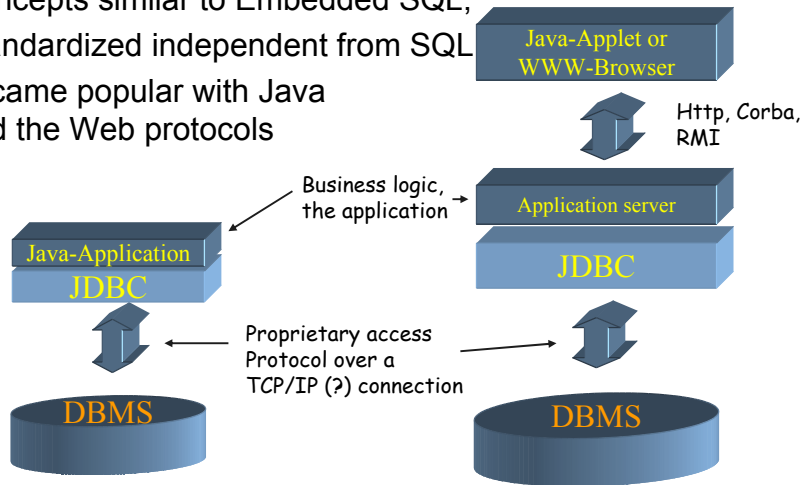


12.4 SQL and Java

- Concepts similar to Embedded SQL,
- Standardized independent from SQL
- became popular with Java and the Web protocols



HS / DBS05-15-ProgLan 43

SQL & Java JDBC und SQLJ

JDBC

- Java call-level interface (API) for SQL DBS
- DB vendor independent
- Supports static and dynamic SQL
- Implemented by nearly all DB vendors

SQLJ

- Is essentially embedded SQL for Java (instead of C)
- Compiles to JDBC method call
- Defined and implemented by major DBS companies (Oracle in particular)

HS / DBS05-15-ProgLan 44

12.4.1 JDBC

Preparation

1. `import java.sql.*;`
2. Load a driver (included in java.sql.*), many vendor products
`Class.forName("oracle.jdbc.driver.OracleDriver");`
url is a variable holding the JDBC-Driver and host information
3. Set up the connection to one or more database(s)
`Connection con = DriverManager.getConnection("jdbc:oracle:thin:@sunset.inf.fu-berlin.de:1521:hstarent", username, password);`
Several connections at a time may be used
4. Create a statement object
`Statement stmt = con.createStatement();`
something like a channel, through which queries are sent to the DB
Note: this is NOT a statement, but a "statement channel".

HS / DBS05-15-ProgLang 45

SQL & Java JDBC

Processing

5. Send SQL strings to be processed, results in a ResultSet object, which is processed with a kind of "hidden cursor"

```
ResultSet rs = stmt.executeQuery("SELECT * FROM  
Movies WHERE pricePD > 2" );
```

6. Process the results one after the other

```
while (rs.next())  
{ // Loop through each column, getting the  
  // column data and displaying  
  for (i = 1; i <= numCols; i++)  
  {  
    if (i > 1) System.out.print(",");  
    System.out.print(rs.getString(i));  
  }  
}
```

Variable binding
by position

HS / DBS05-15-ProgLang 46

JDBC: variable binding, result set iterator

Variable binding and result set

```
java.sql.Statement stmt = con.createStatement();
ResultSet r = stmt.executeQuery("SELECT a, b, c
                                FROM Table1");

while (r.next())
{
    // print the values for the current row.
    int i = r.getInt("a");
    String s = r.getString("b");
    float f = r.getFloat("c");
    System.out.println("ROW = " + i + " " + s + " " + f);
}
```

Compare
variable binding
by position

No explicit cursor, but iterator using methods

```
boolean next() , void close(),
<JavaType> get<JavaType> (),
boolean wasNull()
```

to iterate through the result.

