

Efficient Object-Relational Mapping for JAVA and J2EE Applications – or the impact of J2EE on RDB

Marc Stampfli

Oracle Software (Switzerland) Ltd.

Underestimation

According to customers about **20-50% percent** of the time of developer is used **for manual object-relational mapping**.

Managing persistence related issues is the most **underestimated** challenge in enterprise Java today – in terms of complexity, effort and maintenance

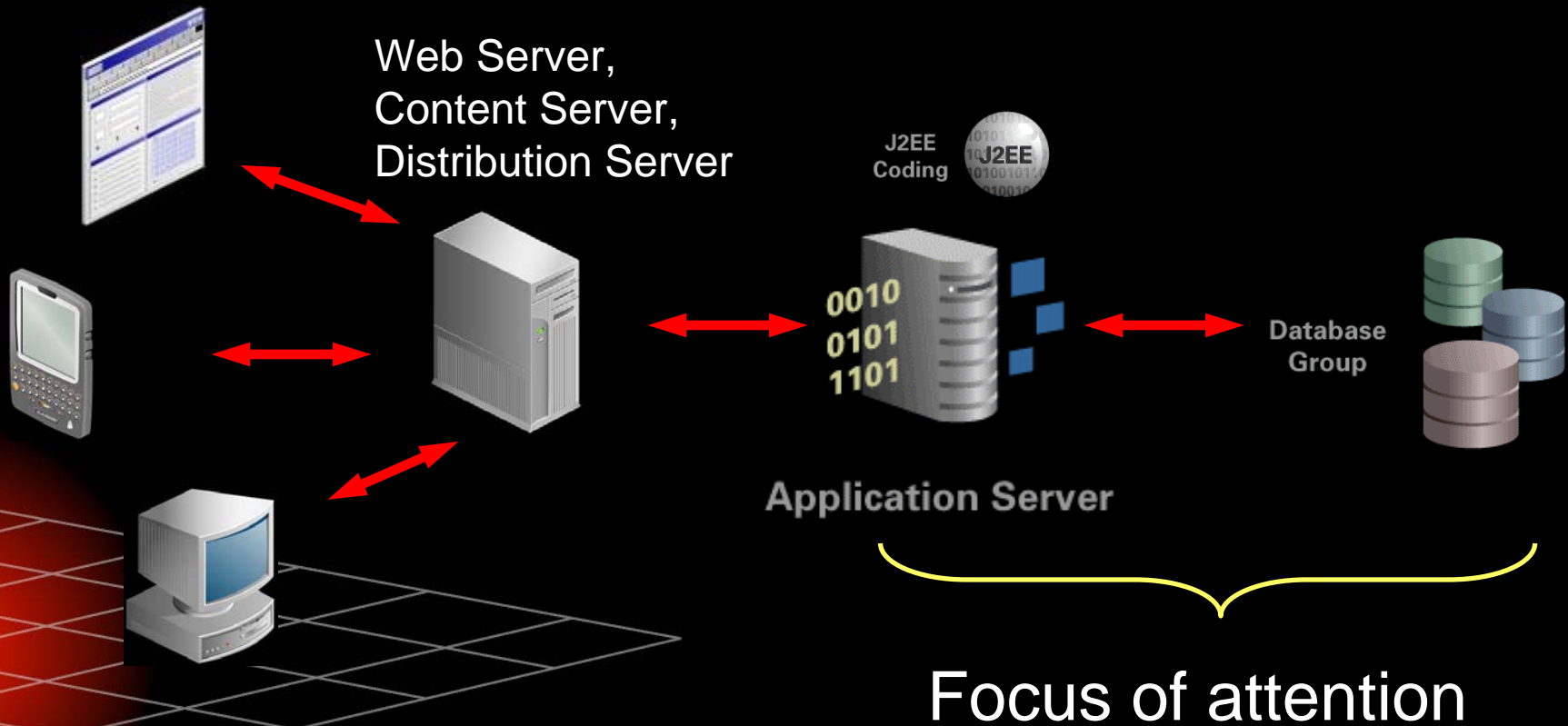
Doesn't JDBC handle these issues?

Agenda

- Impedance Mismatch
- Object Persistence Options
- J2EE Persistence Requirements
- J2EE Persistence Framework

Enterprise App. Architecture

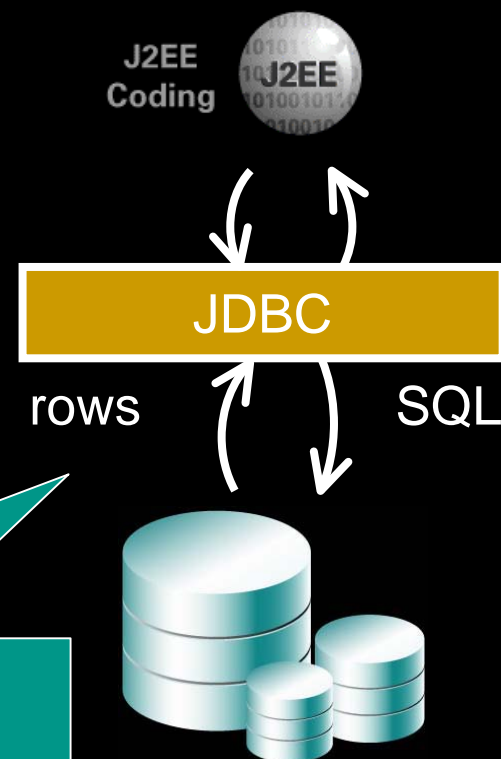
10^g



JDBC

- Java standard for accessing databases
- JDBC is simply the database connection utilities Java developers need to build upon

```
• Connection con =  
  DriverManager.getConnection(...);  
• Statement stmt = con.createStatement();  
• stmt.execute("create table JUGSData (" +  
  "programmer varchar (32),"+ "day char  
  (3),"+ "cups integer);")
```

