
ORACLE®

Understanding Impact of J2EE Applications On Relational Databases

Dennis Leung, VP Development
Oracle9iAS TopLink
Oracle Corporation

ORACLE

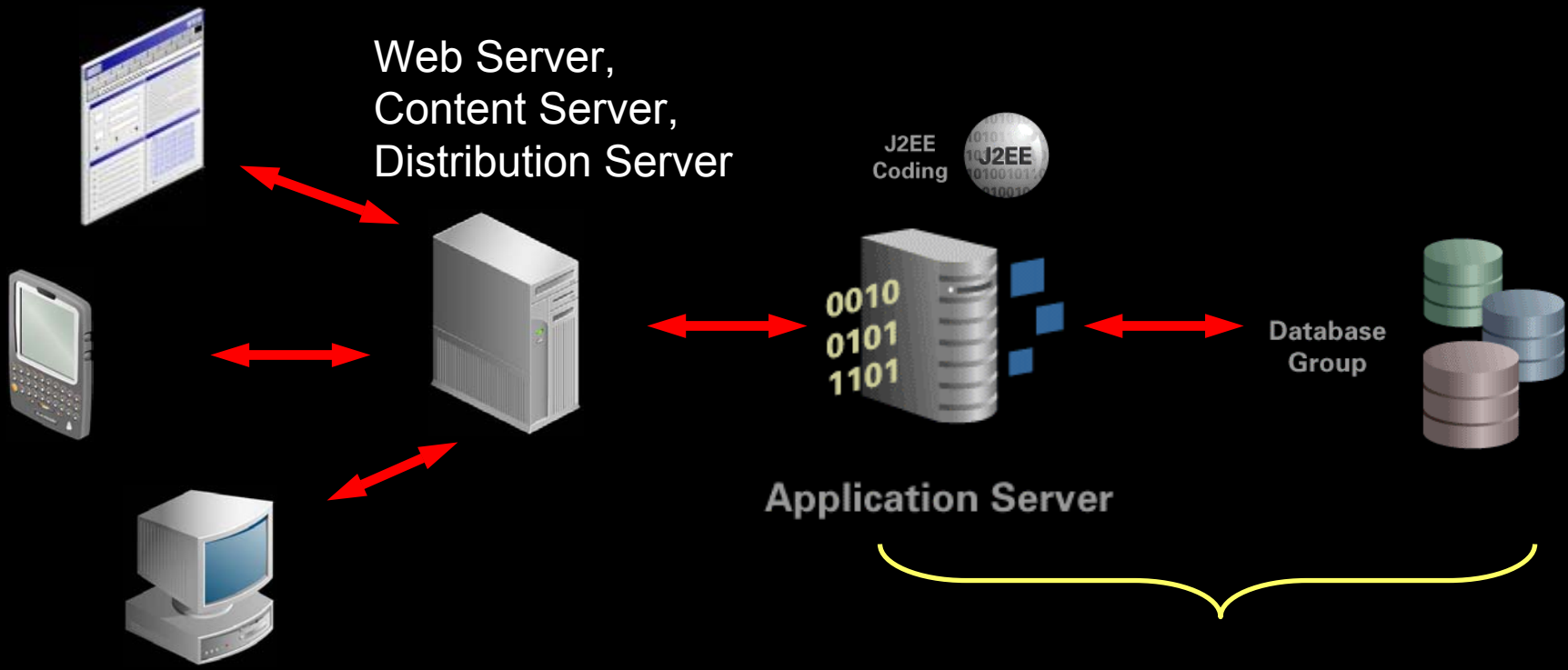
J2EE Apps and Relational Data

- J2EE is one of leading technologies used for building n-tier, web applications
  - J2EE based on object technology
- Relational databases are the most common data source accessed by J2EE apps
- They are diverse technologies that need to be used together
- This talk identifies a few of the issues to be considered.

Underestimation

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

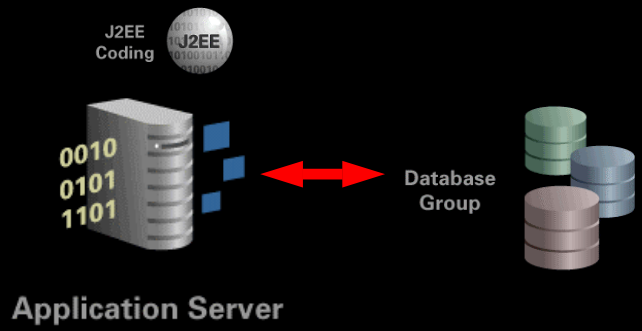
Enterprise App Architecture



Focus of attention

J2EE Architectures

- J2EE Architecture Options
  - Servlets
  - JSP
  - Session Beans
  - Message Driven Beans
  - Web Services
- Bottom Line – Java application needs to access relational data somehow…

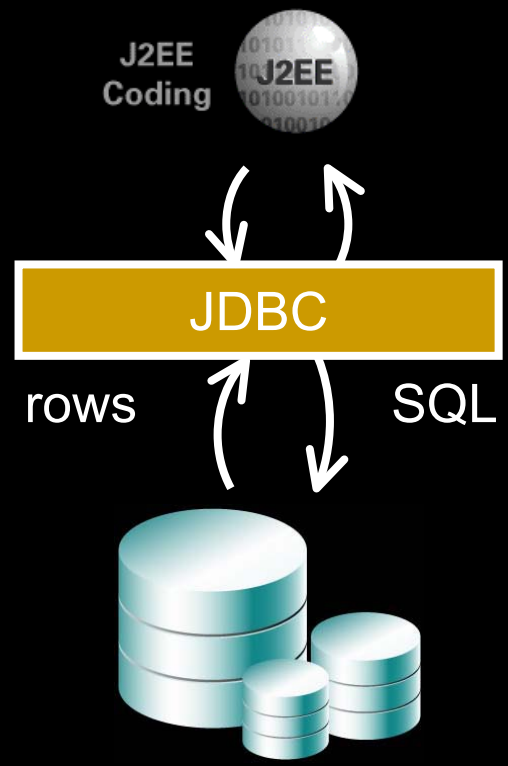


J2EE Access of Relational Data

- Direct JDBC – window on data
  - Direct SQL calls, uses rows and result sets directly
- Entity beans/Business Objects
  - Accessed as objects or components (EJBs), transparent that the data is stored in RDB
  - Need persistence layer in middle tier to handle the object-relational mapping and conversion
  - Focus of this talk