... and many more in JDBC 2.0

HS / DBS05-15-ProgLang 47

SQL and Java Exceptions

```
try {
    // Code that could generate an exception goes here.
    // If an exception is generated, the catch block below
    // will print out information about it. }
catch(SQLException ex) {
    System.out.println("\n--- SQLException caught ---\n");
    while (ex != null) { System.out.println("Message: " +
        ex.getMessage ()); System.out.println("SQLState: " +
        ex.getSQLState ()); System.out.println("ErrorCode: " +
        ex.getErrorCode ()); ex = ex.getNextException();
        System.out.println(""); }
    }
} — process all exceptions
```

```
--- SQLException caught ---
Message: There is already an object named 'Meier' in the database.
Severity 16,
State 1, Line 1
SQLState: 42501      -- Defined as X/Open standard
ErrorCode: 2714     -- Vendor specific
```

HS / DBS05-15-ProgLang 48

JDBC

Accessing columns

- The class `ResultSet` has methods for each type to access result data by position or by name

- By position:

```
String s = rs.getString(2); // the second attribute to be bound.
```

- By name:

```
String rs.getString("b");  
// get the value of the attribute b of the  
// row under the (implicit) cursor
```

- ▶ What about ...

- ▶ Input parameters for queries? (... where attr = :val)
- ▶ Positioned updates without an explicit cursor?
- ▶ Exceptions?
- ▶ transactional facilities ?

HS / DBS05-15-ProgLang 49

JDBC

Prepared statements

```
String mTitle; ....  
try {  
    java.sql.PreparedStatement prepStmt =  
        con.prepareStatement("SELECT count(*)  
            FROM M,T where M.title = ? and T.mId = M.mId);  
    prepStmt.setString(1, mTitle);  
    ResultSet r = prepStmt.executeQuery();  
    while (r.next())  
    {  
        int i = r.getInt(1);  
        // must get by position, no name available  
        System.out.println("Number of tapes for " +  
            movieTitle + " is: " + i)  
    }  
} catch(SQLException {...})
```

Subclass of statement

will be compiled,
? Indicates still missing value

Bind value to position (!)
in statement and execute

HS / DBS05-15-ProgLang 50

Prepared vs non-prepared

- Overhead for compiling SQL-statement basically constant
- Ratio of compile Time / processing time for queries important
- Simple queries: prepare when executed frequently – e.g. in a loop
- Complex queries in a loop: no performance gain
- Many more factors to be analyzed

HS / DBS05-15-ProgLang 51

Very simple performance comparison

System: Postgres local
Query: SELECT * FROM city WHERE name = 'X'
Time: 150

System: Postgres local
Prepared Query: SELECT * FROM city WHERE name = 'X'
Time: 60

System: Postgres local
Query:
SELECT DISTINCT A.Name, B.Name
FROM country A, country B, encompasses uA,
encompasses uB, geo_sea gMA, geo_sea gMB
WHERE A.Name < B.Name AND uA.continent =
'Europa' AND uB.continent = 'Europa'
AND uA.country = A.c_ID AND
uB.country = B.c_ID AND A.c_ID = gMA.c_ID AND
B.c_ID = gMB.c_ID AND gMA.sea = gMB.sea
Time: 44802

System: Postgres local
Prepared Query: SELECT DISTINCT
A.Name, B.Name FROM country A, ..
Time: 42549

all queries executed 100 x
time: msec

System: Ora9i local
Query: SELECT * FROM city WHERE ..
Time: 301

System: Ora9i local
Prepared Query: SELECT * FROM city ..
Time: 120

System: Ora9i local
Query: SELECT DISTINCT A.Name, B.Name
Time: 1692

System: Ora9i local
Prepared Query:
SELECT DISTINCT A.Name, B.Name FROM
Time: 1583

Ora: more effort into optimizing query

HS / DBS05-15-ProgLang 52

JDBC

Positioned update

- Remember:

```
EXEC SQL DECLARE myCurs CURSOR FOR
      SELECT m#, ppd FROM M WHERE ...
      FOR UPDATE ON ppd:
      ....
EXEC SQL FETCH myCurs INTO ...
EXEC SQL UPDATE M SET ppd = ppd + 1
      WHERE CURRENT OF myCurs /* delete in
                              a similar way
```