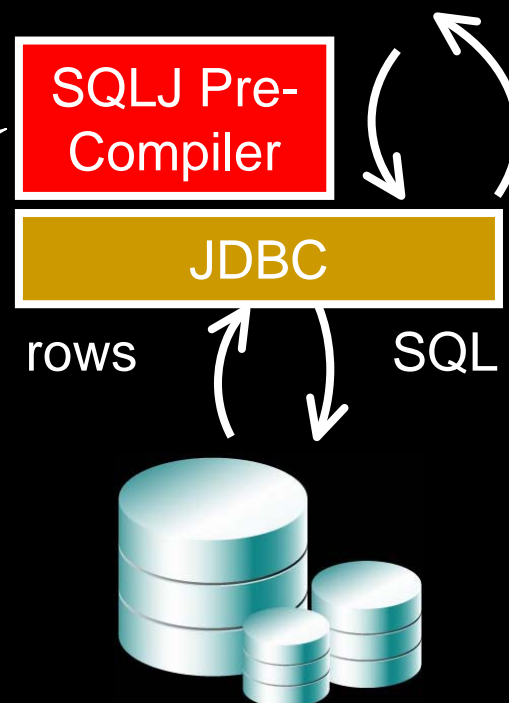


SQLJ

- Meta-Standard for accessing databases
- Pre-compiler to build JDBC Java Code

```
1. #sql iterator Iter (double sal, String
   ename);
2. String ename = 'Smith';
3. Iter it; ...
4. #sql it = { select ENAME, SAL from EMP
   where ENAME = :ename };
```

J2EE
Coding



Impedance Mismatch


- The differences in relational and object technology is known as the “**object-relational impedance mismatch**”
- Challenging problem to address because it requires a combination of relational database and object expertise

Impedance Mismatch

Factor	J2EE	Relational Databases
Logical Data Representation	Objects, methods, inheritance	Tables, SQL, stored procedures
Scale	Hundreds of megabytes	Gigabytes, terabytes
Relationships	Memory references	Foreign keys
Uniqueness	Internal object id	Primary keys
Key Skills	Java development, object modeling	SQL, Stored Procedures, data management
Tools	IDE, Source code management, Object Modeler	Schema designer, query manager, performance profilers, database configuration

Object Level Options

- Depends on what component architecture is used:
 - Entity Beans BMP – Bean Managed Persistence
 - Entity Beans CMP – Container Managed Persistence
 - Access Java Objects via Persistence Layer (POJO or J2EE)
 - Can be off the shelf or “home-grown”



Do you build
your O-R
Mapping Tool
yourself?

Entity Beans - BMP

- In BMP, developers write the persistence code themselves
- Database reads and writes occur in specific methods defined for bean instances
- The container calls these methods - usually on method or transaction boundaries

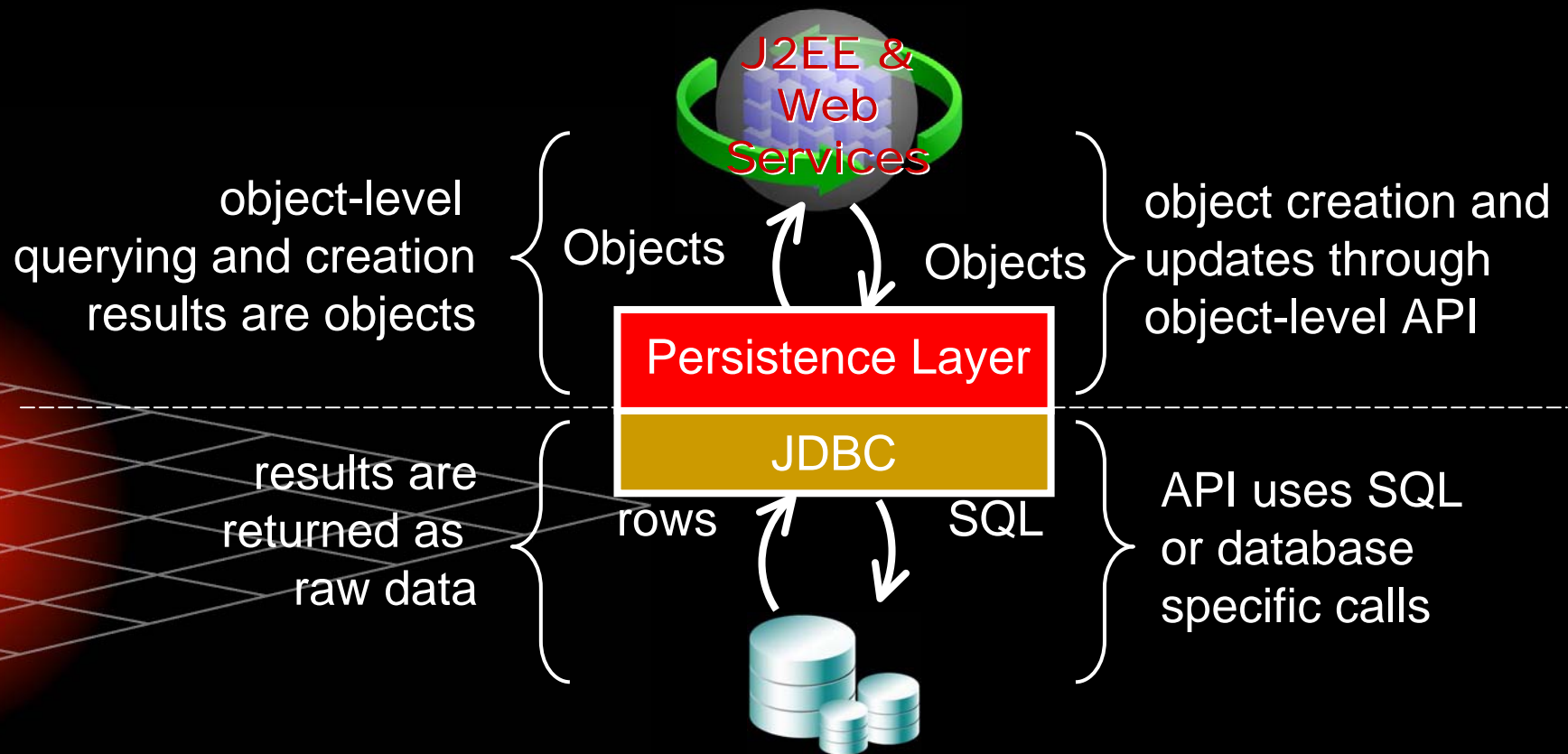
```
ejbLoad() - "load yourself"  
ejbStore() - "store yourself"  
ejbCreate() - "create yourself"  
  findBy...() - "find yourself"  
ejbRemove() - "remove yourself"
```

Entity Beans - CMP

- Persistence is based on information in the deployment descriptors
 - More “automatic” persistence – managed by the Application Server, can be faster than BMP
 - No special persistence code in the bean
 - Description of the persistence done with tools and XML files
- Less control, persistence capabilities are limited to the functionality provided.
 - Very difficult to customize or extend CMP features as it is built-in
 - Do have options to plug-in a 3rd party CMP solution on an app server

Object Persistence Layer

- Abstracts persistence details from the application layer, supports Java objects/Entity Beans



Basic J2EE Persistence Checklist ^{10^g}

- Mappings
- Object traversal
- Queries
- Transactions
- Optimized database interaction
- Database Triggers and Cascade Deletes
- Caching
- Locking
- Database features

Mapping

- Object model and Schema must be mapped
 - True for any persistence approach
- Most contentious issue facing designers
 - Which classes map to which table(s)?
 - How are relationships mapped?
 - What data transformations are required?

