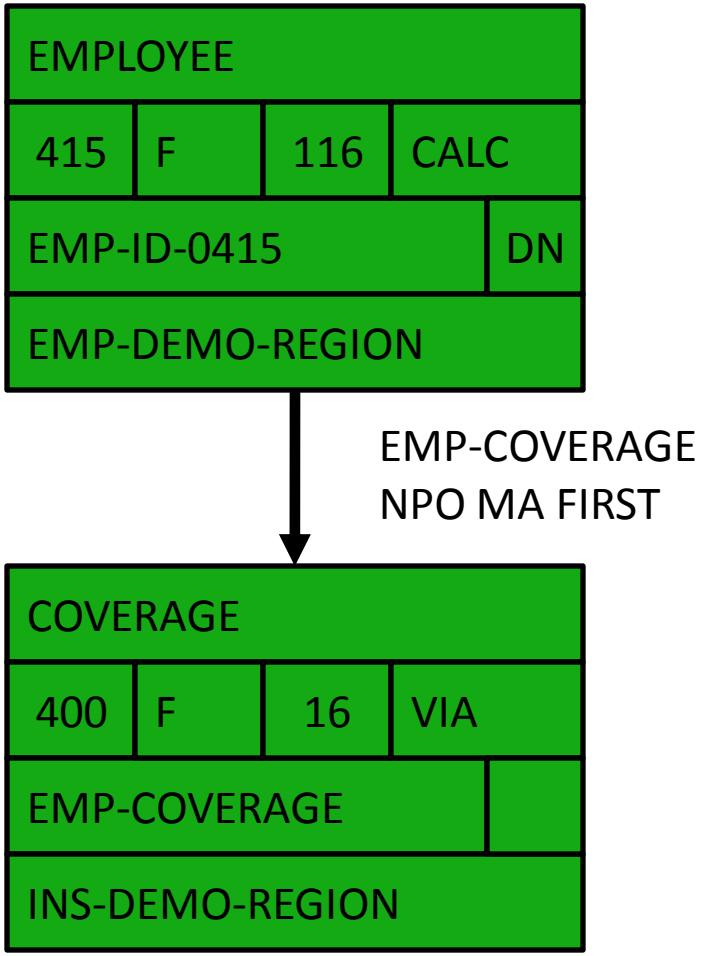
- Record definition
- Record occurrence
- Field
- Set

relational-network mapping
techniques to overcome impedance mismatch

- Syntax extensions
- Views
- Table procedures
- Embedded foreign keys

JPA example

- Employee database
  - EMPLOYEE
  - COVERAGE
- Entity classes
  - Employee
  - Coverage
- Syntax extensions
  - Set specification
  - ROWID pseudo-column



reverse engineered employee class

```
@Entity
@Table(name="EMPLOYEE", schema="EMPSCHM")
public class Employee implements Serializable {
   private short empId0415;
   private String empFirstName0415;
   // private member variables for each column...

   public Employee() {}

   @Id
   @Column(name="EMP_ID_0415",
      nullable=false, precision=4, scale=0)
   public short getEmpId0415() {
      return this.empId0415;
   }
   // access methods for each member variable...
}
```

reverse engineered coverage class

```
@Entity
@Table(name="COVERAGE",schema="EMPSCHM"
public class Coverage implements Serializable {
   private byte selectionYear0400;
   private byte selectionMonth0400;
   // private member variables for each column...

   public Coverage() {}

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

model set relationship

- Owner and member object references
- Manufacture a “primary key” in member
- Overcome lack of foreign key in member
  - Set specification

```
SELECT E.*, C.*
    FROM EMPLOYEE E, COVERAGE C
    WHERE "EMP-COVERAGE"
```

modified employee class
add reference to member objects

```
@Entity
@Table(name="EMPLOYEE",schema="EMPSCHM"
public class Employee implements Serializable {
    …
    private List<Coverage> coverage;
    …
    @Transient
    public List<Coverage> getCoverage() {
       return this.coverage;
    }
    …
}
```

modified coverage class
add reference to owner object

```
@Entity
@Table(name="COVERAGE",schema="EMPSCHM"
public class Coverage implements Serializable {
    …
    private Employee employee;
    …
    @Transient
    public Employee getEmployee() {
       return this.employee;
    }
    …
}
```

modified coverage class
add ROWID as primary key

