

# 9 Embedding SQL into Programming languages



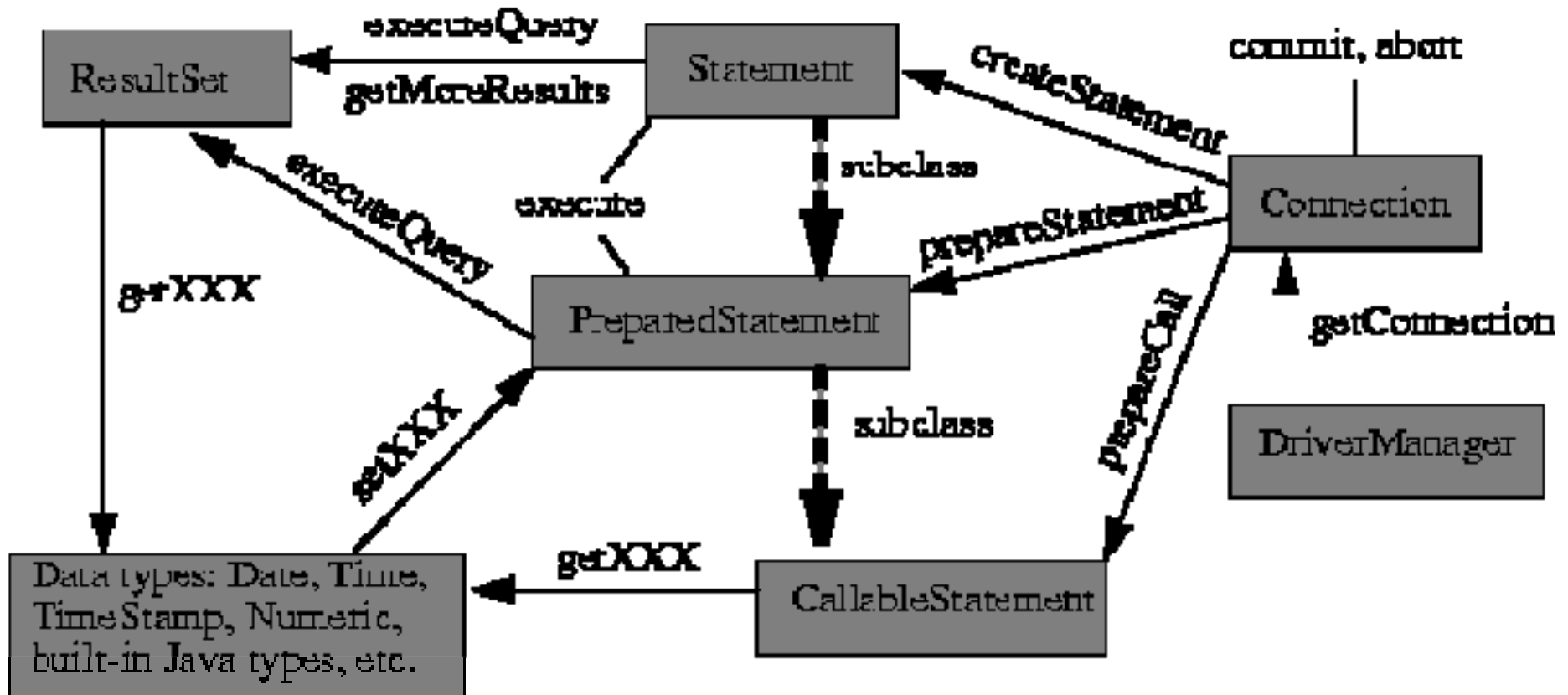
- 9.1 Introduction: using SQL from programs
- 9.2 Embedded SQL
  - Static and dynamic embedding
  - Cursors
  - ESQL / C
  - Positioned Update
- 9.3 SQL and Java
  - JDBC
  - SQLJ
- 9.4 OR mapping and components
- 9.5 Transactions in application programs
  - Definition
  - Isolation levels

Lit.: Kemper / Eickler: chap. 4.19-4.23;

Melton: chap. 12,13,17-19, Widom, Ullman, Garcia-Molina: chapt.8

JDBC Literature, e.g. Sun jdbc guide.(see: material)

# JDBC core interfaces



from Sun Microsystems: JDBC 2.0 API

# Making a connection

```
try {
    Class.forName("oracle.jdbc.OracleDriver"); }
catch (ClassNotFoundException e)
{System.out.println ("Driver not found");}
try {
    DriverManager.registerDriver (
        new oracle.jdbc.OracleDriver());
    return DriverManager.getConnection (url1,database, pw );}
catch (SQLException e){
    System.out.println("SQL-Execption: ");
}
}
```

New concept: `DataSource` interface:

- logical concept, avoids hard coded Driver / Connection details. Name registered with JNDI naming service, which supplies details
- support of connection pools and transactions

# Calling stored procedures

PL/SQL function (from above):

```
create or replace FUNCTION CountryLCity
  (cname IN VARCHAR, p IN int) RETURN int
  IS CURSOR ctry IS
  SELECT * FROM Country
  WHERE CODE LIKE cname || '%'
        and population >p;
  row# int;
BEGIN
  FOR resRecord IN ctry LOOP
    row# :=ctry%ROWCOUNT;
    dbms_output.PUT_LINE ('Name: ' ||
      resRecord.name || ', Capital: ' ||
      resRecord.capital);
  END LOOP;
  RETURN (row#);
END;/
```

# JDBC: calling stored procedures

```
try{
CallableStatement cs = con.prepareStatement
    ("{ ?=call CountryLCity(?,?)}");
cs.registerOutParameter(1, Types.INTEGER);
cs.setString(2,arg1);
cs.setInt(3,arg2);
cs.executeQuery();
int res = cs.getInt(1);
System.out.println("... "+ res + ", ");
} catch (SQLException ex) {
System.err.println("Error While
Executing Function Mondial.CountryLCity " +
ex.toString());
}
```