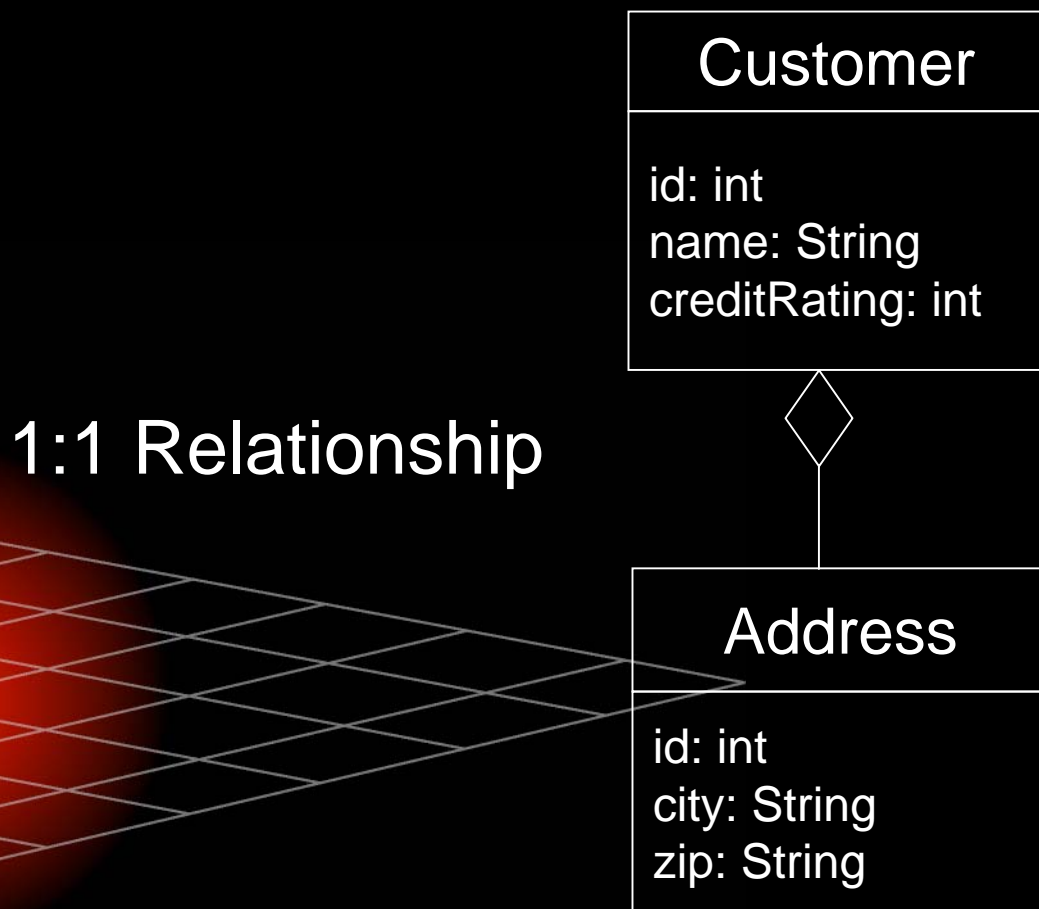
Good and Poor Mapping Support

- Good mapping support:
 - Domain classes don't have to be "tables"
 - References should be to objects, not foreign keys
 - Database changes (schema and version) easily handled
- Poor mapping support:
 - Classes must exactly mirror tables
 - Middle tier needs to explicitly manage foreign keys
 - Classes are disjoint
 - Change in schema requires extensive application changes

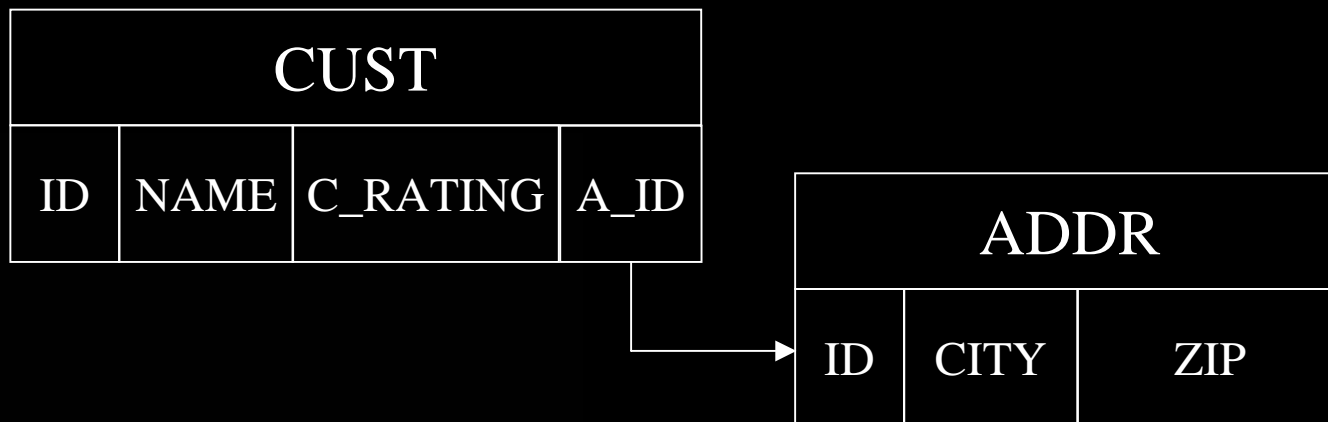
Data and Object Models

- Rich, flexible mapping capabilities provide data and object models a degree of independence
- Otherwise, business object model will force changes to the data schema or vice-versa
- Often, J2EE component models are nothing more than mirror images of data model – NOT desirable