...

needs a cursor name, but there is no cursor name

- JDBC 1 allows to define a cursor, which may be used in subsequent update / delete statements:

```
public void setCursorName(String name) throws
SQLException
```

- Much more flexible (anonymous) cursor handling in JDBC 2, `setCursorName` not implemented in Oracle Driver

HS / DBS05-15-ProgLang 53

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();
```

HS / DBS05-15-ProgLang 54

SQL & Java JDBC

– Transactional properties of connections

JDBC constants:

```
TRANSACTION_NONE
TRANSACTION_READ_UNCOMMITTED
TRANSACTION_READ_COMMITTED
TRANSACTION_REPEATABLE_READ
TRANSACTION_SERIALIZABLE
```

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`

HS / DBS05-15-ProgLang 55

JDBC

Visibility of changes

- Does TA see it's own changes?
 if there is more than one result set in a TA?

```
ResultSet rs = stmt.executeQuery(
    "SELECT emp_no, salary FROM employees");
ResultSet badGuys = stmtx.executeQuery (
    "SELECT emp_no FROM employees where ....");
...int i = executeUpdate ("DELETE FROM employees");
rs.updateFloat("salary", 10000.0);
rs.updateRow();
```

Is the deleted row still visible??

Different ways to
perform an update

HS / DBS05-15-ProgLang 56

JDBC updates

- Not visible if result set is **scroll-insensitive**: no change from other result sets – even in the same TA – are visible
- Updates made in a result set *r* are visible by operations on *r*, but not necessarily deletes / inserts (!)
- If sensitive, it depends on connection isolation level

HS / DBS05-15-ProgLang 57

JDBC: calling stored procedures

- Stored procedures may be called directly
- ```
CallableStatement cs
= con.prepareStatement("{?=call rand(?, 100) }");
cs.setInt(2,1); // set arguments
cs.registerOutParameter(1, Types.INTEGER);
 // register type of output
cs.execute();
System.out.println(cs.getInt(1) + ", ");
 // execute and get result value, position 1
```
- If stored procedure causes updates:
- ```
int i = cs.executeUpdate();
```

HS / DBS05-15-ProgLang 58

JDBC and persistent objects

- Persistence of Java Objects

```
ResultSet rs = stmt.executeQuery(  
    "SELECT Employee FROM PERSONNEL");  
    rs.next();  
    Employee emp = (Employee)rs.getObject(1);
```

- Updating objects in the DB

```
emp.setSalary(emp.getSalary() * 1.5);  
PreparedStatement pstmt = con.prepareStatement(  
    "UPDATE PERSONNEL SET Employee = ?  
    WHERE Employee.no = 1001");  
pstmt.setObject(1, emp);  
pstmt.executeUpdate();
```

- Persistence frameworks: no SQL statements visible for application programmer.

Persistent object collections may be updated in

java terms `emp.setSalary (emp.getSalary*1,5)`

and updated in the DB "automatically".

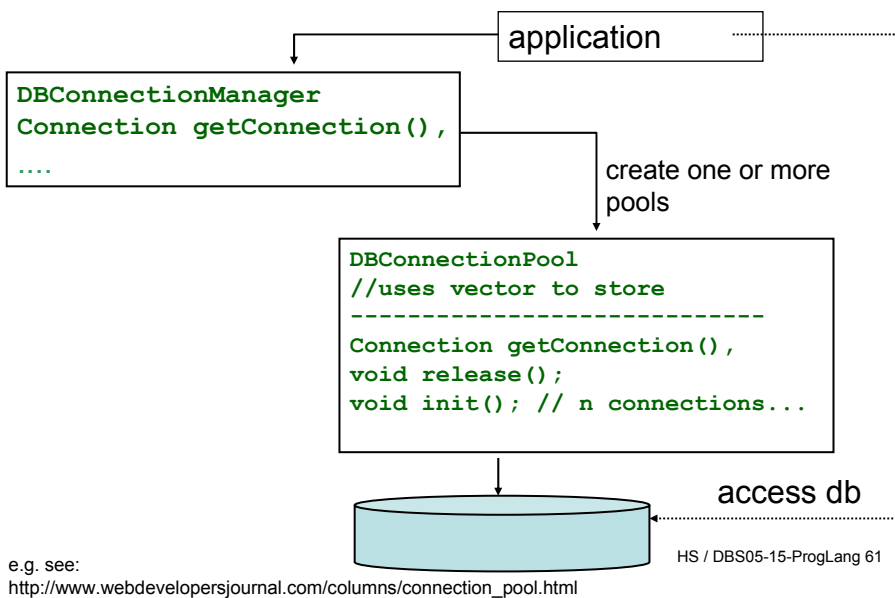
HS / DBS05-15-ProgLang 59

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"](#))

HS / DBS05-15-ProgLang 60

JDBC connection manager



JDBC: programming style

- Do not hard-code connection data but use property file