If stored procedure causes updates:

```
int rowCount= cs.executeUpdate();
```

# JDBC: Updates

Updating result sets (JDBC 2.0)

```
Statement stmt = con.createStatement(  
    ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_UPDATABLE);  
stmt.setFetchSize(25);  
ResultSet rs = stmt.executeQuery(  
    "SELECT emp_no, salary FROM employees");  
  
rs.first();  
rs.updateFloat("salary", 10000.0);  
rs.updateRow();
```

# JDBC connection pooling

## Opening a connection is expensive

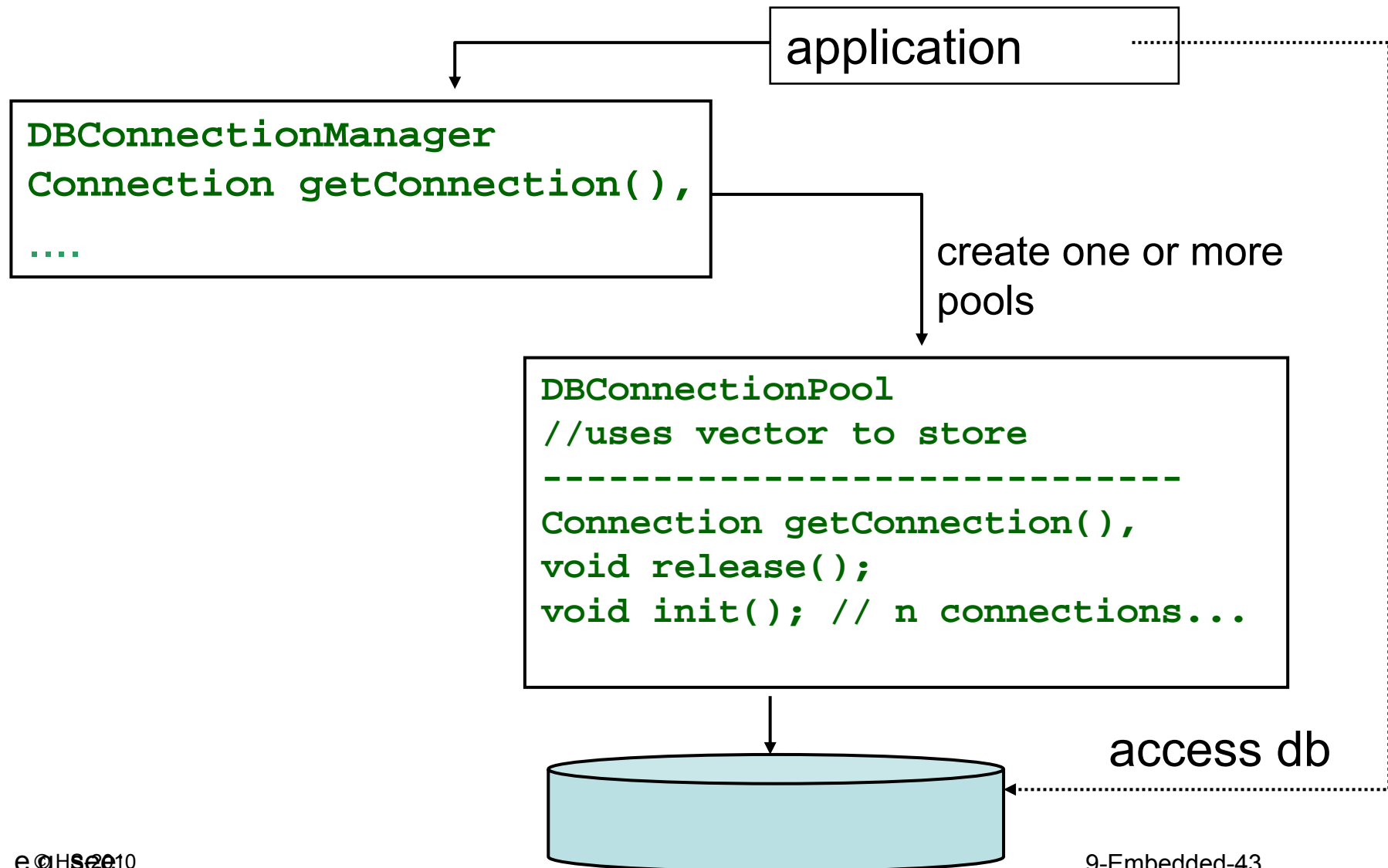
DB login, set up context, check rights, ...  
up to 1 sec in some systems

Typical interactive applications (e.g. web shop):  
many customers, few DB interactions  
⇒ heavy overhead caused by setup of  
DB connection

## Solution

use a set of pre-initialized connections  
**("connection pool")**

# JDBC connection manager





# Transactional processing...

... a short introduction

**Transaction: a unit of work** which consists of a **sequence of steps** (operations on the Database)

**Isolation** of independent application processes T1, T2,...

# Lost update

T1 read  $x = 3$

T2 read  $x = 3$

T1: write  $(x+1)$

T2 write  $(x+1)$

time



1. +2.: T1 and T2 read DB field/record into local variable,
- 2.+ 3. Both change locally and write back.

Isolation compromised

**Lost update!** Database corrupt.

Transaction isolation more demanding than concurrency in programs?

# Transaction semantics

DBS has to guarantee certain executional properties

## "All or nothing" semantics

All effects are made permanent at COMMIT, **not before** .