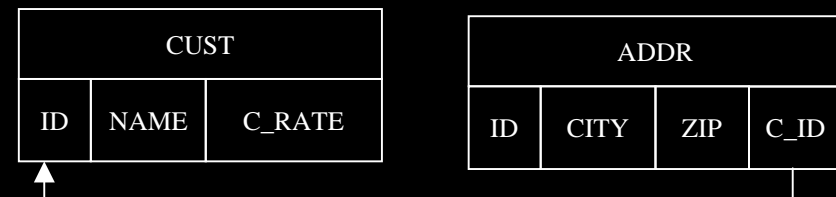
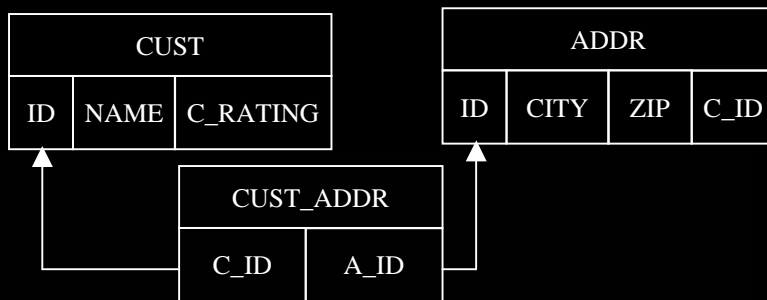
Simple Object Model



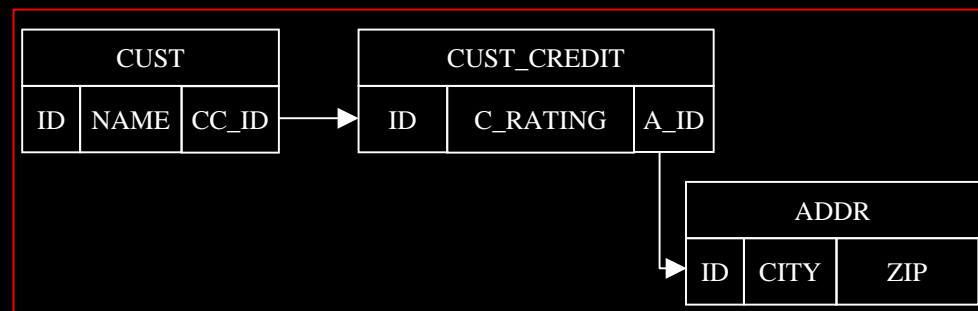
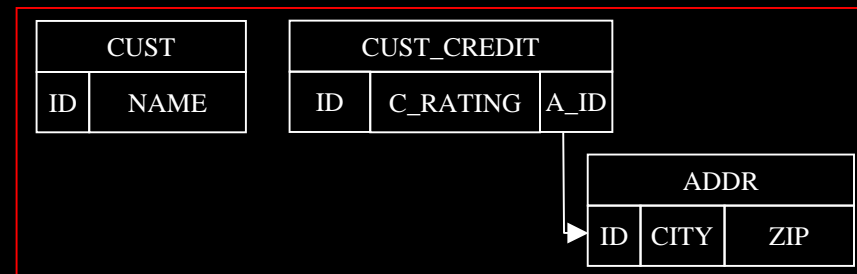
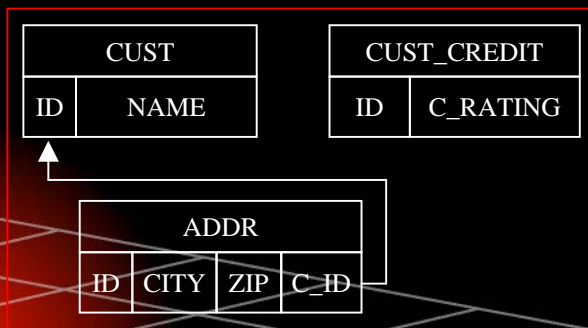
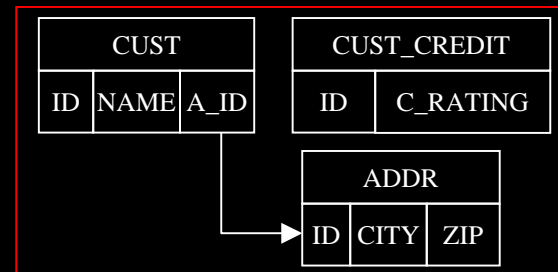
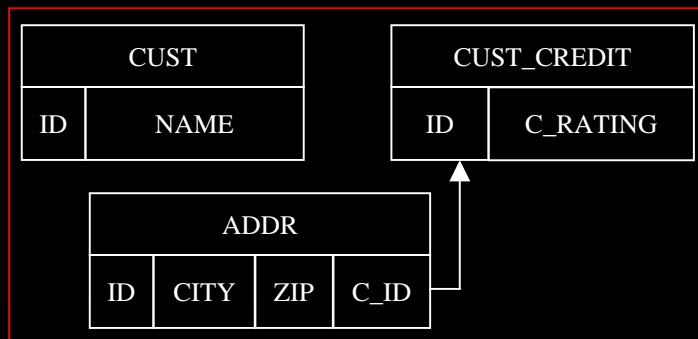
Typical 1-1 Relationship Schema



Other possible Schemas...



Even More Schemas...



