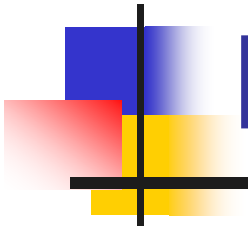
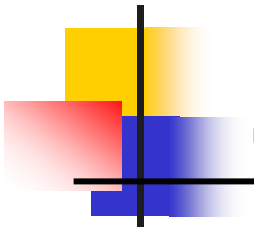


Persisting Java

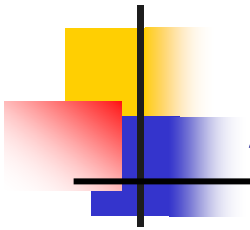
Donald Smith

Oracle Corporation



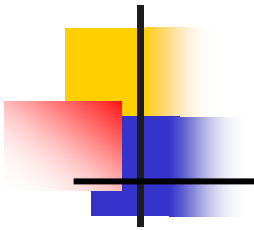
Speaker's Qualifications

- Decade of experience in OO Persistence
- Presented at Java One, Oracle World, OOPSLA, JAOO, Sun Tech Days, TheServerSide Symposium, *etc.*
- Author of numerous articles on persistence challenges



About the Audience…

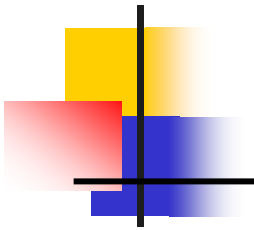
- Who considers themselves first and foremost to be a DBA or "Database expert"?
- Who considers themselves first and foremost to be a Java developer?
- Who considers themselves first and foremost to be an Architect?
- Who considers themselves first and foremost to be a manager, and will you admit it?



Purpose of This Session

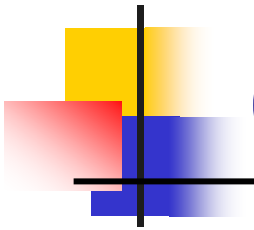
Learn and understand the impact of using J2EE with Relational Databases.

Understand issues and criteria for making effective persistence decisions.



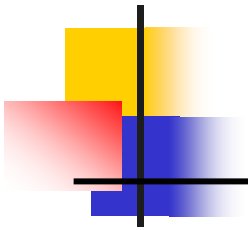
Purpose of This Session

AKA “How to befriend your DBA, an insiders guide for J2EE Developers/Architects/Managers”



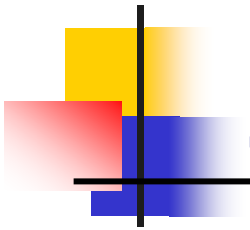
Call To Action

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



Agenda

- Ways to access RDB from J2EE
- Impedance Mismatch
- J2EE *vs* DBA desires
- J2EE Persistence Checklist
  - Mapping
  - Just-in-time reading, N+1 Reads
  - Queries
  - Transactions, Locking
  - Caching
  - Triggers
  - Other issues!

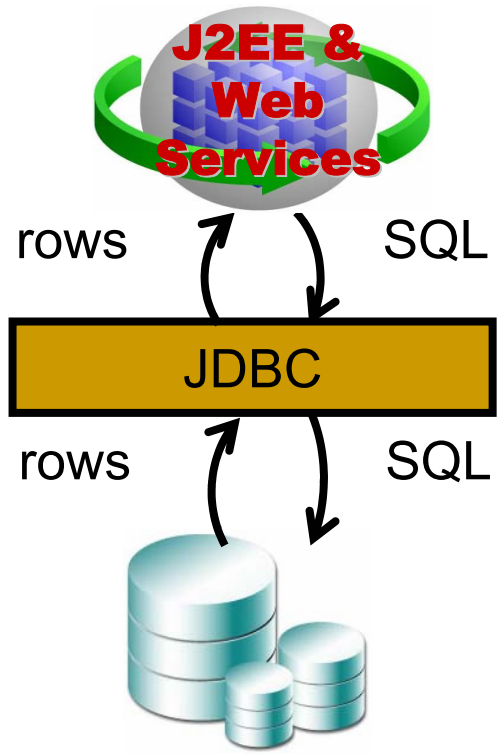


J2EE Access of Relational Data

- Direct JDBC
  - Direct SQL calls
  - Use rows and result sets directly
- Persistence Layer
  - Accessed as objects or components
  - Transparent that the data is stored in RDB
  - Persistence layer in middle tier handles object-relational mapping and infrastructure
  - Required if doing business logic in the middle tier!
  - Focus of this Session