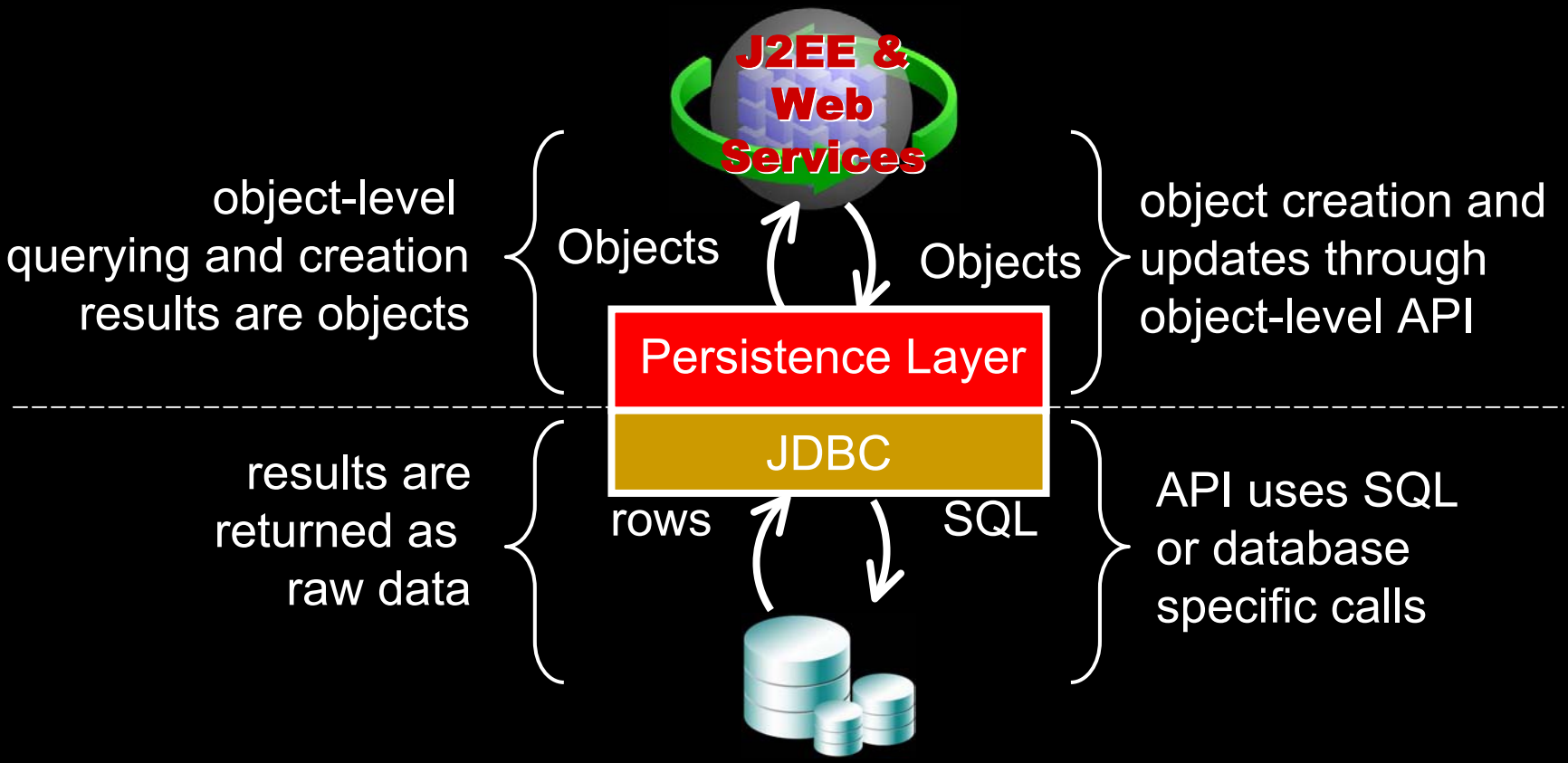
JDBC

- Java Standard for accessing databases
- Issue SQL, get back result sets/rows
- All J2EE apps will use JDBC to access RDBs



Object Persistence Layer

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



Entity Beans - CMP

- Persistence is based on information in the deployment descriptors
  - More “automatic” persistence – managed by the Application Server
  - 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

Impedance Mismatch

- The differences in relational and object technology is know 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

J2EE Developer Desires

- Data model should not constrain object model
- Don’t want database code in object/component code
- Accessing data should be fast
- Minimize calls to the database – they are expensive
- Object-base queries – not SQL
- Isolate J2EE app from schema changes
- Would like to be notified of changes to data occurring at database

DBA Desires

- Adhere to rules of database (referential integrity, stored procedures, sequence numbers etc.)
- Build the J2EE application but do NOT expect to change schema
- Build the J2EE application but the schema might change
- Let DBA influence/change database calls/SQL generated to optimize
- Be able to log all SQL calls to database
- Leverage database features where appropriate (outer joins, sub queries, specialized database functions)

Differences

- Desires are contradictory
  - “Insulate application from details of database but let me leverage the full power of it”
  - Different skill sets
  - Different methodologies
  - Different tools
- Technical differences must also be considered!

Basic J2EE Persistence Checklist

- Design Time
  - Mappings
  - GUI, tools, database types, Java types
- Run Time
  - Queries
  - Object Traversal
  - Transactions
  - Optimized database interaction
  - Locking
  - Caching
  - Database features

How Are Databases Affected?