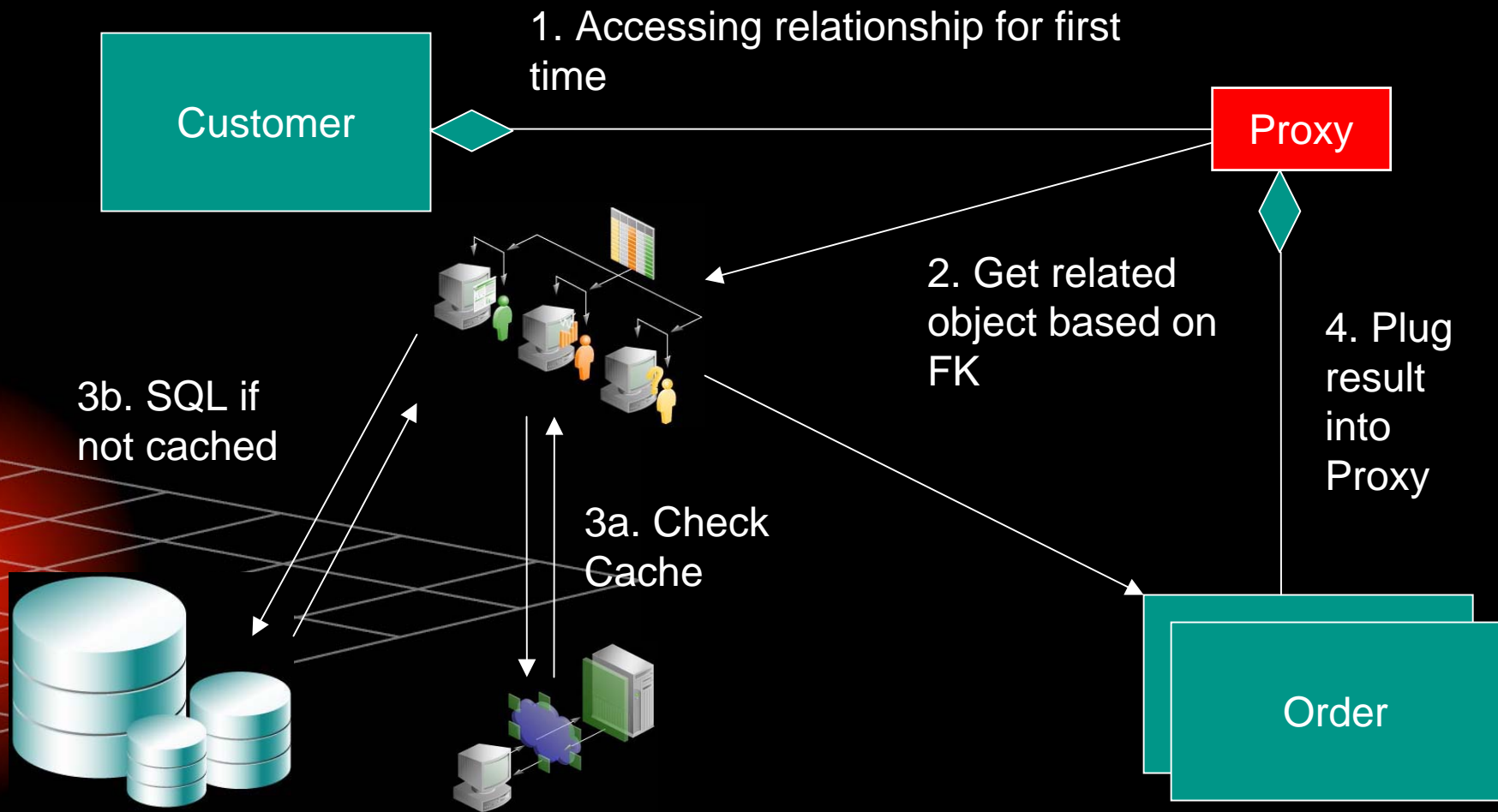
Mapping Summary

- Just showed **nine** valid ways a 1-1 relationship could be represented in a database
 - Most persistence layers and application servers will only support *one*
- Without good support, designs will be forced
- Imagine the flexibility needed for other mappings like 1-M and M-M

Object Traversal – Lazy Reads ^{10^g}

- J2EE applications work on the scale of a few hundreds of megabytes
- Relational databases routinely manage gigabytes and terabytes of data
- Persistence layer must be able to transparently fetch data “just in time”

Just in Time Reading – Faulting Process



Object Traversals

- Even with lazy reads, object traversal is not always ideal
 - To find a phone number for the manufacturer of a product that a particular customer bought, may do several queries:
 - Get customer in question
 - Get orders for customer
 - Get parts for order
 - Get manufacturer for part
 - Get address for manufacturer
 - Very natural object traversal results in 5 queries to get data that can be done in 1

N+1 Reads Problem

- Many persistence layers and application servers have an N+1 reads problem
- Causes N subsequent queries to fetch related data when a collection is queried for
- A side effect of the impedance mismatch and poor mapping and querying support in persistence layers

N+1 Reads

- Must have solution to minimize queries
- Need flexibility to reduce to 1 query, 1+1 query or N+1 query where appropriate
 - 1 Query when displaying list of customers and addresses – known as a “Join Read”
 - 1+1 Query when displaying list of customers and user may click button to see addresses – known as a “Batch Read”
 - N+1 Query when displaying list of customers but only want to see address for selected customer

Queries

- Java developers are not usually SQL experts
 - Maintenance and portability become a concern when schema details hard-coded in application
- Allow Java based queries that are translated to SQL and leverage database options
 - EJB QL, object-based proprietary queries, query by example

Queries

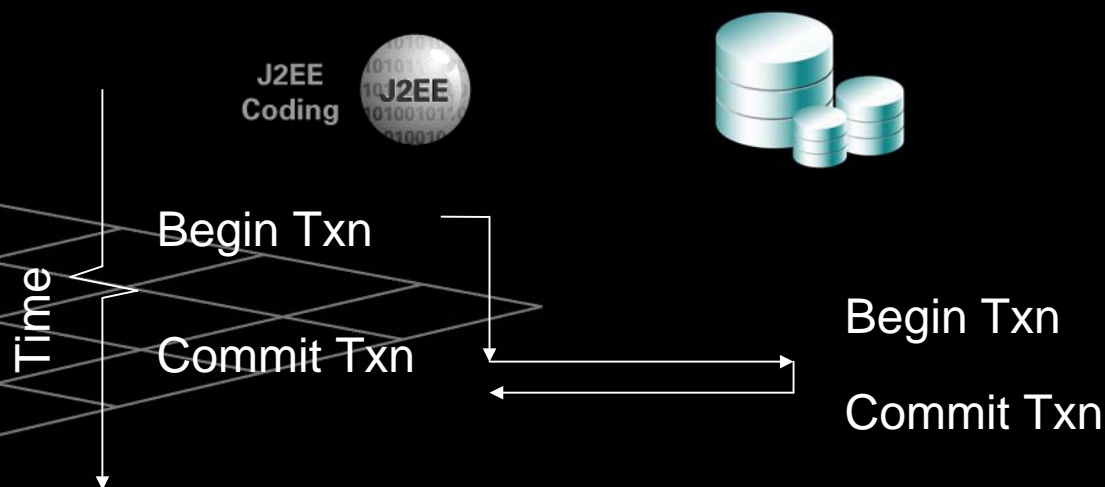
- Persistence layer handles object queries and converts to SQL
- SQL issued should be as efficient as written by hand
- Should utilize other features to optimize
 - Parameter binding, cached statements
- Some benefits to dynamically generated SQL :
 - Ability to create minimal update statements
 - Only save objects and fields that are changed
 - Simple query-by-example capabilities

Query Requirements

- Must be able to trace and tune SQL
- Must be able use ad hoc SQL where necessary
- Must be able to leverage database abilities
 - Outer joins
 - Nested queries
 - Stored Procedures
 - Oracle Hints

Transaction Management

- J2EE apps typically support many clients sharing small number of db connections
- Ideally would like to minimize length of transaction on database