TA has no effect after ROLLBACK

ATOMICITY

## "Now and forever"

DBS guarantees the effects after COMMIT has been processed successfully

DURABILITY

## "Solve concurrency conflicts"

Conflict resolution of concurrent operations on DB

## "Keep consistent DB consistent"

Preservation of integrity

ISOLATION

CONSISTENCY

# Transactions

How does DB System guarantee the properties?

## Implementation of DBS

*Application programming:* DB transactions

Syntactically mark unit of work:

```
START TRANSACTION ..... COMMIT;
```

or:

```
START TRANSACTION .....  
    IF (everything_OK) COMMIT  
    ELSE ROLLBACK; ENDIF - no effect on DB
```

Exception thrown by system (Transaction abort) if application wants to commit, but DBS cannot guarantee proper execution

## Transactional properties of connections

JDBC constants:

`TRANSACTION_NONE`

`TRANSACTION_READ_UNCOMMITTED`

`TRANSACTION_READ_COMMITTED`

`TRANSACTION_REPEATABLE_READ`

`TRANSACTION_SERIALIZABLE`

Different  
isolation levels

### JDBC API:

- `public void setTransactionIsolation(int level) throws SQLException`
- `public void setAutoCommit(boolean autoCommit)`  
-- no autocommit when updating result sets!
- `public void commit()`  
throws `SQLException`
- `public void rollback()`  
throws `SQLException`

# JDBC: programming style

- Separate DB interaction and processing
- SQL code goes into a separate object
- Encapsulate DB execution and error handling from application processing
  
- Many advantages, but not always a trivial task.

## 9.4.2 SQLJ

### Part 0: SQLJ [Embedded SQL](#)

Mostly reviewed and implemented

Integrated with JDBC API

Oracle has placed Translator source into public domain

### Part 1: SQLJ [Stored Procedures and UDFs](#)

Using Java static methods as SQL stored procedures & functions

Leverages JDBC API

### Part 2: SQLJ [Data Types](#)

Pure Java Classes as SQL ADTs

Alternative to SQL3 Abstract Data Types

# SQLJ and JDBC

```
// SQLJ
int n;
#sql { INSERT INTO emp VALUES (:n) };
```

```
// JDBC
int n;
Statement stmt = conn.prepareStatement
    ("INSERT INTO emp VALUES (?)");
stmt.setInt(1,n);
stmt.execute ();
stmt.close();
```





```
CREATE TABLE PARTS_MASTER
(PART_ID NUMBER(8) PRIMARY KEY,
PART_NAME VARCHAR(40),
SUPPLIER VARCHAR(200));
```

```
CREATE TABLE MRP
(PART_ID REFERENCES PARTS_MASTER,
QUANTITY_ON_HAND NUMBER(6),
REORDER_THRESHOLD NUMBER(6));
```

```
// Part of a SQLJ program, showing definition of one method:
```

```
public class inventory {
```

```
...
```

```
public void pullStock (int part, int quantity)
```

```
throws OutOfStock {
```

```
int on_hand, threshold;
```

```
#sql { SELECT QUANTITY_ON_HAND, REORDER_THRESHOLD
        INTO :on_hand, :threshold FROM MRP
        WHERE PART_ID = :part FOR UPDATE };
```



```
on_hand -= quantity;
if (on_hand < threshold) {
    String supplier;
    #sql { SELECT SUPPLIER INTO :supplier
           FROM PARTS_MASTER
           WHERE PART_ID = :part };
    inventory.orderMore(part, quantity, supplier);
}
if (on_hand < 0) {
    #sql { ROLLBACK };
    throw new OutOfStock();
} else {
    #sql { UPDATE MRP SET QUANTITY_ON_HAND = :on_hand
           WHERE PART_ID = :part };
    #sql { COMMIT };
}
}
...
}
```

## 9.5 OR-Mapping

### Motivation

**Hide details** of DB schema from application program

typical usage:

- application started from scratch
- application is rather simple: few persistent classes
- **Relational schema is generated from persistent classes**
- **Mapping** between tables and persistent classes is **generated**
- Most simple approach:
  - **"row type" (attributes) = class**
  - **"row" = "object"**

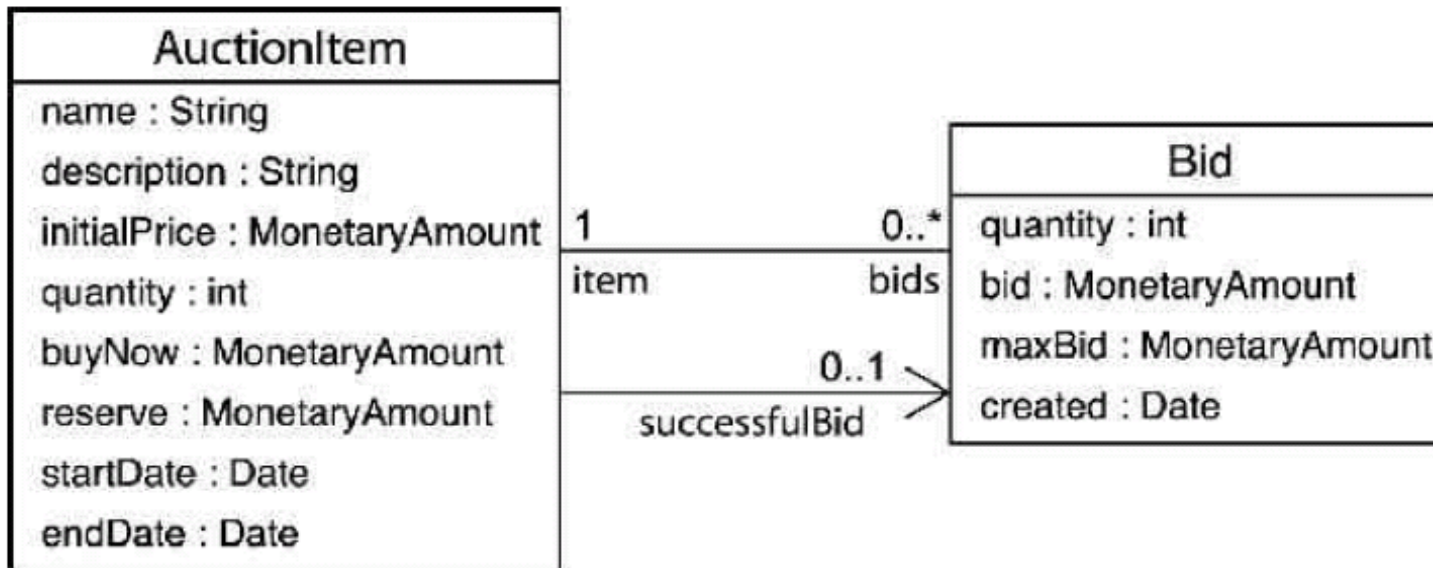
Systems: **Hibernate**, Object Relational Bridge (ORB)  
and many more