- In reality, it's almost the other way around, J2EE app is influenced by database, since RDBs are usually the incumbent technology
  - Database "rules" need to be followed
  - Object model may be influenced by data model
  - Database interaction must be optimized for performance
  - "Source of truth" for data integrity is database, not app server
  - Existing business logic in database

Mapping

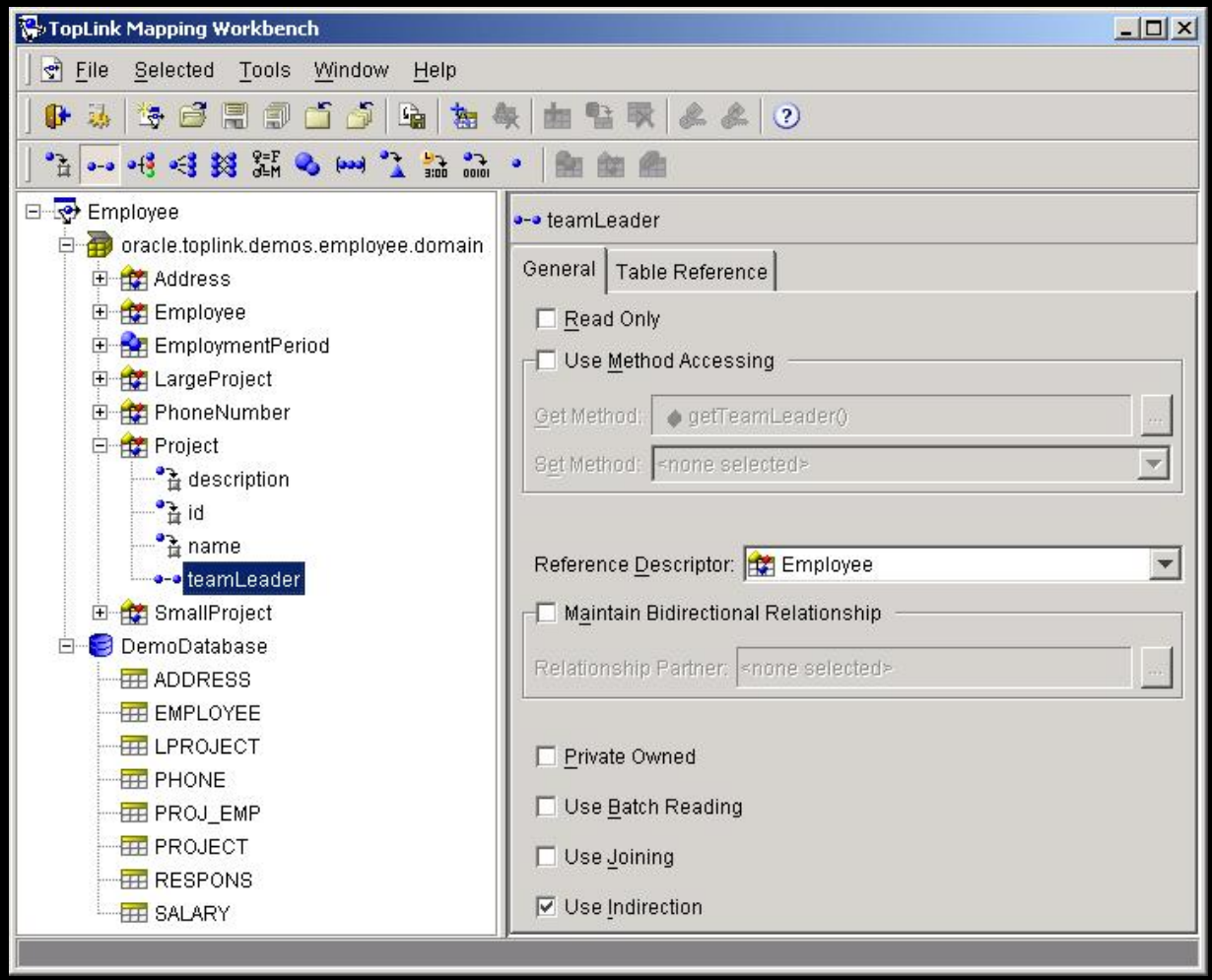
- Object model and Schema must be mapped
- Most contentious issue facing designers
  - Which classes map to which table(s)?
  - How are relationships mapped?
  - What data transformations are required?

Good and Poor Mapping Support

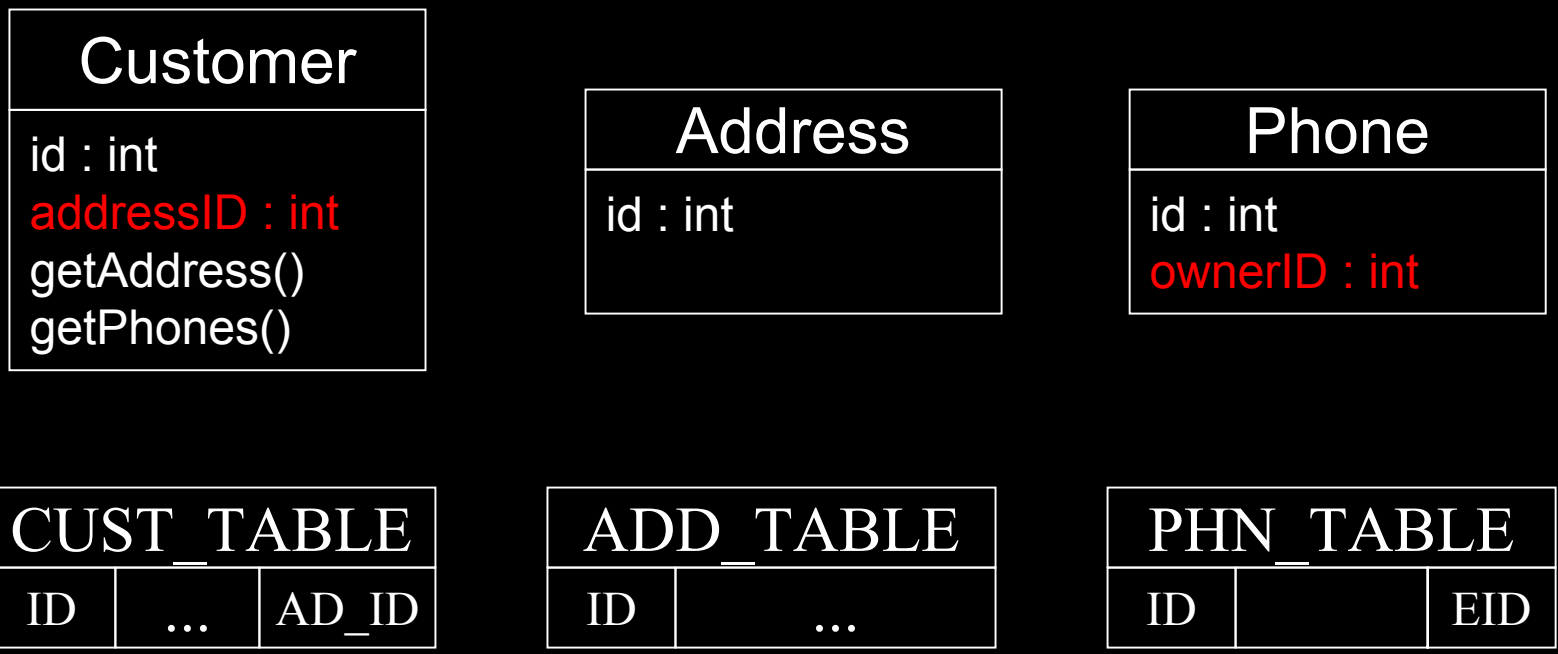
- Good mapping support:
  - Business classes don't have to be "tables"
  - References should be to objects, not foreign keys
  - Database changes (schema and version) easily handled.
- Poor mapping support:
  - Classes must exactly mirror tables
  - Middle tier needs to explicitly manage foreign keys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