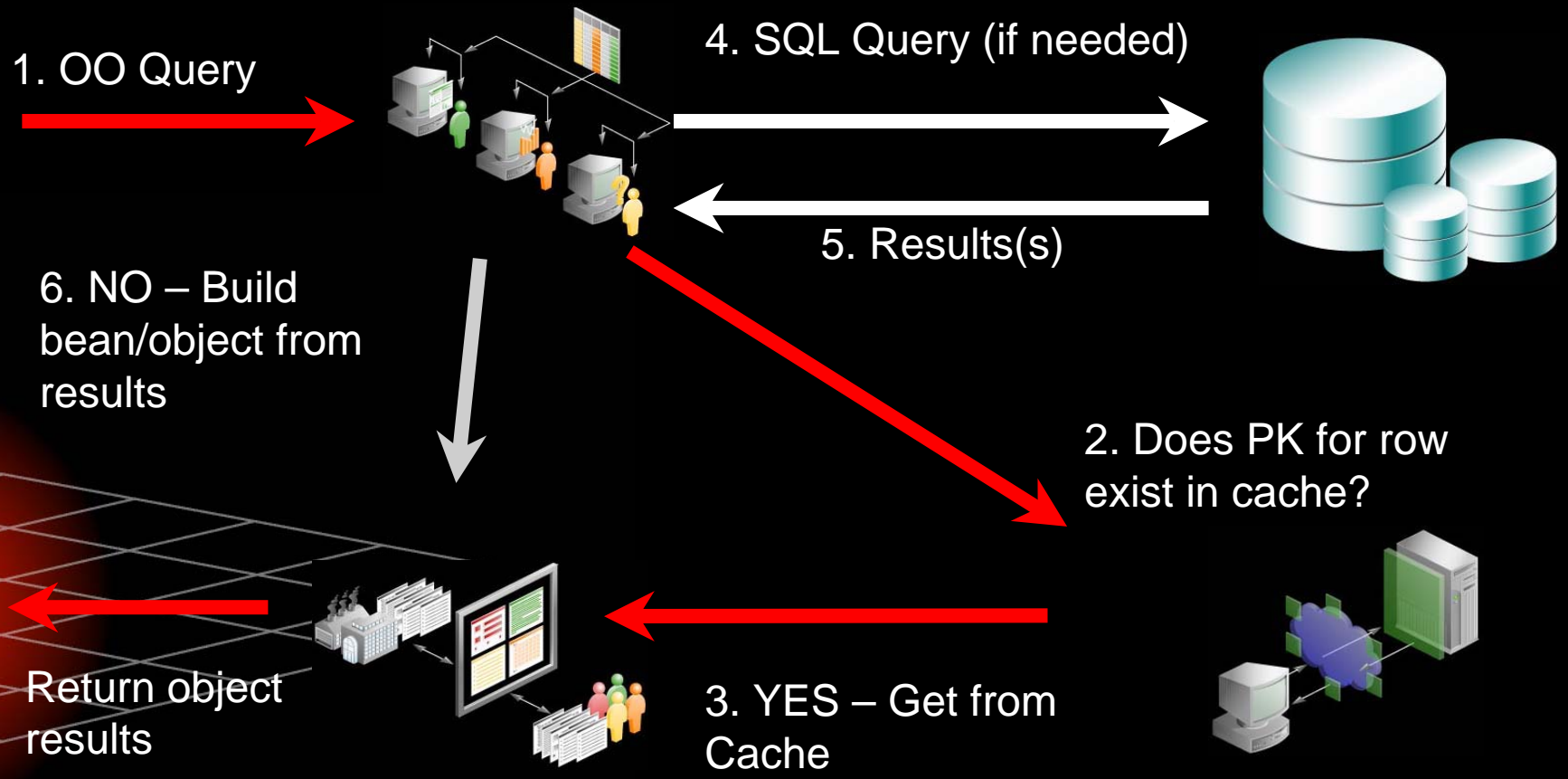


Caching

- Any application that caches data, now has to deal with stale data
- When and how to refresh?
- Will constant refreshing overload the database?
- Problem is compounded in a clustered environment
- App server may want be notified of database changes

10^g

Caching



Locking

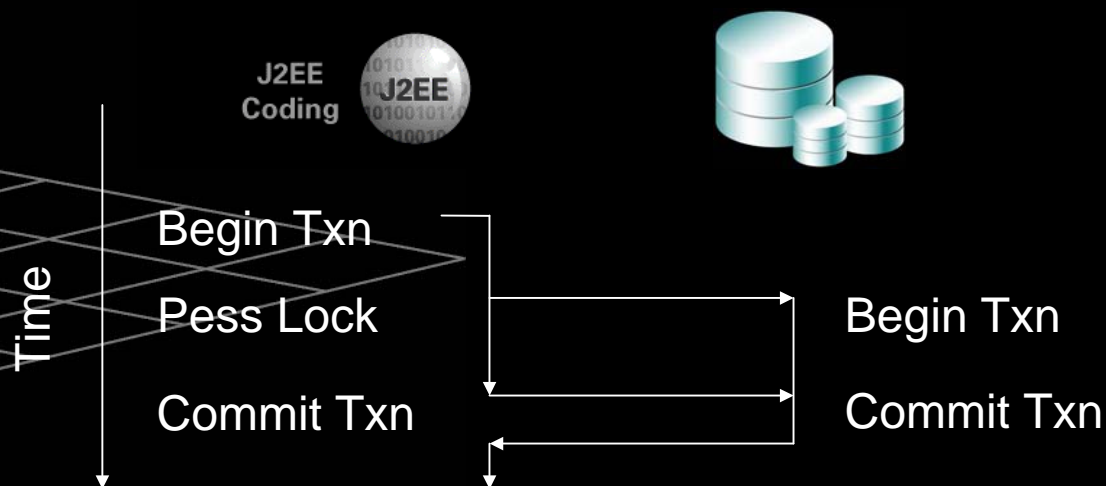
- J2EE developers want to think of locking at the object level
- Databases may need to manage locking across many applications
- Persistence layer or application server must be able to respect and participate in locks at database level

Optimistic Locking

- DBA may wish to use version, timestamp and/or last update field to represent optimistic lock
 - Java developer may not want this in their business model
 - Persistence layer must be able to abstract this
- Must be able to support using any fields including business domain

Pessimistic Locking

- Requires careful attention as a JDBC connection is required for duration of pessimistic lock
- Should support `SELECT FOR UPDATE [NOWAIT]` semantics