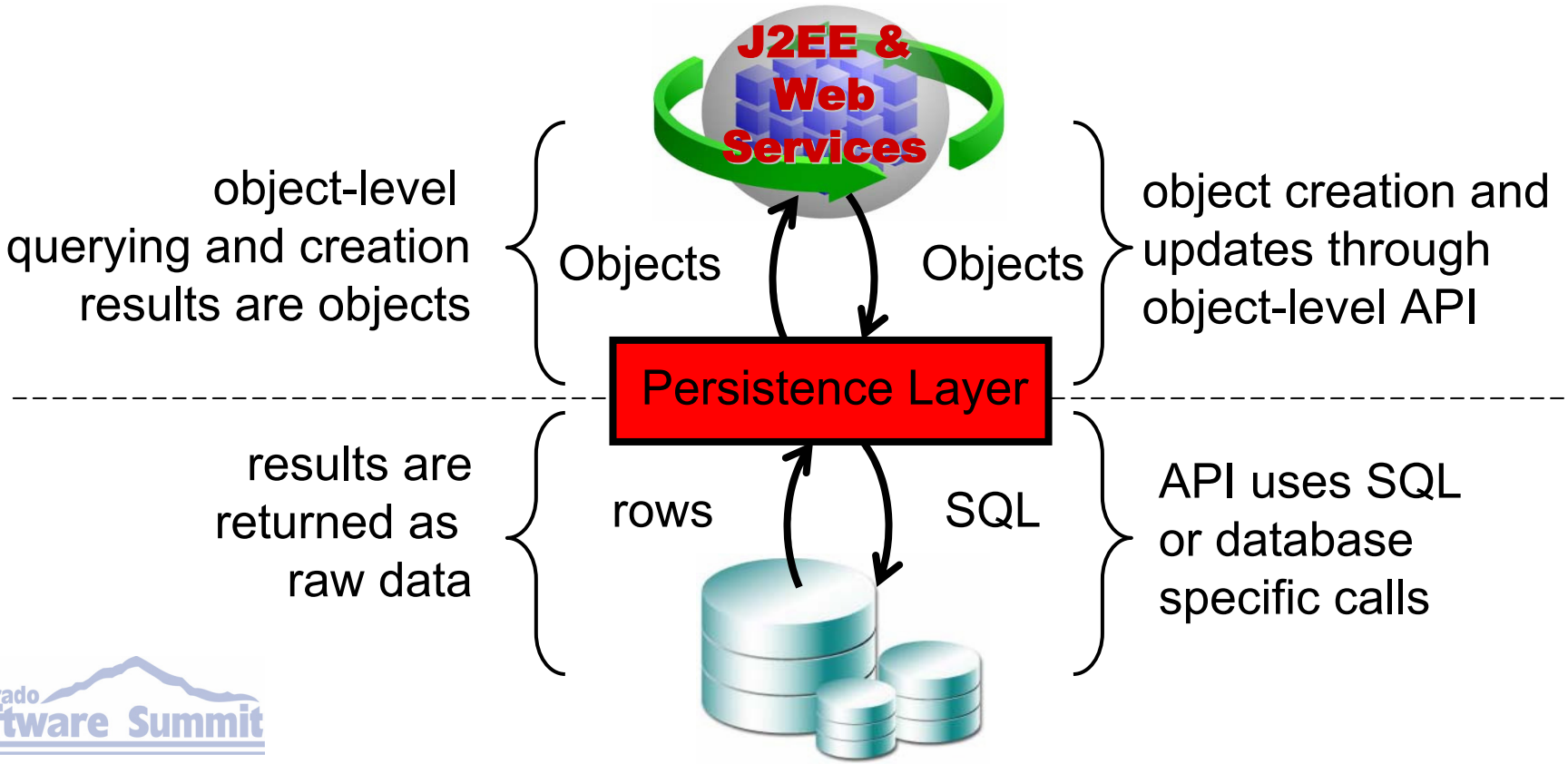
Direct JDBC

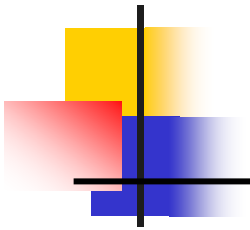
- JDBC is NOT a persistence framework
  - JDBC is a database connection utility
  - Ok for "Window on data" style application
  - Ok when business logic is entrenched on database
  - Little impact on RDB
  - J2EE becomes nothing more than a presentation layer…



Object Persistence Layer

- Abstracts persistence details from the application layer



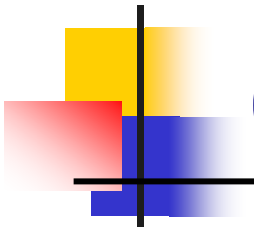


Impedance Mismatch

- 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

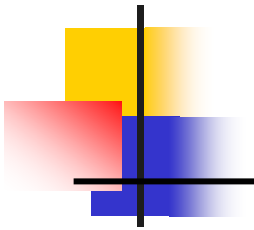
Impedance Mismatch

Factor	J2EE	Relational Databases
Logical View Of Data	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, *etc.*	Schema designer, query manager, database configuration, *etc.*



Object Level Options

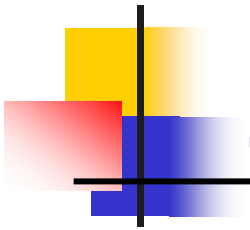
- Depends on what component architecture is used:
  - Entity Beans BMP – Bean Managed Persistence
  - Entity Beans CMP – Container Managed Persistence
  - POJO – "Plain Ol' Java Objects" *via* Persistence Layer
    - May be "home-grown" following "DAO" Data Access Object patterns
    - May use JDO – "Java Data Objects"
    - May use off the shelf products



Entity Beans or POJO?

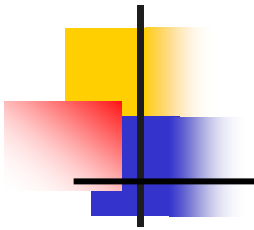
- Hot topic – Should you use Entity Beans or POJO?
- If we use Entity Beans – CMP or BMP?
- If we use POJO – DAO, JDO, 3rd Party?

Not relevant to this discussion! The issues to be discussed apply regardless of component type or architecture used.



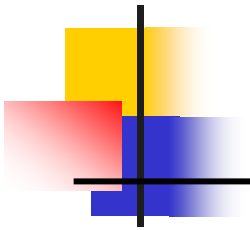
J2EE Developer Desires

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



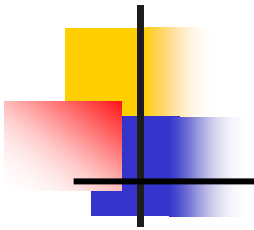
DBA Desires

- 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 profile all SQL calls to database
- Leverage database features where appropriate (outer joins, sub queries, specialized database functions)



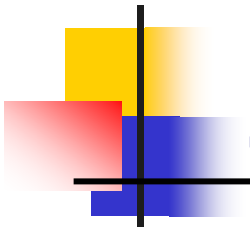
Differences

- 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!



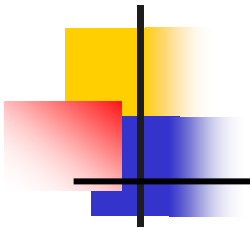
How Are Databases Affected?