```
@Entity
@Table(name="COVERAGE",schema="EMPSCHM"
public class Coverage implements Serializable {
    …
    private byte[] rowId;
    …
    @Id
    @Column(name="ROWID",
       nullable=false, precision=4, scale=0)
    public byte[] getRowId() {
       return this.rowId;
    }
    …
}
```

modified employee class
use set specification instead of foreign key

```
@Entity

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

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

@Table(name="EMPLOYEE",schema="EMPSCHM"
public class Employee implements Serializable {
```

populate set occurrence objects

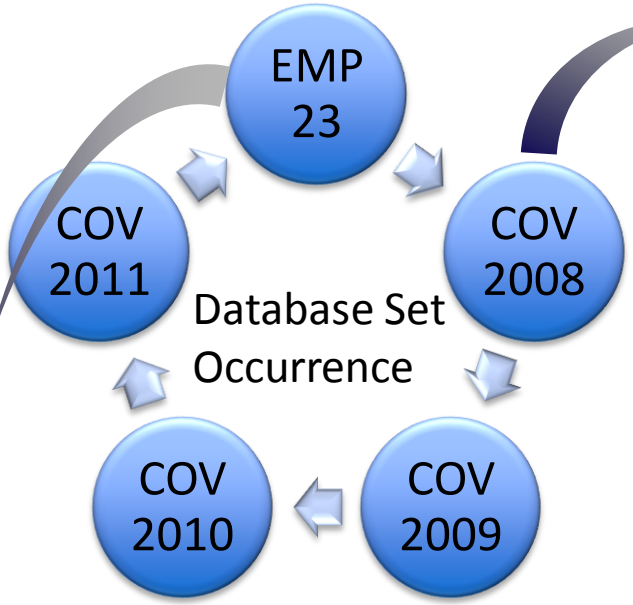
```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("NonSqlJPA");
EntityManager em = emf.createEntityManager();

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	2008	F
08010204	2009	F
08010210	2010	M
08010303	2011	F

Java Objects

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

update member

```
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 = ?
```

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 EMP_ID_0415 = ?
```

insert member

- Usually need foreign keys
- Alternative is use of a procedure

relational to network mapping
tradeoffs

SQL Extensions	Views	Table Procedures	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

foreign keys

- Easier support for INSERT
- JPA recognizes relationship
  - Reverse engineering of object definitions
  - Generated SQL
- Still need primary key in member

reverse engineered employee class
generated relationship for foreign key

```
@Entity
@Table(name="EMPLOYEE",schema="EMPSCHM"
public class Employee implements Serializable {
    …
    private List<Coverage> coverage;
    …
    @OneToMany(mappedBy="employee",
        targetEntity=Coverage.class,
        fetch=FetchType.EAGER,
        cascade=CascadeType.ALL)
    public List<Coverage> getCoverage() {
        return this.coverage;
    }
    …
}
```

reverse engineered coverage class
generated relationship for foreign key

```
@Entity
@Table(name="COVERAGE",schema="EMPSCHM"
public class Coverage implements Serializable {
    …
    private Employee employee;
    …
    @ManyToOne(optional=false)
    @JoinColumn(name=EMP_ID_0400, udpatable=false)
    public Employee getEmployee() {
       return this.employee;
    }
    …
}
```

tips for using JPA

- Exceptions can be vague
- Use Type 2 driver for debugging
  - Can use Type 4 for production
- Enable SQL trace
  - Use ODBC Administrator
- Displays generated SQL statements

discussion of potential enhancements

- SQL DDL enhancements
  - Add referential constraint with ALTER TABLE?
- Enhanced foreign key support for set members
  - Foreign implied by owner primary key?
  - Restructure option to add foreign key?
  - Virtual option for foreign key?
  - Useful if limited set options supported?
- Enhanced primary key support for set members
  - More persistent ROWID?
  - Virtual primary key for via member?

summary

- Object-Relational Mapping Concepts
- Java Persistence API
- Relational-Network Mapping
- Sample JPA access to Employee database
- Enhancements discussion

Q&A

Exhibition Center: related technologies

- **Booth 516** – CA IDMS

Session # MI340SN

Please scan this image to fill in your session survey on a mobile device or complete a hard copy session evaluation form

ca world®'11

Mainframe networking lunch

Engage in CA solution discussion with your peers and CA experts

Where: Exhibition Center

When: Tuesday and Wednesday

Time: 12:00pm – 1:15pm

ca world®'11

terms of this presentation
for information purposes only

This presentation was based on current information and resource allocations as of November 2011 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.

thank you

ca world®'11