Mapping Tools

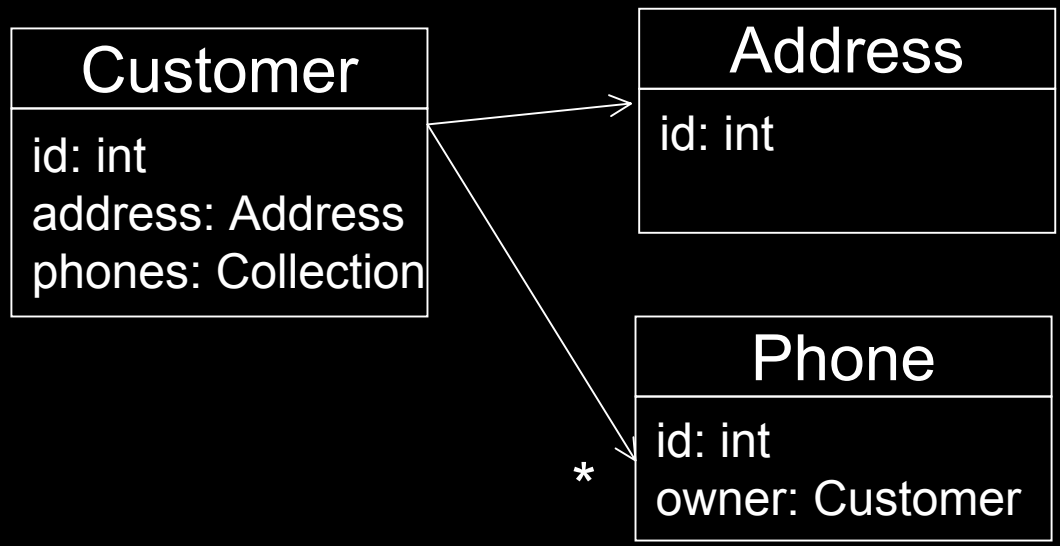
- Lots of mapping tools out there, however don't get fleeced by a slick GUI.
- The underlying mapping support is what's important



Business Objects Should Not Require Foreign Key Knowledge



Should Just Reference Objects Not Foreign Keys



CUST_TABLE		
ID	...	A_ID

ADD_TABLE	
ID	...

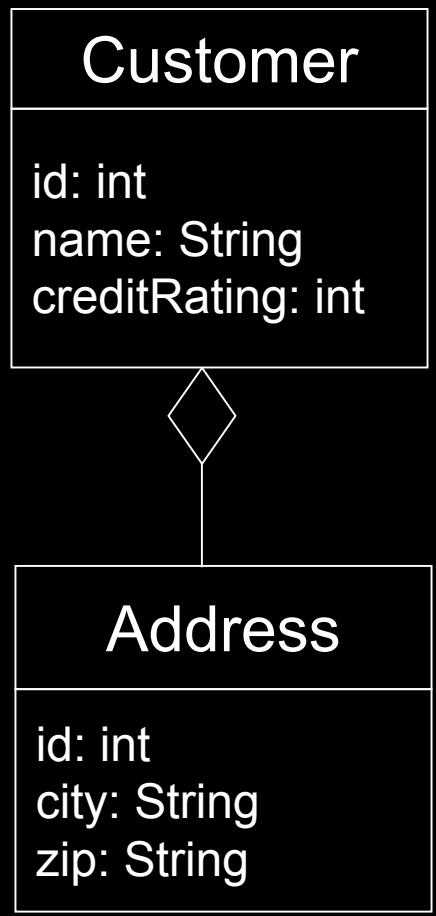
PHN_TABLE		
ID		E_ID

Data and Object Models

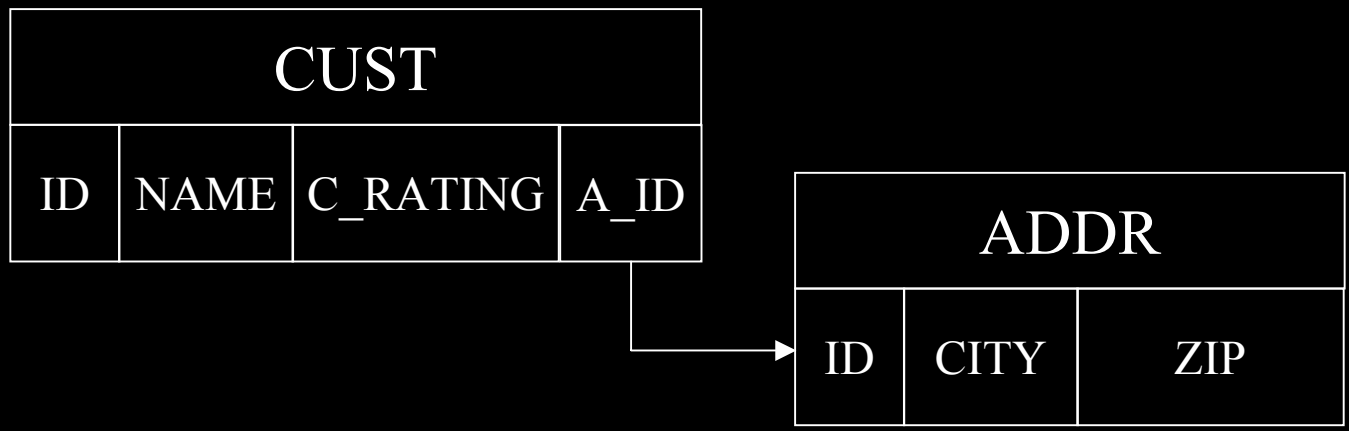
- Rich, flexible mapping capabilities provide data and object models a degree of independence
- Otherwise, business object model will force changes to the data schema or vice-versa
- Often, J2EE component models are nothing more than mirror images of data model – NOT desirable