- In practice, it's usually the other way around, J2EE app is influenced by database…
  - Database "rules" need to be followed
  - Database interaction must be optimized for performance
  - "Source of truth" for data integrity is database, not app server



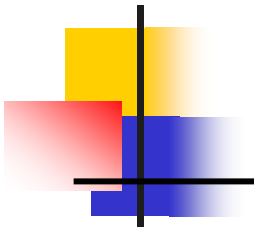
J2EE Persistence Checklist

- Mappings
- Object Traversal
- Queries
- Transactions
- Optimized database interaction
- Locking
- Caching
- Database features



Mapping

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

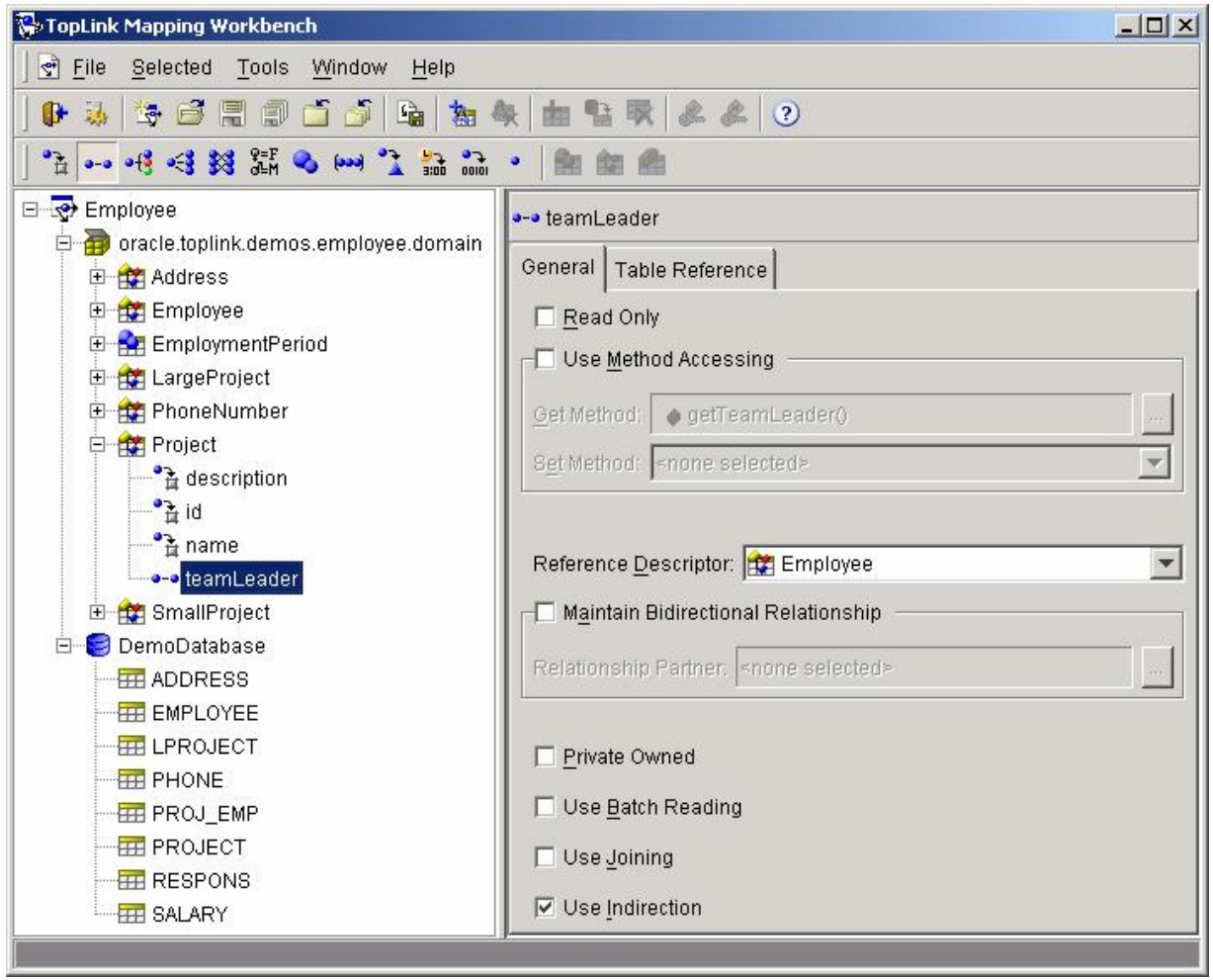


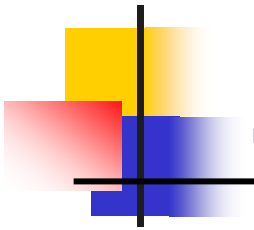
Data and Object Models

- 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 or *vice-versa* – NOT desirable

Mapping Tools

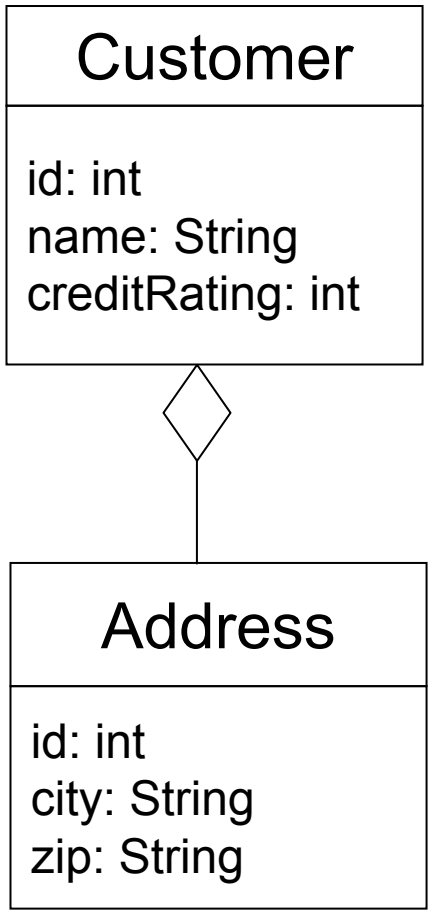
- The underlying mapping flexibility is very important