Conclusion

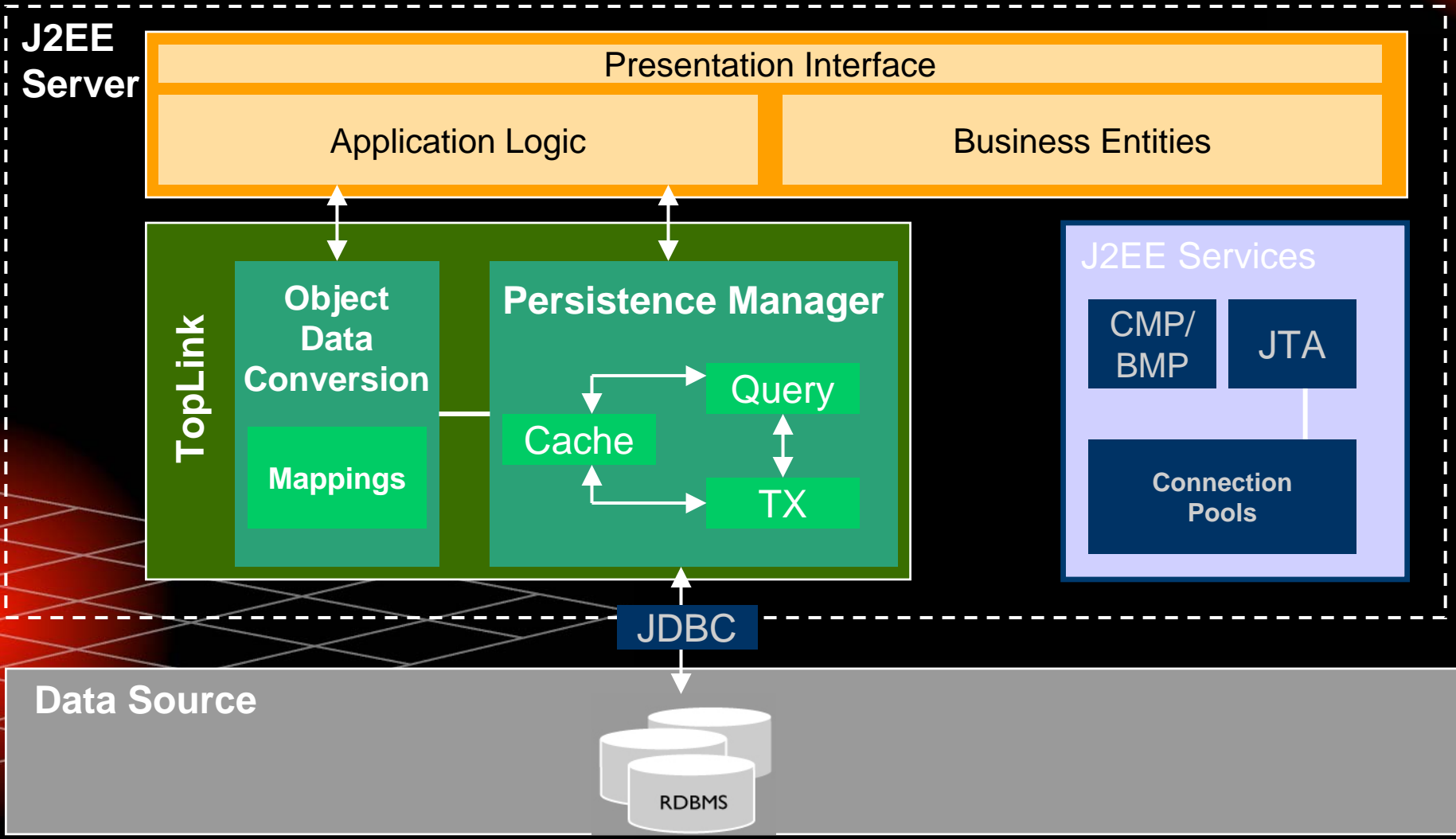
- J2EE apps accessing relational databases:
 - Don't need to compromise object/data model
 - Need to fully understand what is happening at database level
 - Can utilize database features
 - Do not have to hard code SQL to achieve optimal database interaction
 - Can find solutions that effectively address persistence challenges and let them focus on J2EE application

TopLink Key Technical Features

- TopLink Persistency Layer Framework from Oracle Application Server 10g solves these issues by:
 - Meta-Data Architecture
 - Comprehensive Visual Mapping Workbench
 - Advanced Mapping Support and Flexibility
 - Query Flexibility
 - Just In Time reading
 - Caching
 - Transaction support and integration
 - Locking
 - Performance tuning options
 - SDK