Simple Object Model

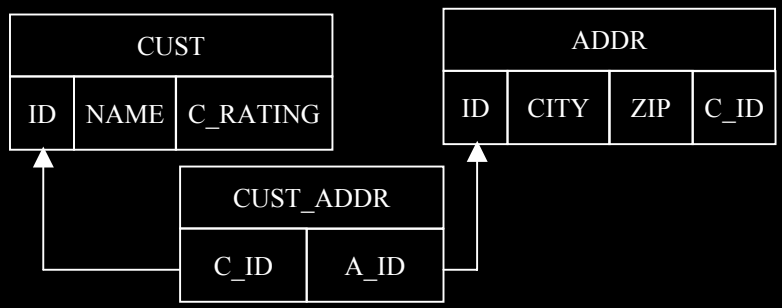
1:1 Relationship

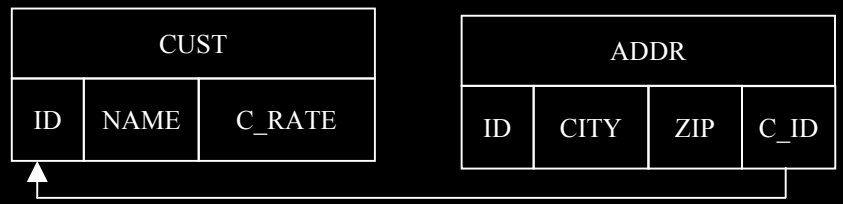


Typical 1-1 Relationship Schema

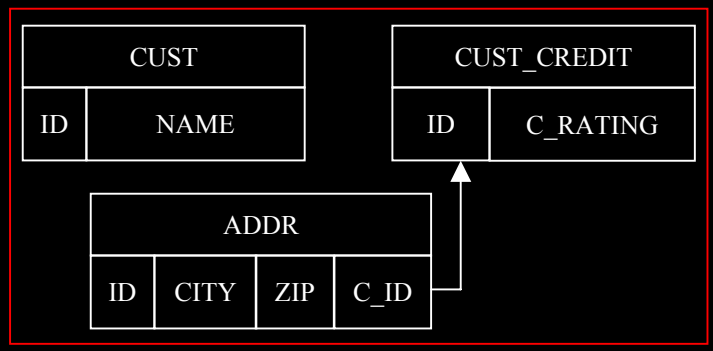


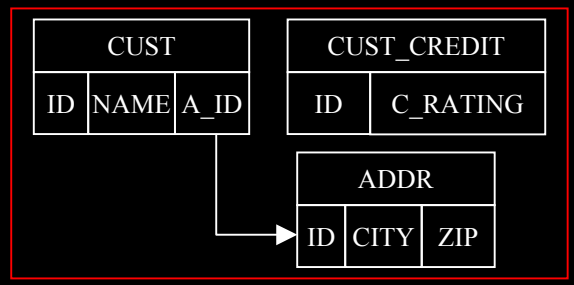
Other possible Schemas…

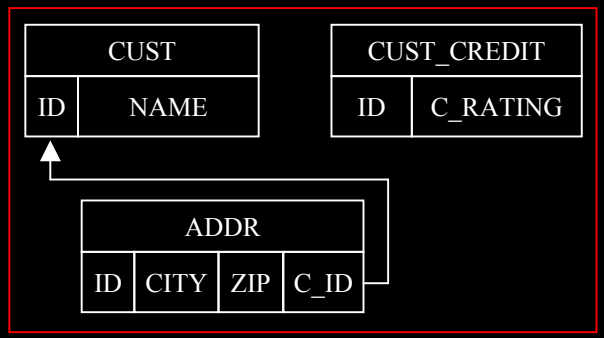


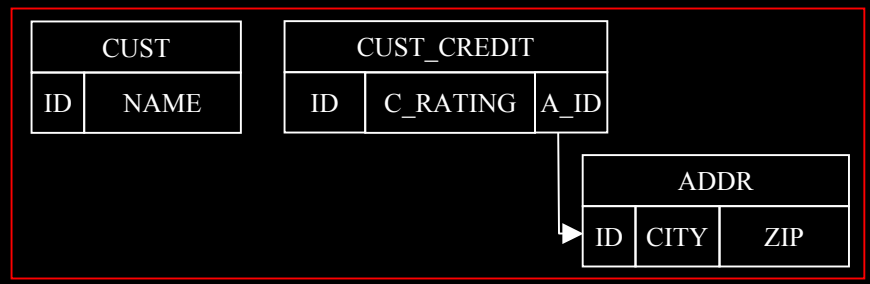


Even More Schemas…









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

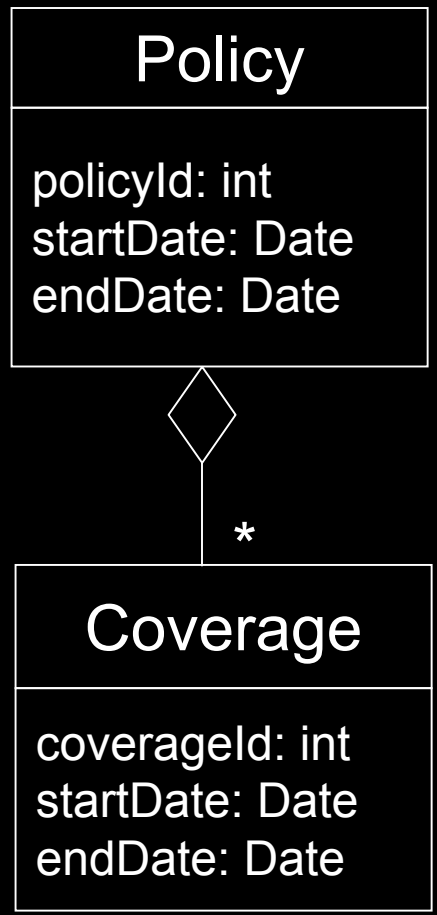
Difficult Case – “Historization”