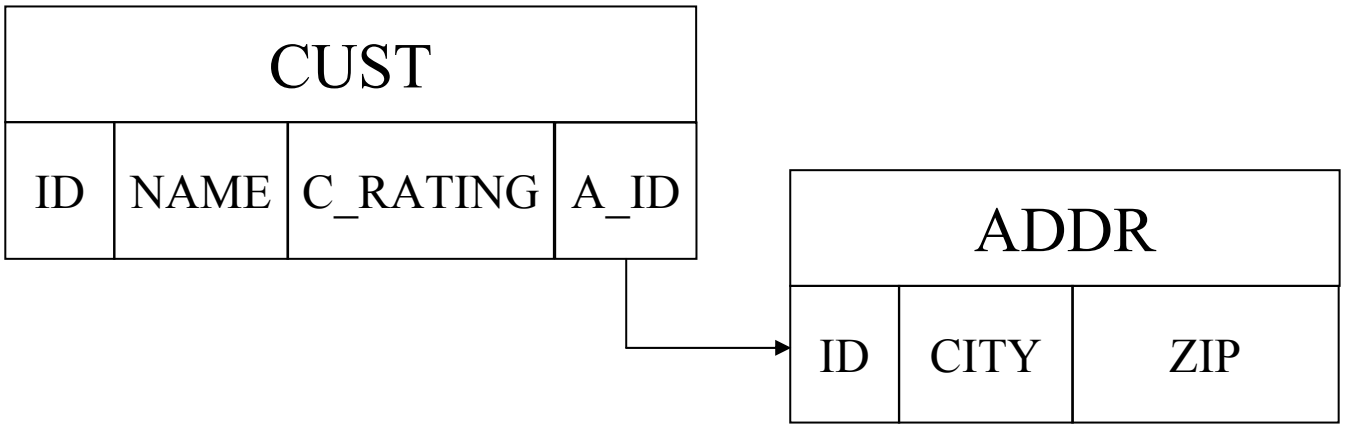


Simple Object Model

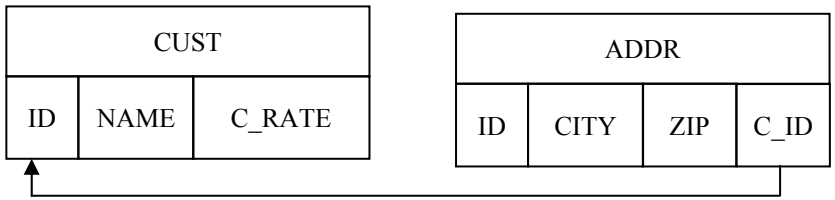
1:1 Relationship

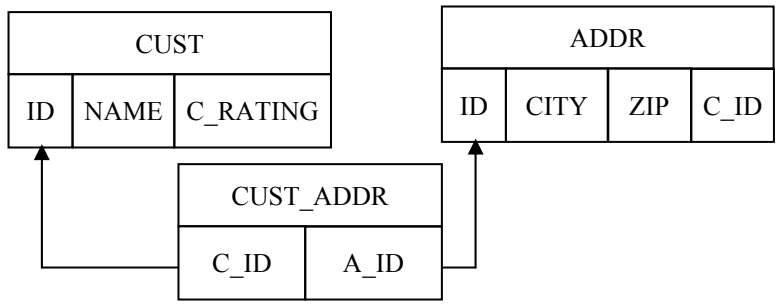


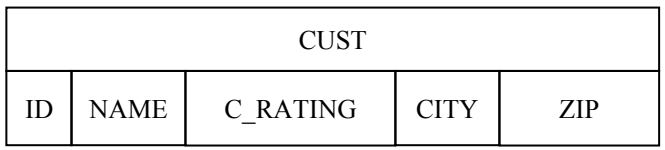
Typical 1-1 Relationship Schema



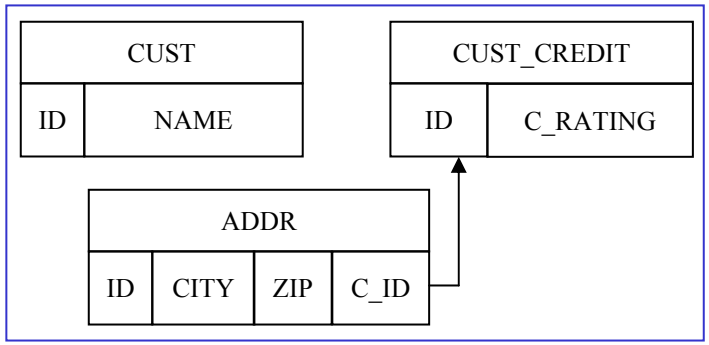
Other Possible Schemas...