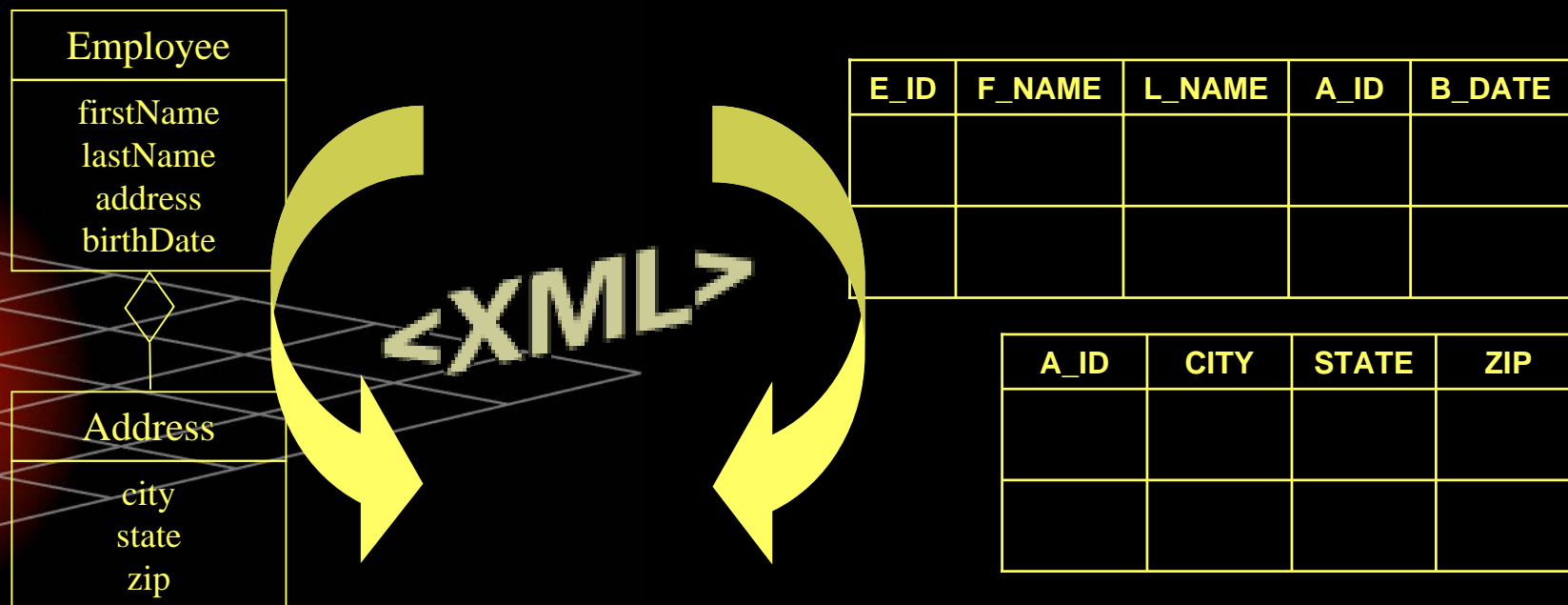


TopLink Runtime Architecture



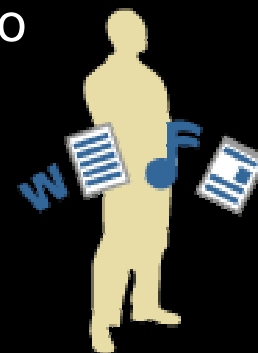
Meta-Data Architecture for Object Relational Mapping

- Mapping information is kept in XML descriptors and not in the objects
- Meta-data means OracleAS TopLink is NOT at all intrusive on either the object model or the schema



Advanced Mapping Support and Flexibility

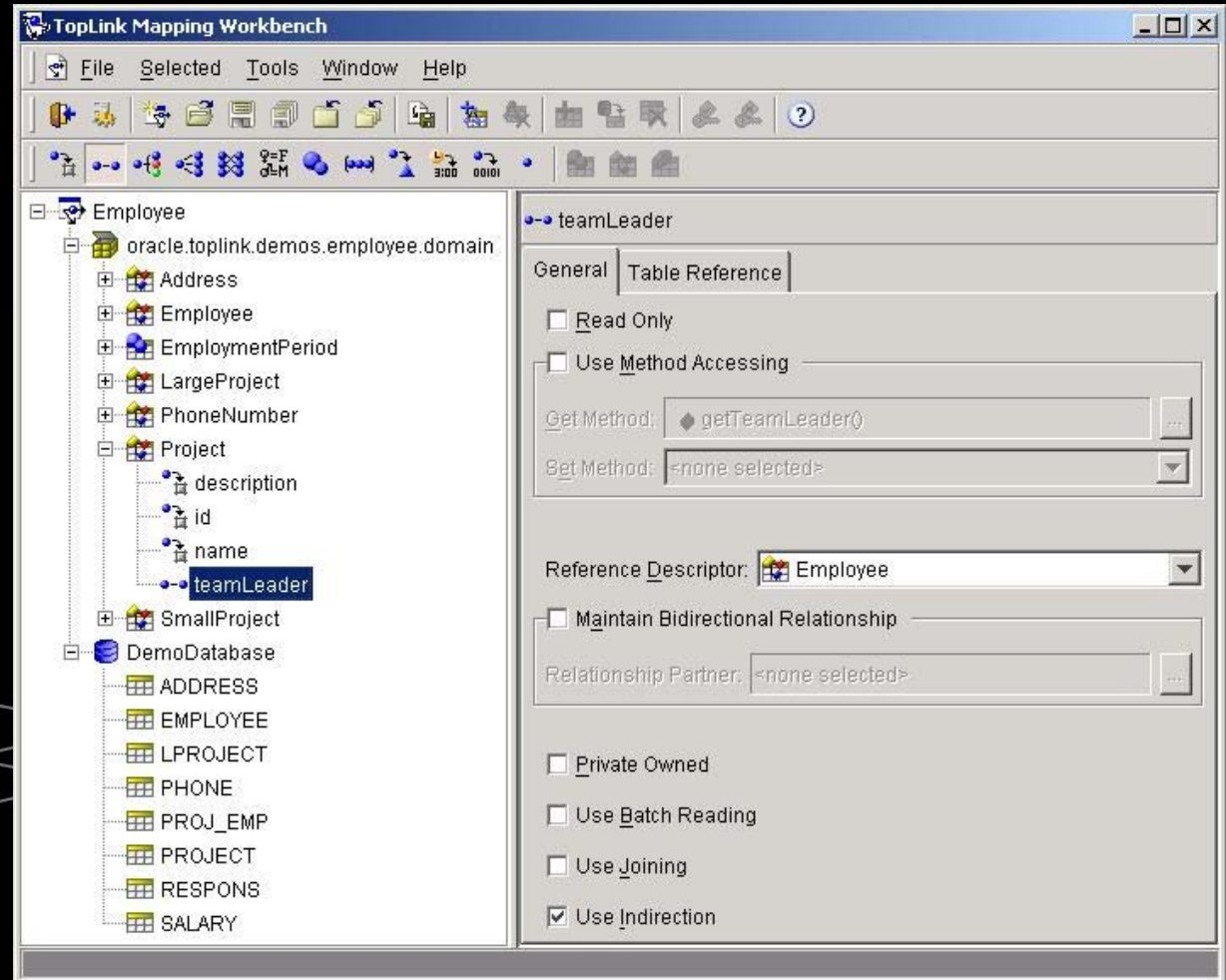
- Direct to Field, One to One, One to Many, Many to Many
 - Any kind of foreign key relationships in Database supported – including intermediate tables
- Object Type, Transformation
 - Enumeration ('Male'-> 'M') or conversions (String to Number)
 - User defined transformations
- Aggregates, Multiple tables
 - Multiple objects/beans per row
 - Map an object/bean to multiple tables
- And many more – Serialized mappings, Direct Collections, Object-Relational Mappings, etc



Mapping Workbench

10^g

- Lots of mapping tools out there, however don't get fleeced by a slick GUI
- The underlying mapping support is what's important



Summary

- Oracle Application Server 10g – TopLink Persistence Layer solves all the mentioned problems
 - Mapping
 - Queries
 - Transactions
 - Deferred Read Management
 - Locking
 - Caching
- TopLink is independent of Database and Application Server Technology



ORACLE®

**For further Information contact:
Marc Ph. Stampfli**

E-Mail: marc.stampfli@oracle.com

Phone: +41 56 483 32 11