- Composite primary key, consisting of “real” pkey and date range

Policy_ID	Start_Date	End_Date
1035	1/12/1999	2/5/2001
1035	2/6/2001	12/10/2002
1035	12/11/2002	1/1/2099

Insurance Historization Example

- Mapping is static, but what objects to recover is based on dynamic information.
- Can be done, but not very transparently, especially for relationships.



General J2EE Persistence Interaction

- Application business objects/components are modeled and mapped to relational data store
- Data is read from database and business objects/Entity Beans are created
- Objects are traversed, edited, created, deleted, cached, locked etc
- Changes stored on the database
- Multiple concurrent clients sharing database connections

Reading - 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

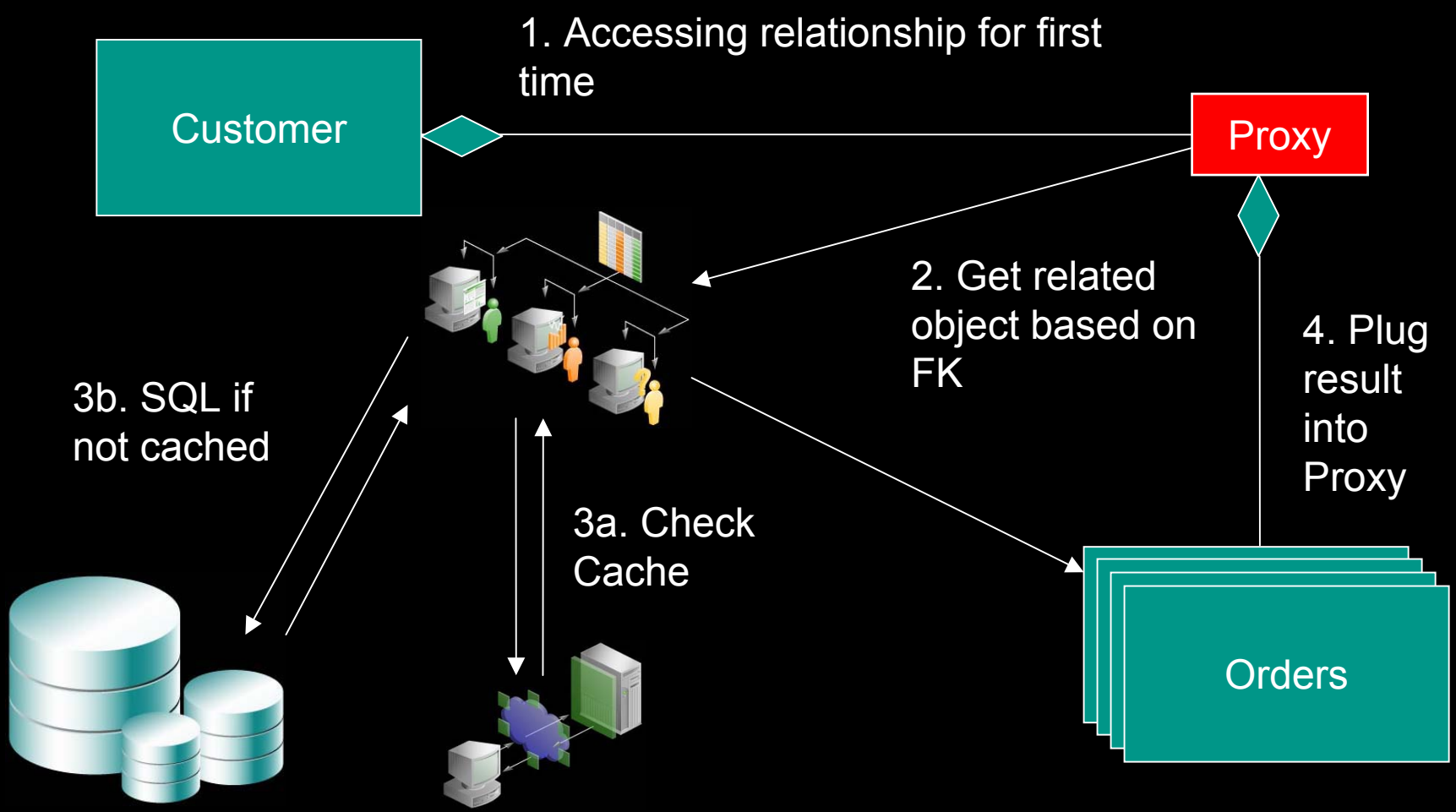
Query Requirements

- 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

Object Traversal – Lazy Reads

- 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



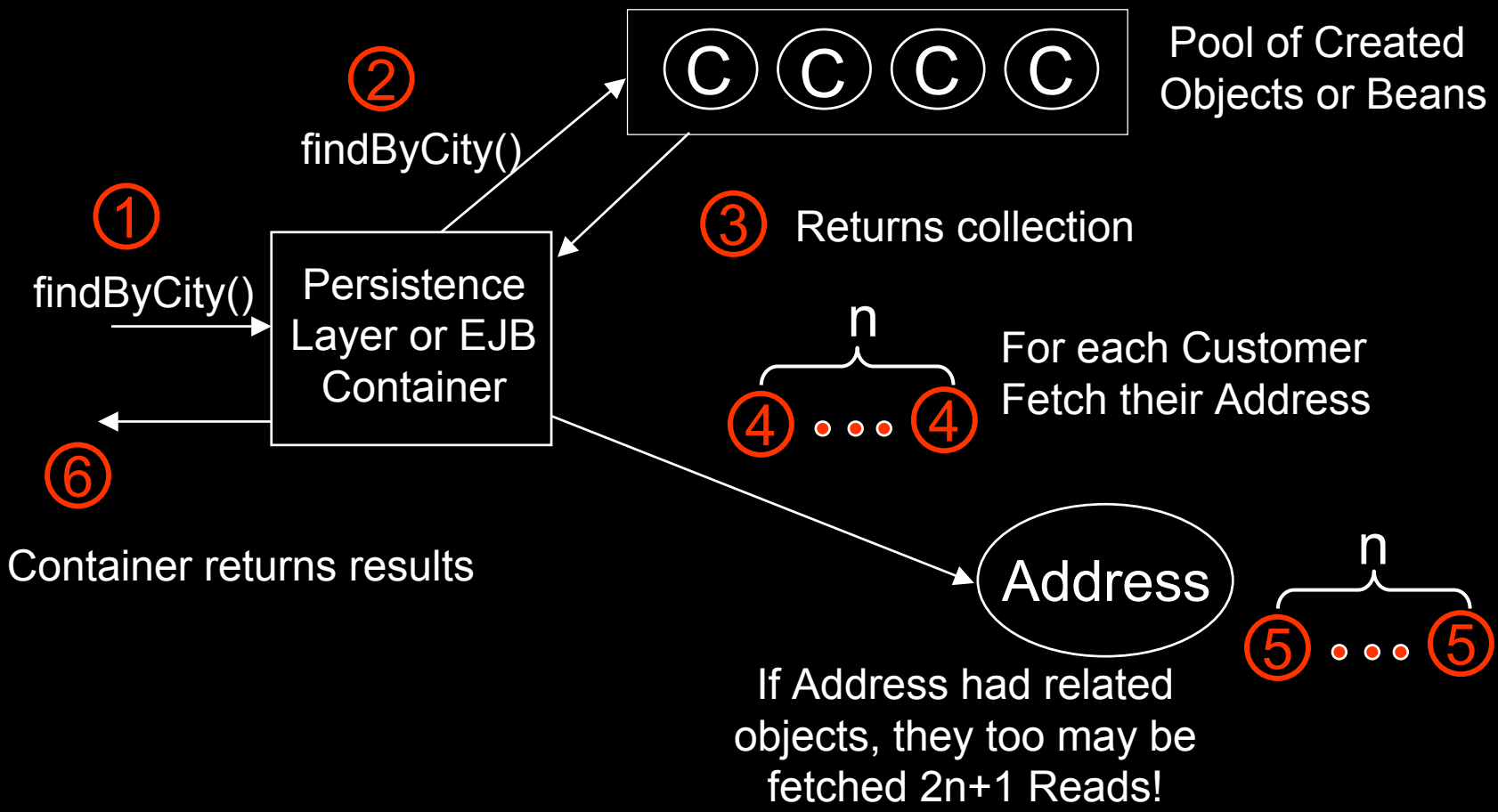
Object Traversals

- 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 Problem



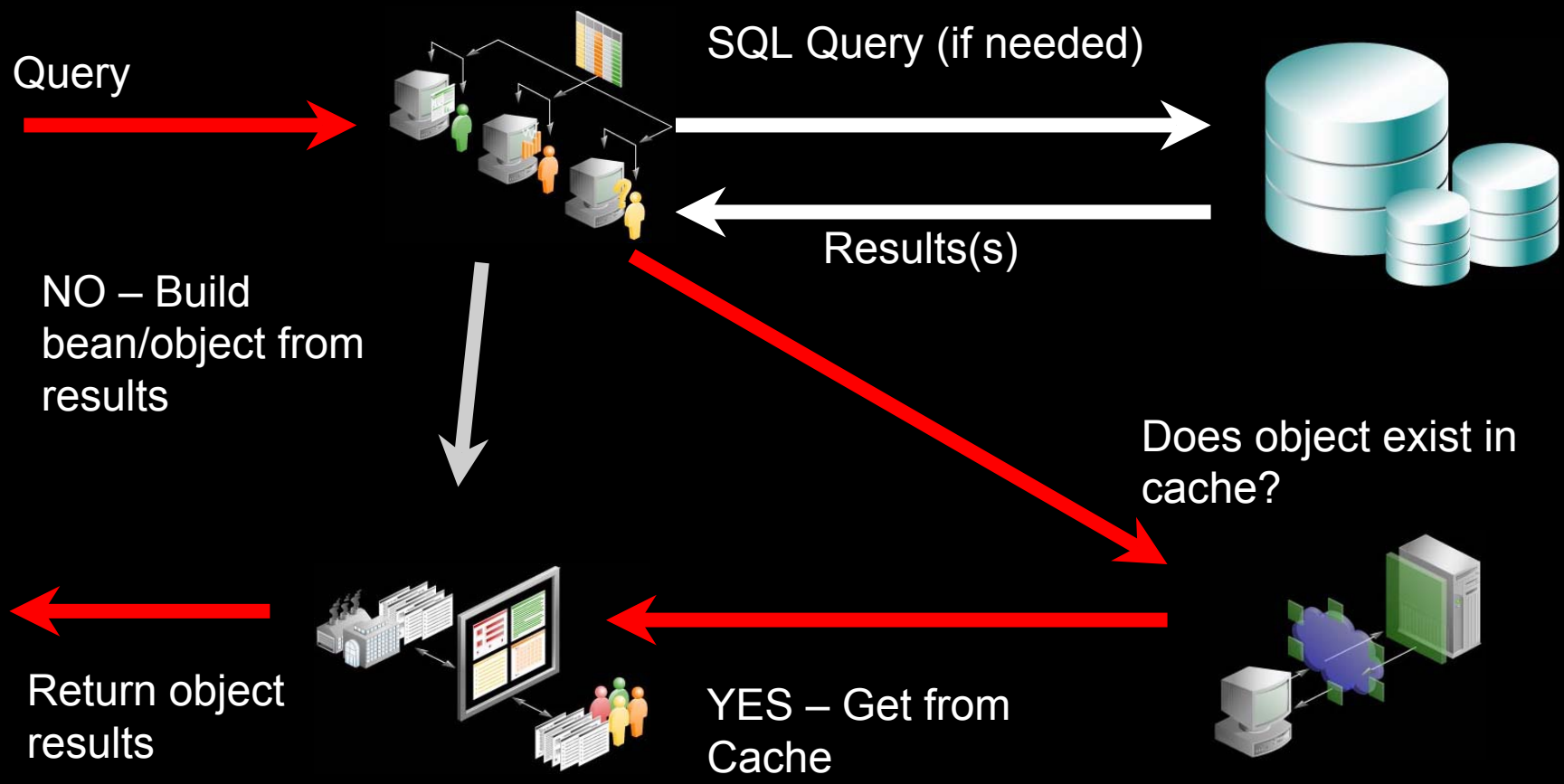
N+1 Reads

- 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

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

Caching


Query
SQL Query (if needed)
Results(s)
NO – Build bean/object from results
Does object exist in cache?
Return object results
YES – Get from Cache