# Goal of OR-Mapping

Take advantage of the things SQL databases do well, without leaving the Java (?) language of objects and classes.

# OR-mapping with Hibernate example

From Classes to tables,  
from rows to objects



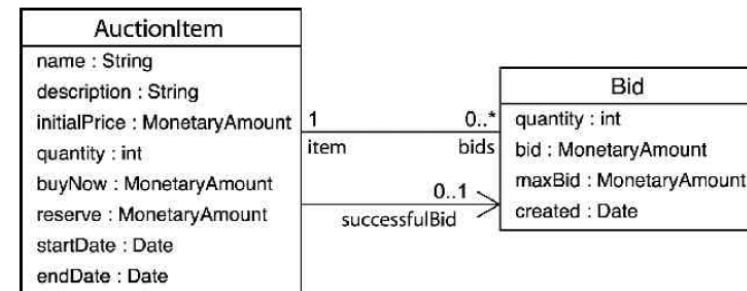
Mapping has to be defined  
by developer or generated.

# OR Mapping Definition

```
<class name="AuctionItem" table="AUCTION_ITEM">
  <id name="id" column="ITEM_ID">
    <generator class="native"/>
  </id>

  <property name="description" column="DESCR"/>
  <one-to-one name="successfulBid"
    column="SUCCESSFUL_BID_ID"/>

  <set name="bids" cascade="all" lazy="true">
    <key column="ITEM_ID"/>
    <one-to-many class="Bid"/>
  </set>
</class>
```



# OR-Mapping Hibernate example

Expl.: Retrieve an AuctionItem and change the description:

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

AuctionItem item =
    (AuctionItem) session.get(AuctionItem.class, itemId);

item.setDescription(newDescription);

tx.commit();
session.close();
```

# OR-Mapping Hibernate example

Retrieve an AuctionItem and change the description,  
as **detached object**:

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
AuctionItem item =
(AuctionItem) session.get(AuctionItem.class, itemId);
tx.commit();
session.close();
```

```
item.setDescription(newDescription);
```

```
Session session2 = sessionFactory.openSession();
Transaction tx = session2.beginTransaction();
session2.update(item);
tx.commit();
session2.close();
```

reattach!  
Danger of lost update?



# OR-Mapping Hibernate Example

**Retrieve an AuctionItem and create a new persistent Bid:**

```
Bid bid = new Bid()
bid.setAmount(bidAmount);
Session session=sessionFactory.openSession();
Transaction tx=session.beginTransaction();

AuctionItem item =
    (AuctionItem)session.get(AuctionItem.class, itemId);

bid.setItem(item);
item.getBids().add(bid);

tx.commit();
session.close();
```

No managed associations!

# Hibernate Query Options

- Hibernate Query Language (HQL)  
object-oriented dialect of ANSI SQL
- Native SQL queries  
direct passthrough with automatic mapping  
named SQL queries in metadata

Similar language concept in .NET: LINQ  
a language for querying data sources  
(Database/SQL, XML files and more)  
integrated in C#, VB

# Hibernate Query Language (HQL)

- Make SQL “object-oriented”
  - classes and properties instead of tables and columns
  - supports polymorphism
  - automatic association joining
  - *much* less verbose than SQL
- Full support for relational operations
  - inner/outer/full joins, cartesian product
  - projection, ordering, aggregation and grouping
  - subqueries and SQL functions

# OR-Mapping: Hibernate HQL

## Most simple HQL query!

```
from AuctionItem;
```

*i.e.* get all the **AuctionItems**:

```
List allAuctions =  
session.createQuery("from AuctionItem").list();
```

# OR-Mapping Hibernate example

```
select item
from AuctionItem item
join item.bids as bid
where item.description like "Hibernate%"
and bid.amount > 100
```

*i.e.* get all the **AuctionItems** with a **Bid** worth more than 100 and an item description that starts with "Hibernate".

# Hibernate Query: Criteria / HQL

```
List auctionItems =
session.createCriteria(AuctionItem.class).setFetchMode("bids
", FetchMode.EAGER)
    .add( Expression.like("description", desc) )
    .createCriteria("successfulBid")
    .add( Expression.gt("amount", minAmount) ).list();
```

## Equivalent HQL:

```
from AuctionItem item
left join fetch item.bids
where item.description like :description
and item.successfulbid.amount > :minAmount
```