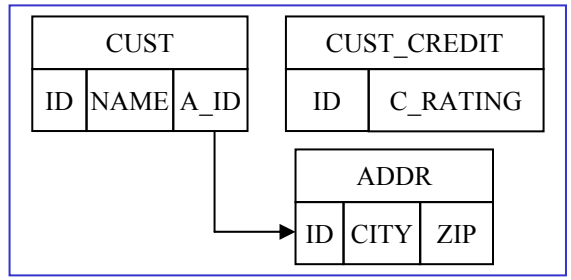


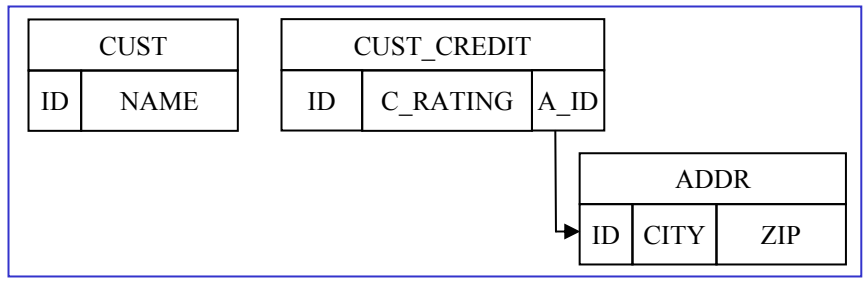


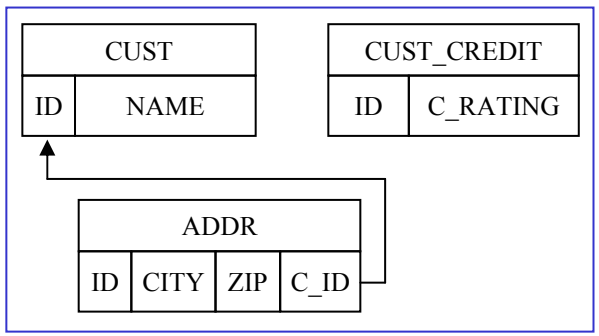


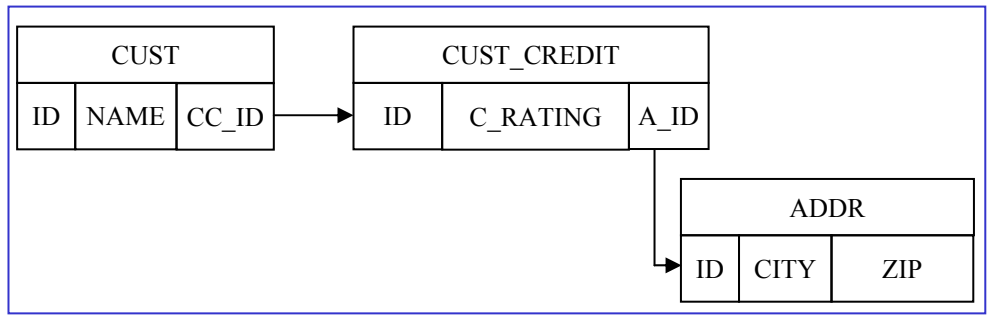
Even More Schemas...

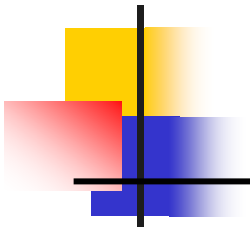






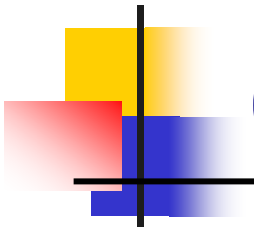






Mapping Summary

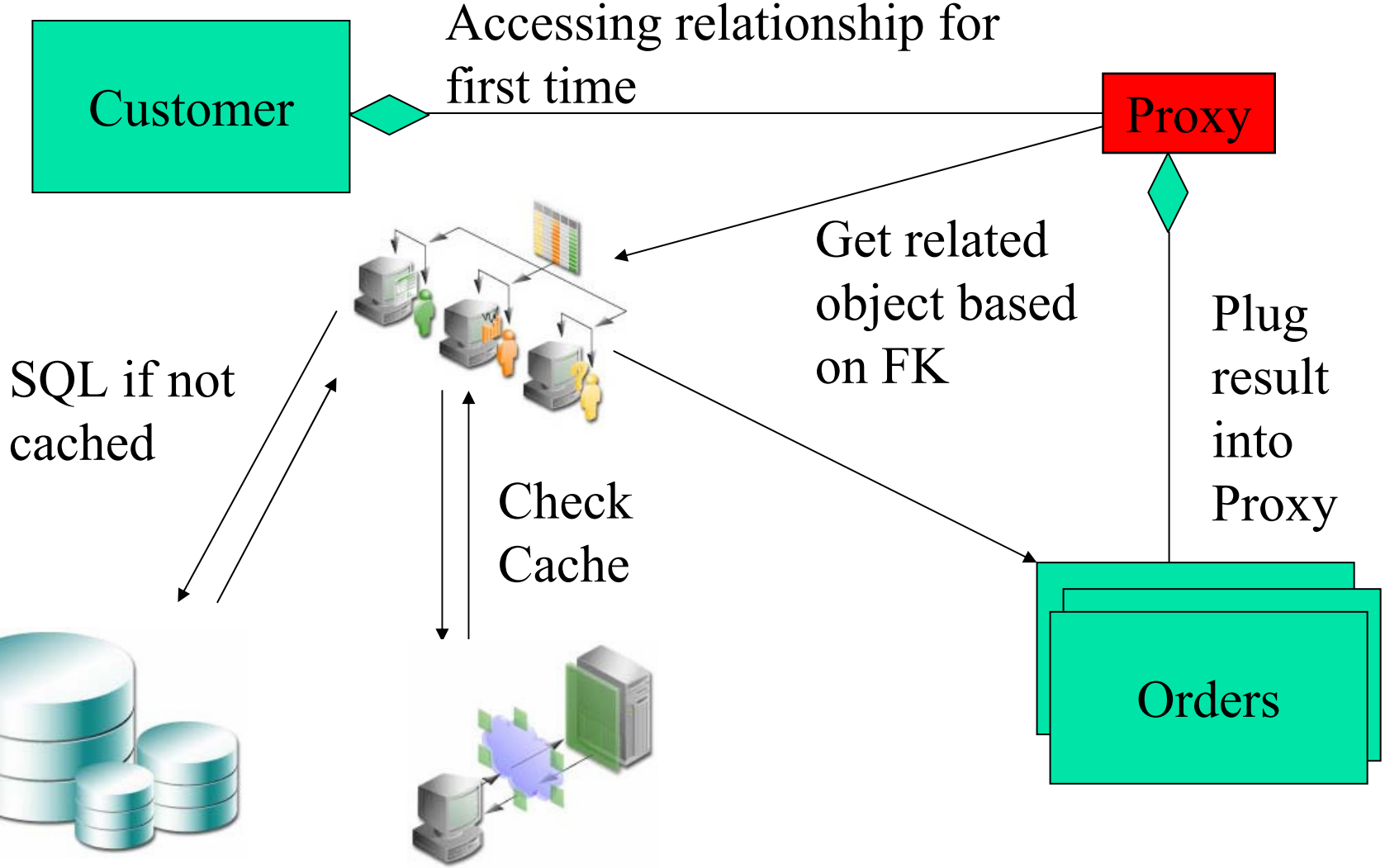
- Just showed **nine** valid ways a 1-1 relationship could be represented in a database
  - Most persistence layers and application servers will only support *one*
- Without good support, designs will be forced
- Imagine the flexibility needed for other mappings like 1-M and M-M