```
drivers=jdbc:postgresql jdbc:oracle:thin
logfile=D:\\user\\src\\java\\DBConnectionManager\\log.txt

ora.url=jdbc:oracle:thin:@localhost:1521:tarenths
ora.maxconn=2
pg.url=jdbc:postgresql://localhost:5432/geo
pg.user=demo
pg.password=demopw
```

different pools

`init()` method:

Create property object

```
Properties dbProp = new Properties();
```

and Read property file

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.

HS / DBS05-15-ProgLang 63

12.4.2 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

HS / DBS05-15-ProgLang 64

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();
```

HS / DBS05-15-ProgLang 65

SQLJ Example

```
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 };
```

HS / DBS05-15-ProgLang 66

```

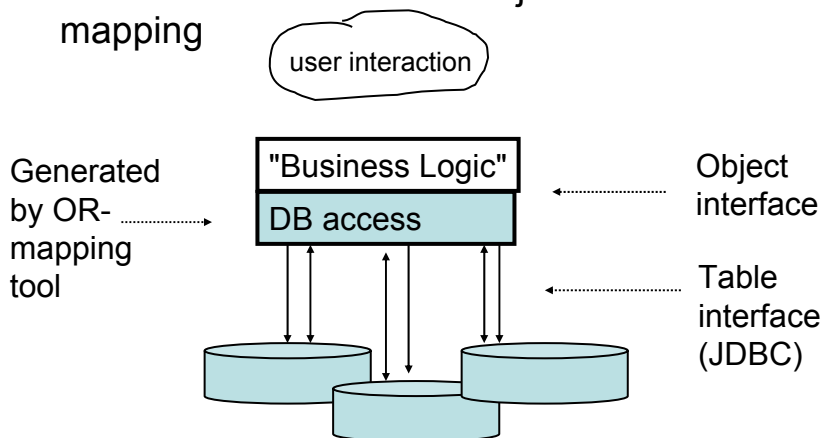
on_hand -= quantity;
if (on_hand < threshold) {
    String supplier;
    #sql { SELECT SUPPLIER INTO :supplier
          FROM PARTS_MASTER
          WHERE PART_ID = :part };
    inventory.orderMore(part, quantity, supplier);
}
if (on_hand < 0) {
    #sql { ROLLBACK };
    throw new OutOfStock();
} else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