# HQL: Passing Parameters

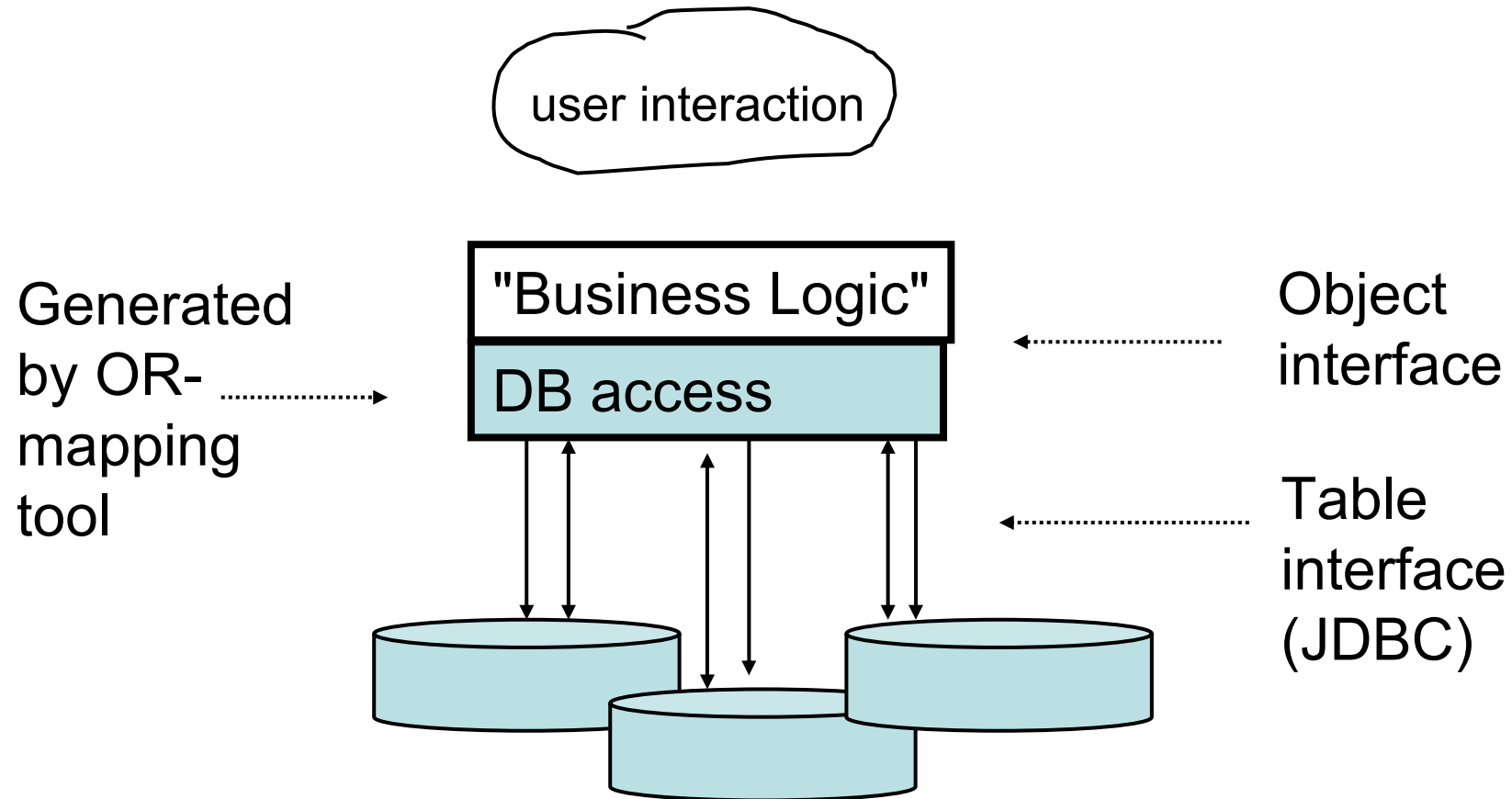
```
Query q = s.createQuery(  
    "from foo in class Foo  
    where foo.name=:name  
    and foo.size=:size");
```

```
q.setProperties(fooBean);
```

```
// fooBean has getName() and getSize() List foos = q.list();
```