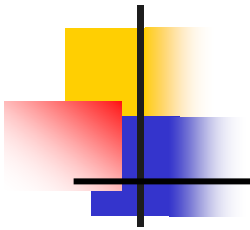


Object Traversal – Lazy Reads

- 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"

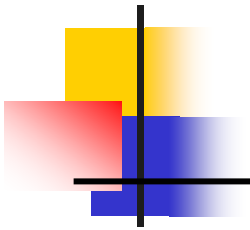
Just in Time Reading – Faulting Process





Object Traversals

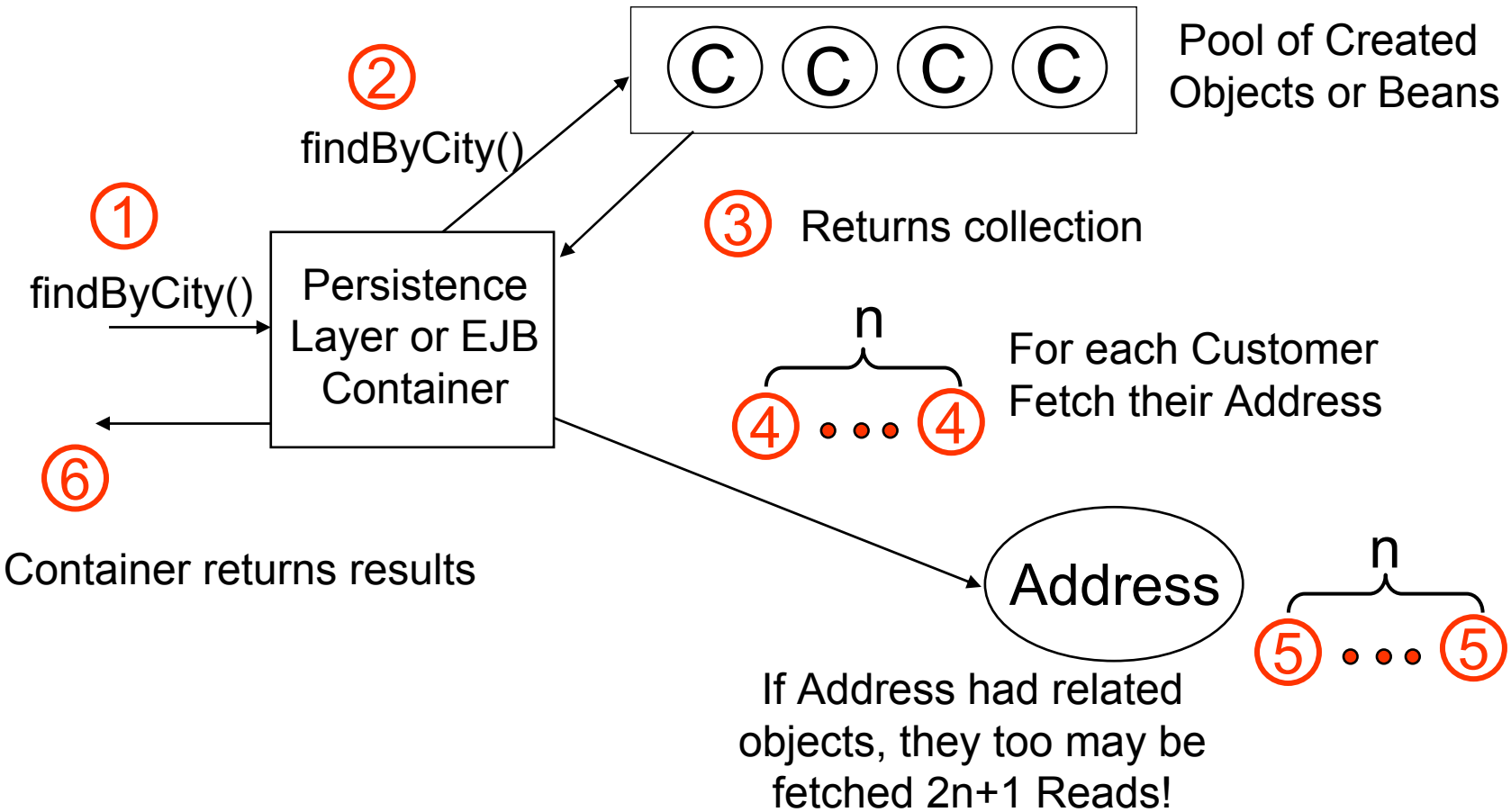
- Even with lazy reads, object traversal is not always ideal
  - To find a phone number for the manufacturer of a product that a particular customer bought, may do several queries:
    - Get customer in question
    - Get orders for customer
    - Get parts for order
    - Get manufacturer for part
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