```

HS / DBS05-15-ProgLang 67

12.5 Components and OR-Mapping

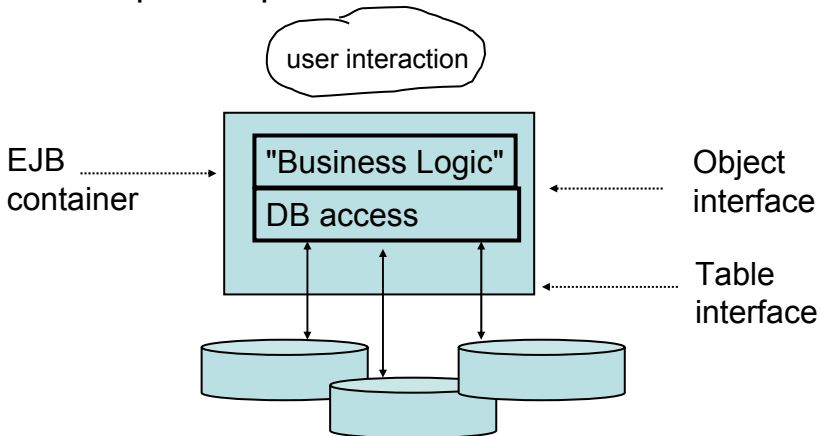
- Persistent Architecture : Object-Relational mapping



HS / DBS05-15-ProgLang 68

Components and OR-Mapping

- Component persistence Architecture



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

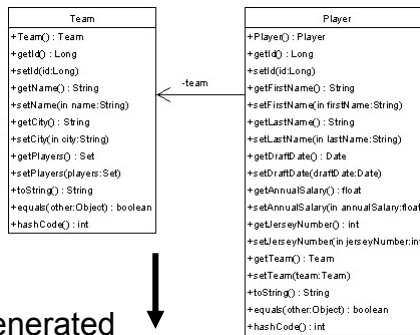
HS / DBS05-15-ProgLang 69

Motivation

- OR mapping
 - 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 (OBJ) and many more

OR mapping

From Classes
to tables,
from rows
to objects



mapping defined
by developer or generated

teams	
PK	team_id
	team_name
	city

players	
PK	player_id
	first_name
	last_name
	draft_date
	salary
	jersey_number
	team_id

hibernate_unique_key	
	next_hi

HS / DBS05-15-ProgLang 71

OR mapping

- Example continued
- Retrieval and update

```
// method: loading the Player's state Player
Session session = SessionFactory.openSession();
player = new Player();
session.load(player, playerId);
player.setAnnualSalary(4000000f);
session.saveOrUpdate(player);
```

SQL-like QL
allows to retrieve
by content

- Advantage:
 - No explicit result set processing
 - Object oriented framework
- Disadvantage
 - overhead compared to JDBC
 - no stored procedures
 - not a big conceptual difference

HS / DBS05-15-ProgLang 72

Components: EJB

- Motivation
 - should support 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, chaching
 - tools for bean installation and deploying
 - portability: developping business beans without knowing the particular DB infrastructure
- ⇒ Economic impact!

HS / DBS05-15-ProgLang 73

EJB

- Disadvantage
 - not easy to learn
 - not easy 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 specification is underspecified)
 - Performance: heavy penalty depending on application

How portability is achieved will not be discussed here -> course on J2EE development

HS / DBS05-15-ProgLang 74

EJB performance

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

* Study by Zink / Hettel, Hamburg, AK Frameworks2000

- TPC-W
 - represents Web application (Business model: online book shop)
 - Browser (Client)
 - Web-Server
 - Application-Server
 - database

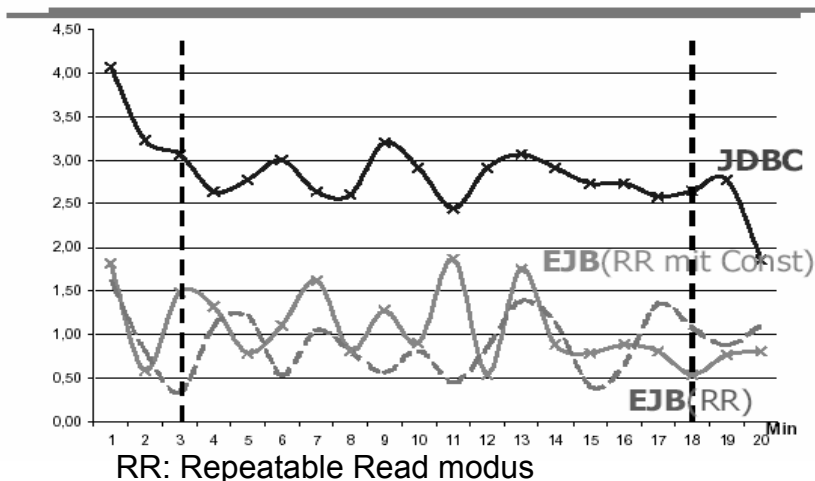
EJB using "container managed persistence", i.e. automatic mapping between DB objects and EJBs