# 9.5 Components and OR-Mapping

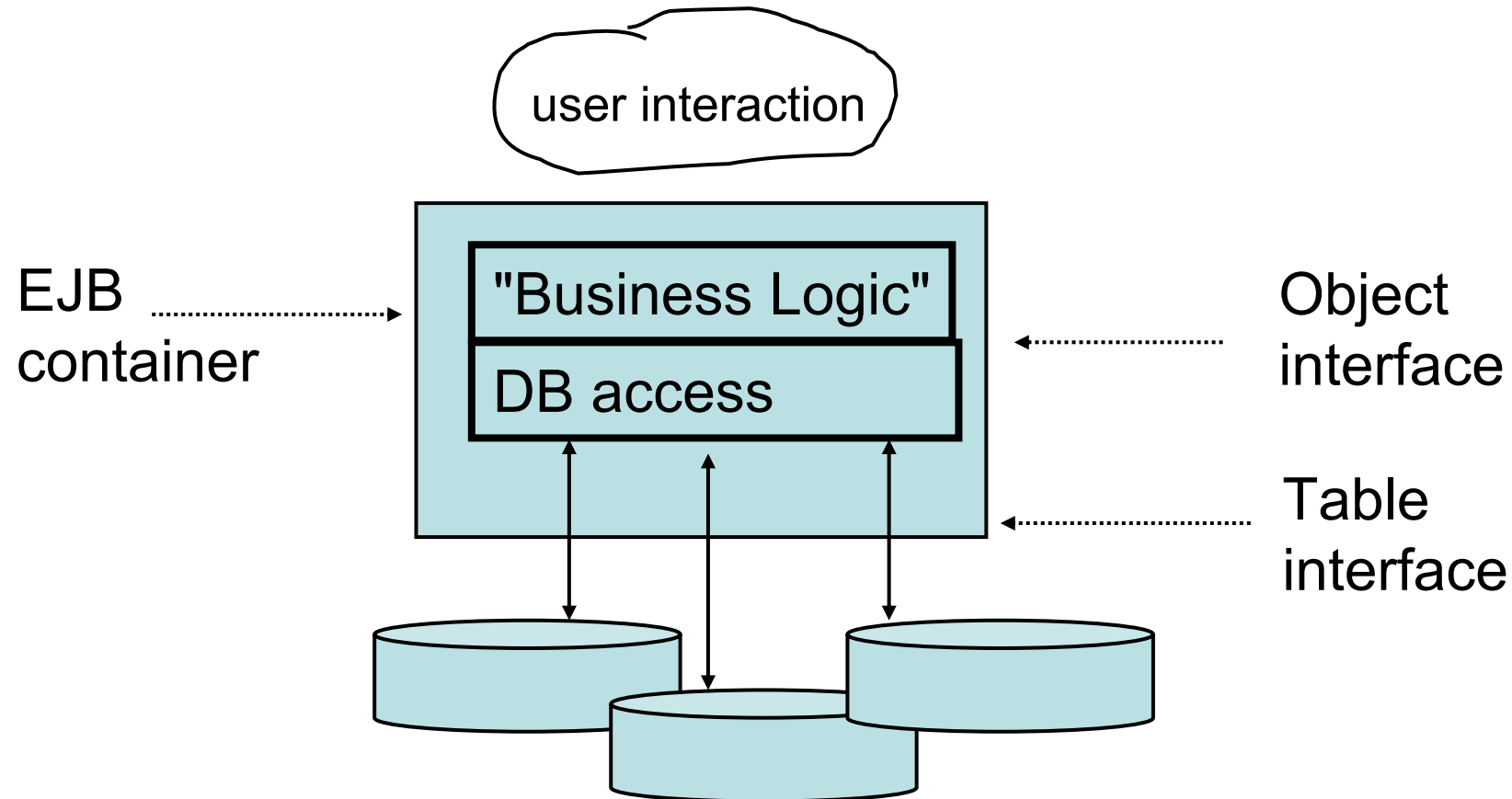
Persistent Architecture : Object-Relational mapping





# Components and OR-Mapping

## Component persistence Architecture



EJB: Enterprise Java Beans, component model for Business related software

# Components: EJB

## Motivation **EJB should relieve application from programmer technical issues**

- multi-user (concurrency management) and distributed transaction management
- security
- resource management and component lifecycle (threads, database connections, etc.)
- remote accessibility and component location transparency
- persistence, caching
- tools for bean installation and deploying
- portability: developing business beans without knowing the particular DB infrastructure
- **most important: scalability**

## Disadvantage

- not easy to comprehend
- not easy to understand side effects  
e.g. concurrency control in DB or in EJB container
  - Container: compromises durability ("changed data on disk?")
  - DB: heavy traffic between EJB cache and DB cache
- Depend on vendor implementations  
(EJB used to be underspecified)
- Performance: heavy penalty depending on application

Details in courses on SE (?) and  
(Distributed) Transaction Processing

# EJB performance

## Comparison of JDBC and EJB architectures using TPC-W\*

TPC-W

represents Web application  
(Business model: online book shop)

- Browser (Client)
- Web-Server
- Application-Server
- database

Measurement on DB size:

- books: 9.000, Authors: 2.500
- Customers: 5.000, Addresses 9.000,

Transactions defined by TPC-W (search, buy, ...)

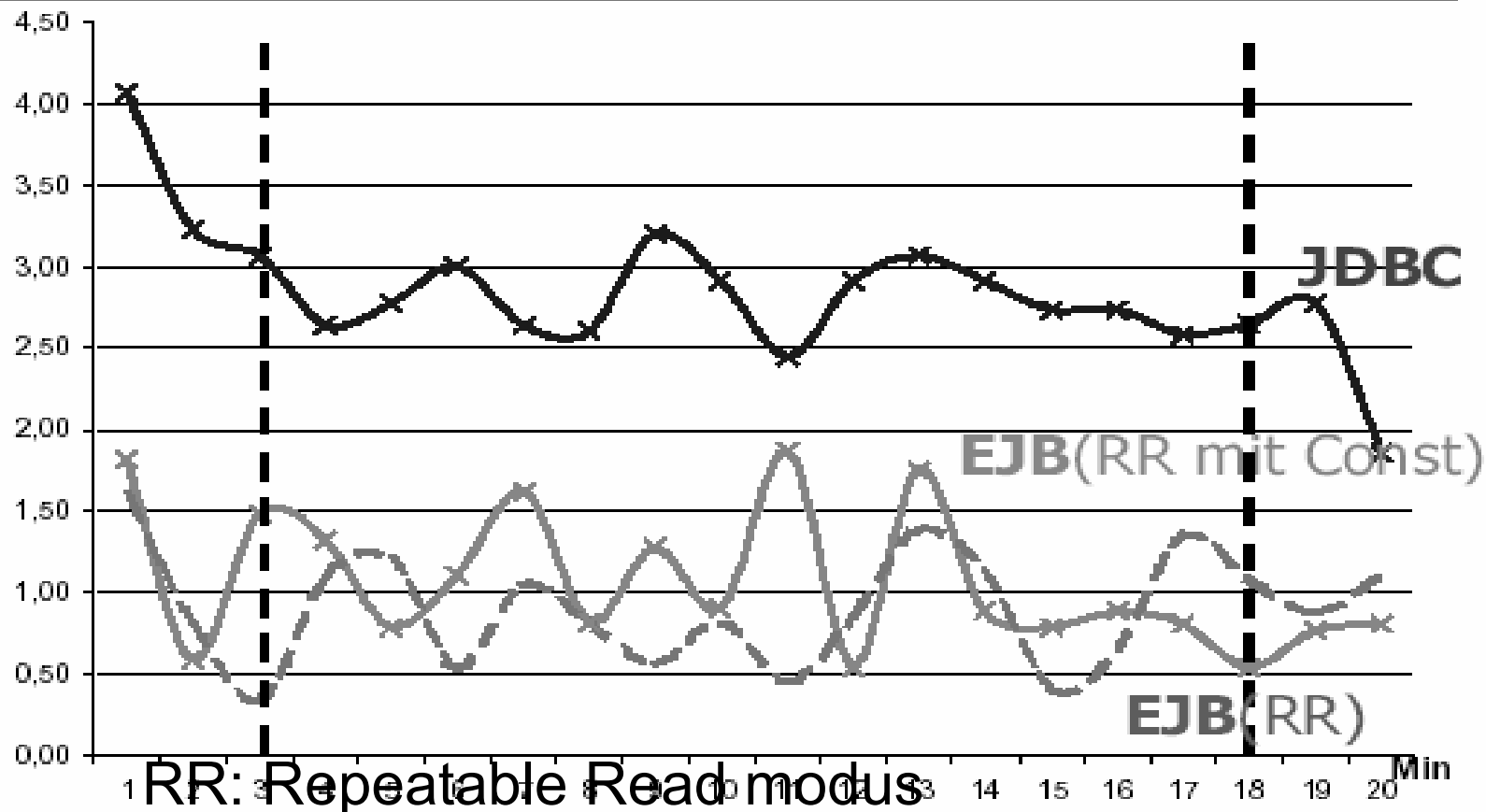
\* Study by Zink / Hettel,  
Hamburg,  
AK Frameworks2000