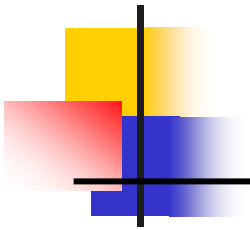


N+1 Reads Problem

- 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

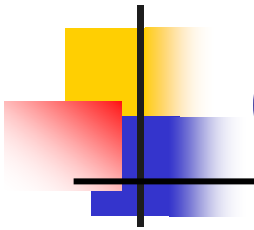
N+1 Reads Problem





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



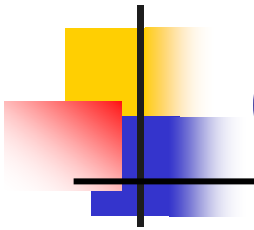
Queries

- 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

```
employee.manager.address = someAddress
```

Generates…

```
SELECT * FROM EMP t1, EMP t2, ADDR t3
WHERE t1.MGR_ID = t2.EMP_ID AND
t2.ADDR_ID = t3.ADDR_ID AND t3.ADDR_ID =
<someAddress.id>
```

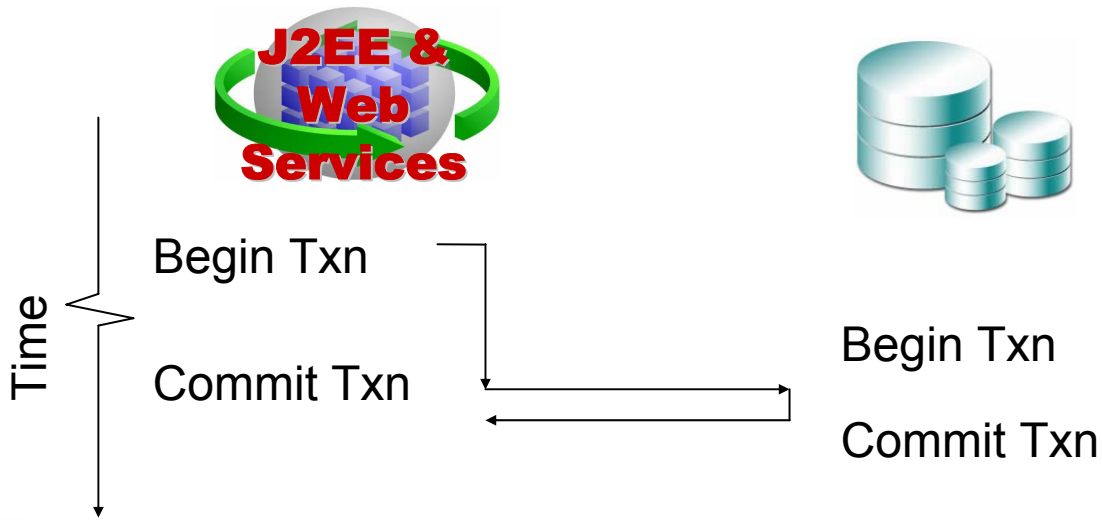


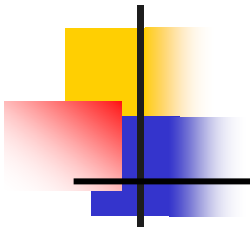
Query Requirements

- 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

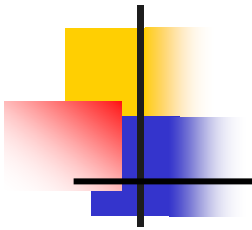
- J2EE apps typically support many clients sharing small number of db connections
- Ideally would like to minimize length of transaction on database