Database Triggers

- Database triggers will be completely transparent to the J2EE application.
- However, their effects must be clearly communicated and considered.
- Example: Data validation –> audit table
  - Objects mapped to an audit table that is only updated through triggers, must be read-only on J2EE

Database Triggers

- More challenging when trigger updates data in the same row and the data is also mapped into an object.
- Example: Annual salary change automatically triggers update of life insurance premium payroll deduction
  - J2EE app would need to re-read payroll data after salary update OR
  - Duplicate business logic to update field to avoid re-read. Saves a DB call but now business logic in 2 places.

Referential Integrity

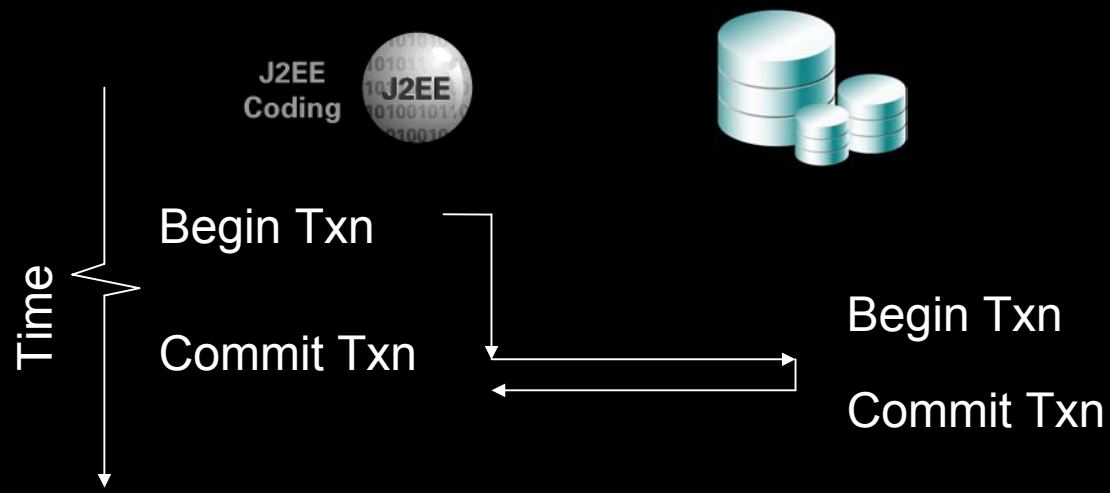
- Java developers manipulate object model in a manner logical to the business domain
- May result in ordering of INSERT, UPDATE and DELETE statements that violate database constraints
- Persistence layer should automatically manage this and allow options for Java developer to influence order of statements

Cascaded Deletes

- Cascaded deletes done in the database have a real effect on what happens at J2EE layer.
- Middle tier app must:
  - Be aware a cascaded delete is occurring
  - Determine what the “root” object is
  - Configure persistence settings or application logic to avoid deleting related objects already covered by cascaded delete.

Transaction Management

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



Locking

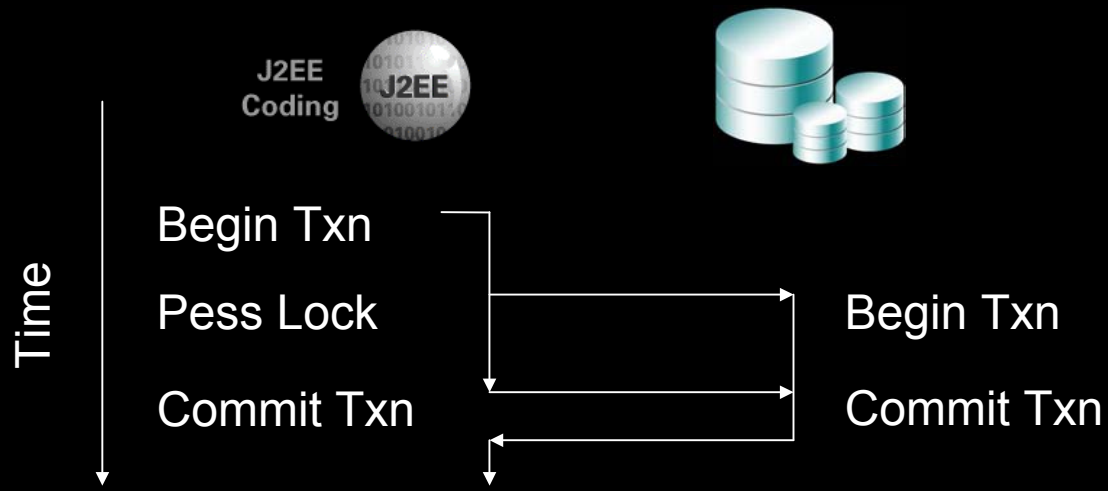
- J2EE Developers want to think of locking at the object level
- Databases may need to manage locking across many applications
- Persistence layer or application server must be able to respect and participate in locks at database level

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



Other Issues

- Use of special types
  - BLOB, Object Relational
- Open Cursors
- Batch Writing
- Sequence number allocations
- Database portability
- Inheritance
- Many, many more …

J2EE Apps & RDB Summary

1. Project teams should involve DBAs early
2. Don’t need to compromise object/data model
3. Need to fully understand what is happening at database level
4. Can utilize database features
5. Do not have to hard code SQL to achieve optimal database interaction
6. Can find solutions that effectively address persistence challenges and let them focus on J2EE application


QUESTIONS
ANSWERS