EJB using "container  
managed persistence", i.e.  
automatic mapping between  
DB objects and EJBs

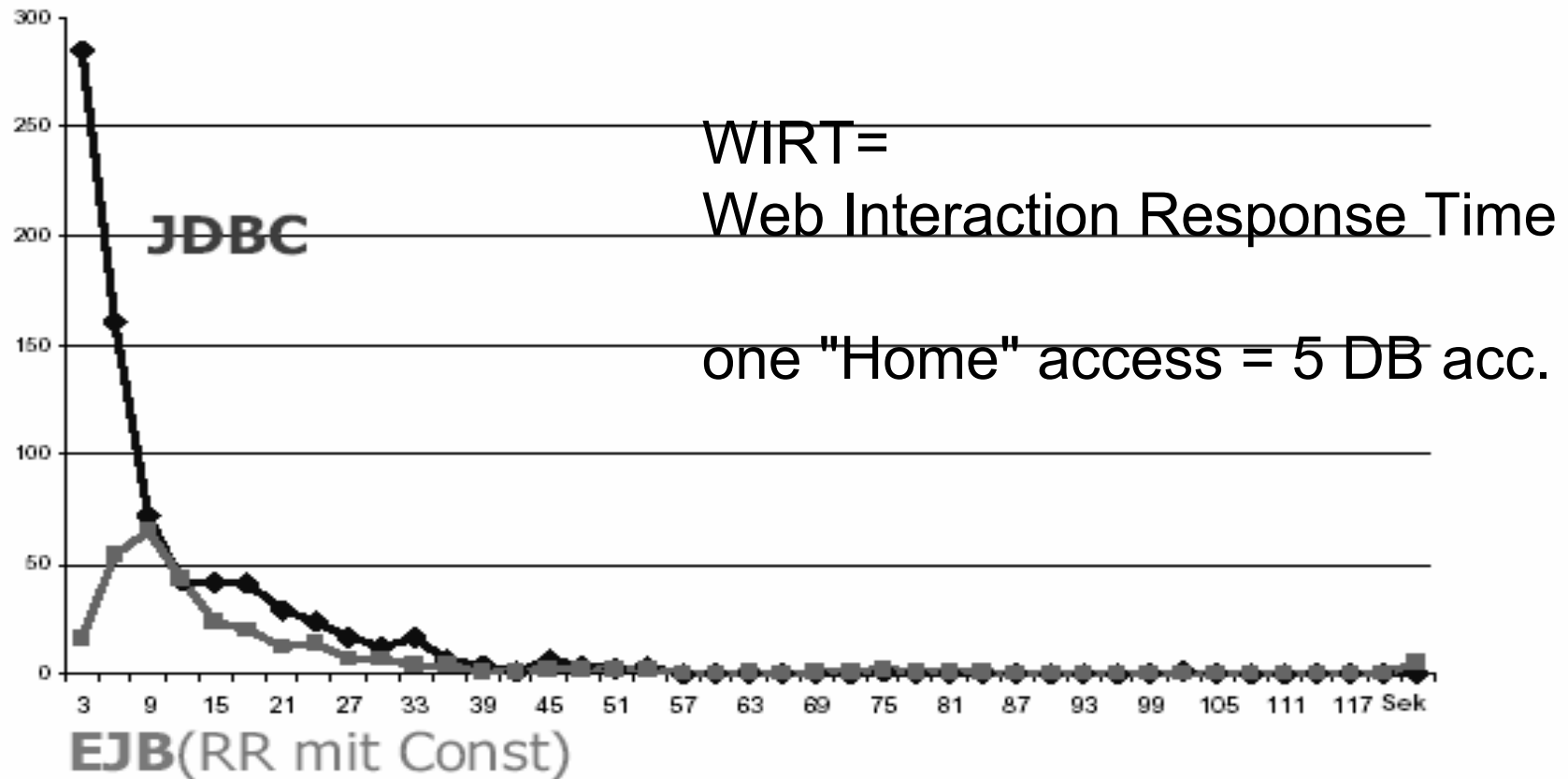
# JDBC / EJB performance comparison

ATTENTION: pretty OLD study, but tendency is correct.