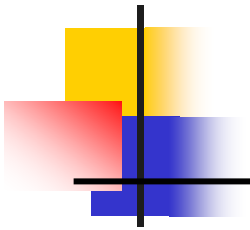
Database Triggers

- 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



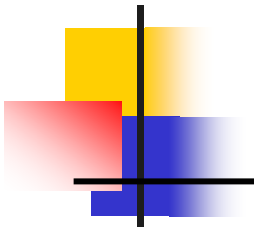
Database Triggers

- 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



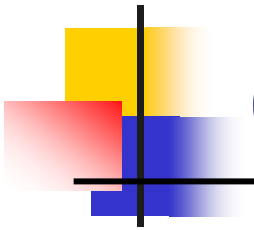
Cascaded Deletes

- 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



Referential Integrity

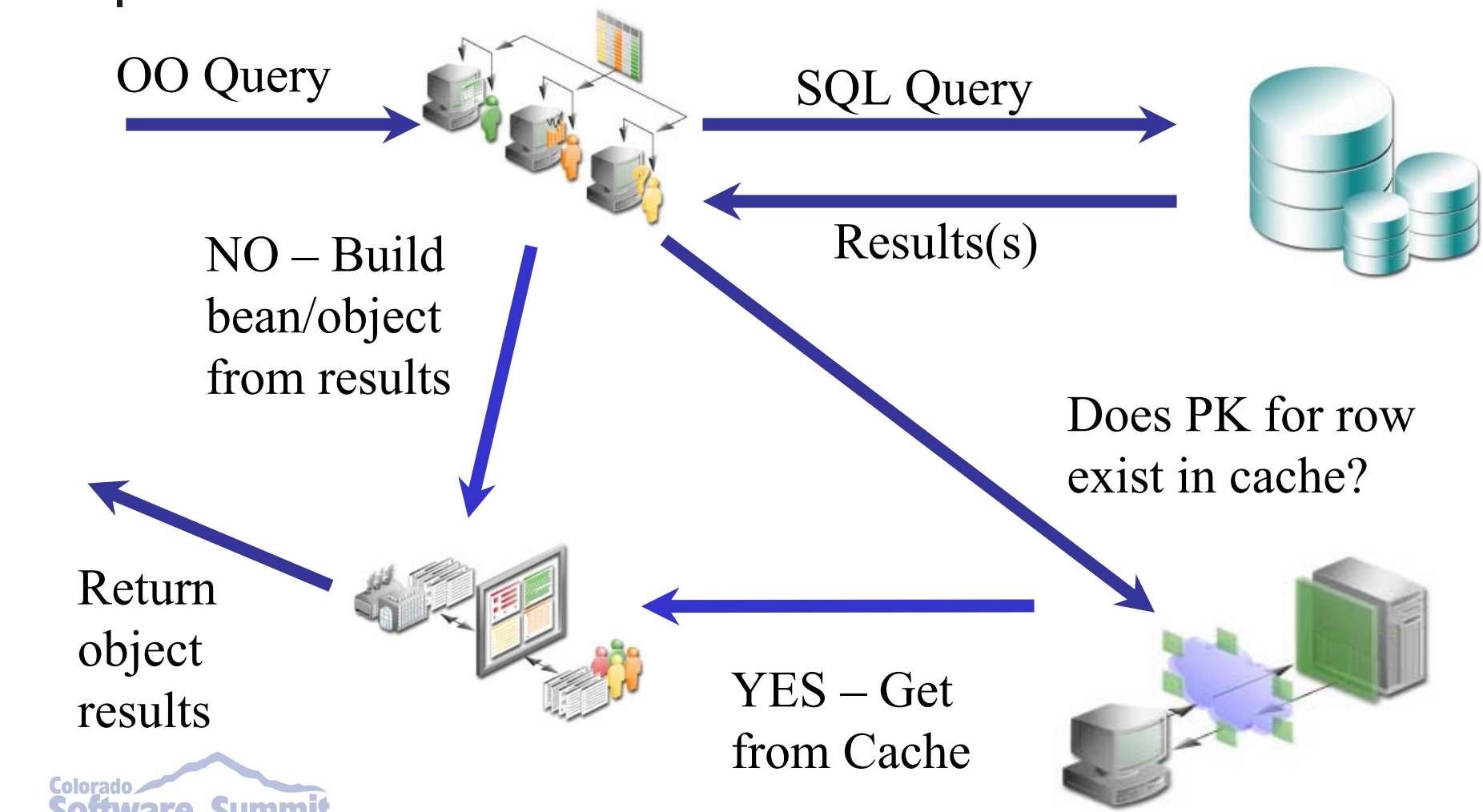
- Java developers manipulate object model in a manner logical to the business domain
- May result in ordering of INSERT, UPDATE and DELETE statements that violate database constraints
- Persistence layer should automatically manage this and allow options for Java developer to influence order of statements

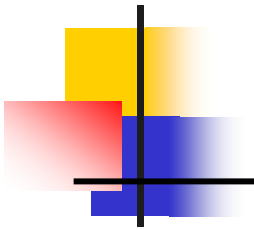


Caching

- 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

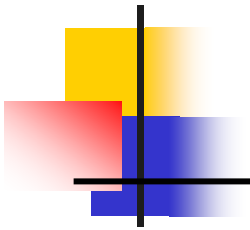
Caching





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

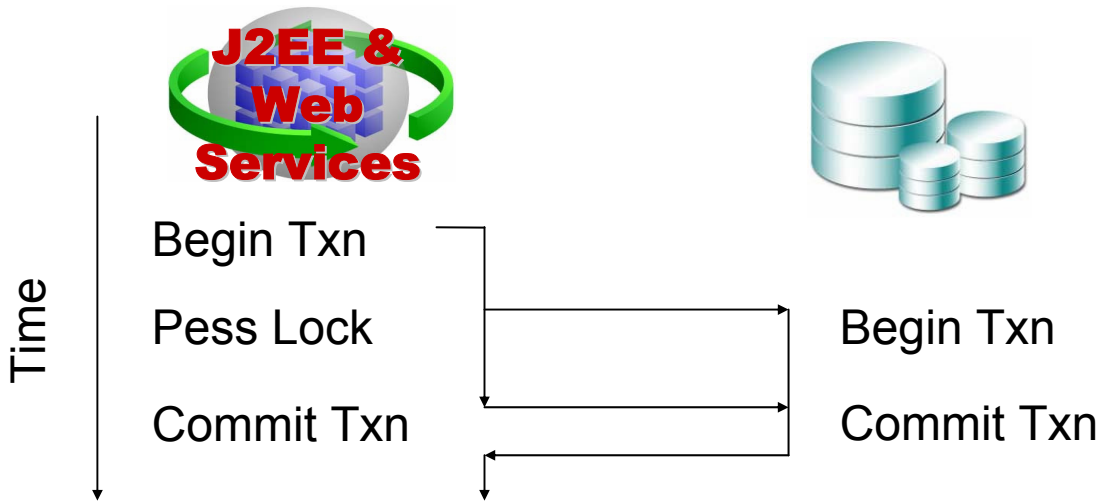


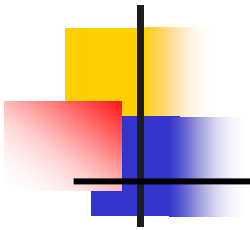
Optimistic Locking

- 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

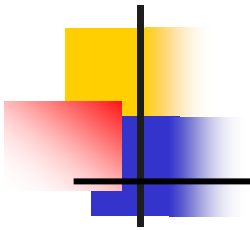
- Requires careful attention as a JDBC connection is required for duration of pessimistic lock
- Should support SELECT FOR UPDATE [NOWAIT] semantics





Other Impacts

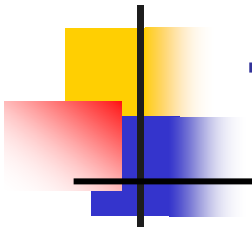
- Use of special types
  - BLOB, Object Relational
- Open Cursors
- Batch Writing
- Sequence number allocations



Summary

- 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

If You Only Remember One Thing…



When choosing a persistence layer, pick one that maximizes your flexibility across all the issues outlined here!