Measurement on DB size:

- books: 10.000, Authors: 2.500
- Customers: 5.000, Addresses 10.000,
- Transactions defined by TPC-W (search, buy, ...)

JDBC / EJB performance comparison

Web Interactions per Second (WIPS)

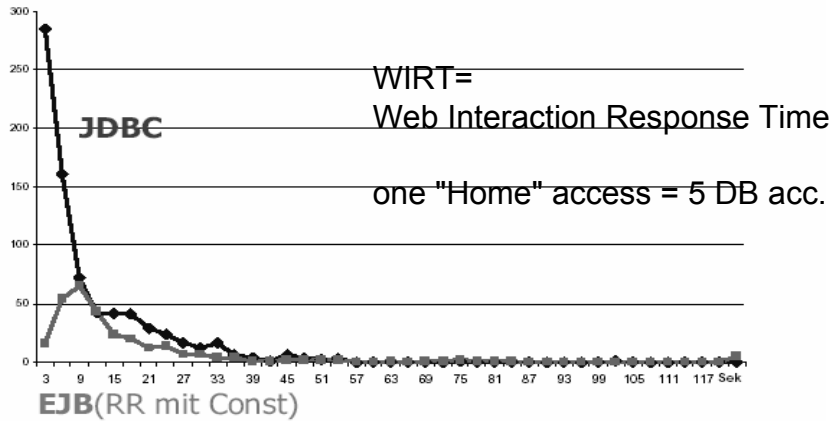


* source Zink / Hettel, 2000

HS / DBS05-15-ProgLang 76

JDBC / EJB performance comparison

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

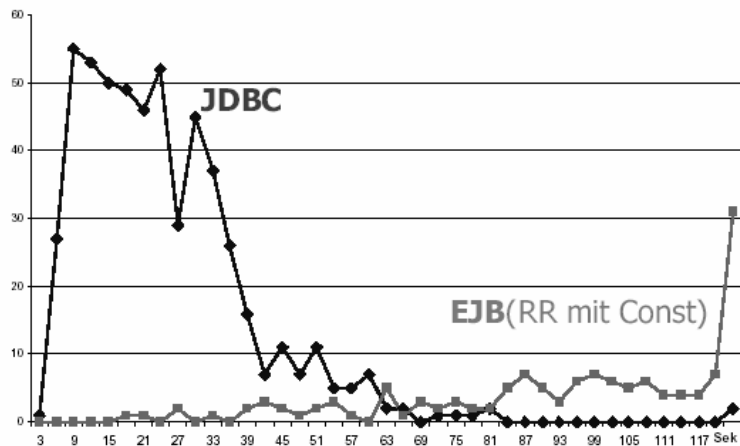


* source Zink / Hettel, 2000

HS / DBS05-15-ProgLang 77

JDBC / EJB performance comparison

EJB vs. JDBC: WIRT der Seite „SearchResult“

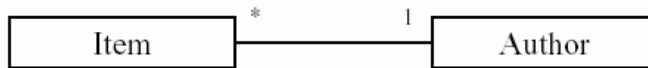


* source Zink / Hettel, 2000

HS / DBS05-15-ProgLang 77

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

HS / DBS05-15-ProgLang 79

JDBC / EJB performance comparison

- Investigation shows tendency, but should not be overestimated:
 - EJB 2.0 have been improved
 - 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

HS / DBS05-15-ProgLang 80

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
- 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
- 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.