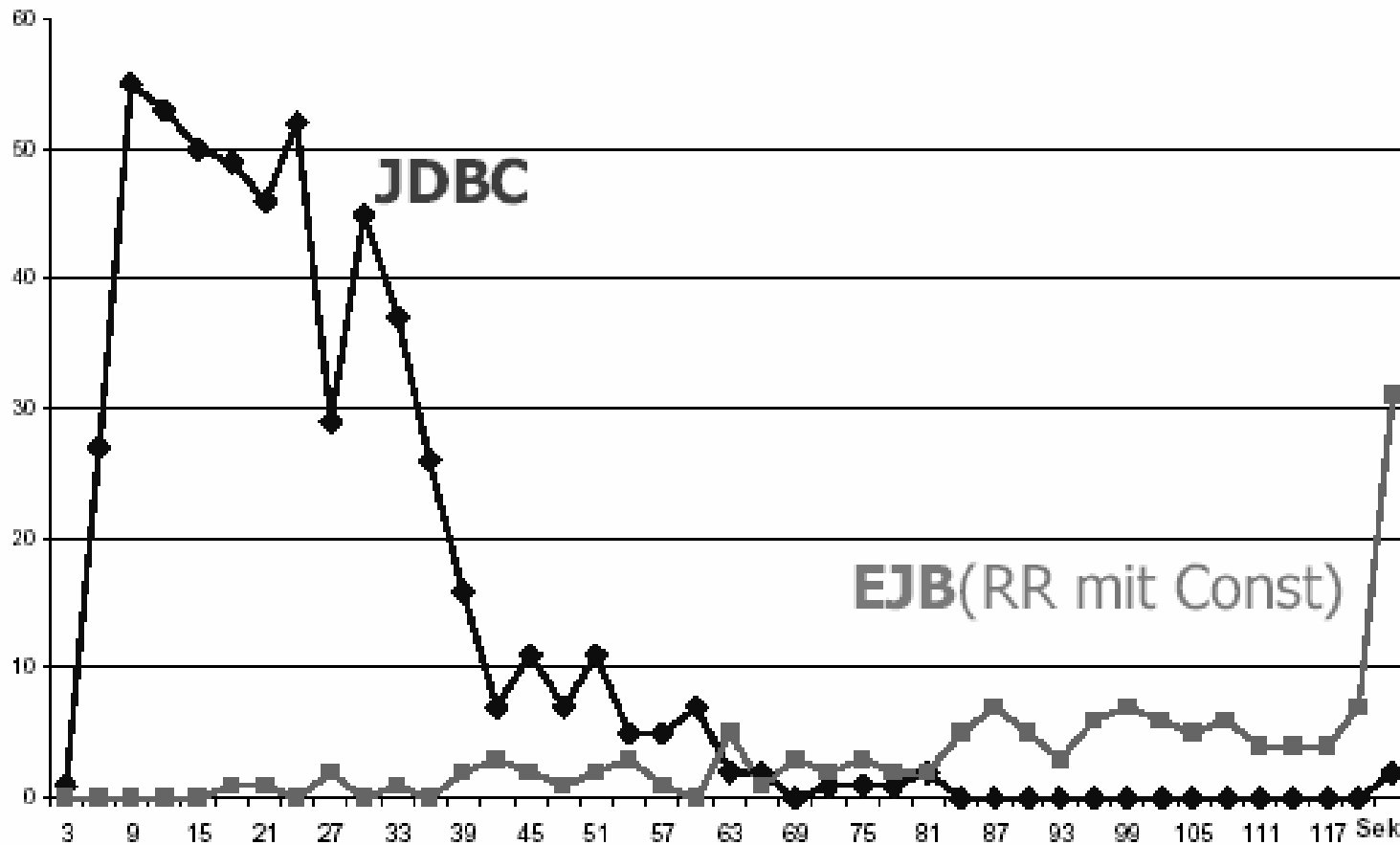
## Web Interactions per Second (WIPS)



## EJB vs. JDBC: WIRT-Verteilung der Seite "Home"

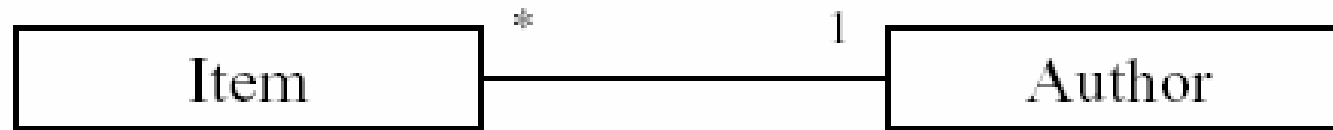


## EJB vs. JDBC: WIRT der Seite „SearchResult“



# JDBC / EJB performance comparison

Search operation: up to 50 associations



JDBC:

Database join

EJB: up to 50 `findByPrimaryKey()` - calls

EJB solution cannot perform better than JDBC



# JDBC / EJB performance comparison

Investigation shows tendency, but should not be overestimated:

EJB 2.0 / 3.0 have been improved considerably

Application servers which manage the containers, have been improved

"Bean managed persistence" may be tuned

Albert Einstein:

Not everything which can be measured is important,  
not everything which is important can be measured

# Summary

- Access of Database from **application program** more important than interactive SQL
- **cursor** : iterator through result sets explicit or implicit
- **Dynamic versus static** (compiled) SQL
- **Stored procedures**: important tool
- **JDBC** : similar to "call level interface (CLI)
- **SQLJ** : "Embedded" version, not as important as JDBC (?)
- Persistence frameworks useful in simple applications, questionable in large high performance systems.
- **Transaction = unit of work**: very important concept in multi user environments (remember: acid)
- **Isolation level**: lower levels acceptable in many cases, "serializability" prevents any harm due to read / write conflicts (details to come)