Database Programming

With slides by Ph.Bonnet, J.Ulmann, L.Bouganim, M.Kifer, B.O Neill

Outline

- Host-Language and SQL
  - Problems
  - SLI vs. CLI vs. SP
- JDBC (Java Database Connectivity)
  - Call-Level Interface
  - java.sql
- Object-Relational Mapping

Application Program

- *Host language*: A conventional language (*e.g.*, C, Java) that supplies control structures, computational capabilities, interaction with physical devices
- *SQL*: supplies ability to interact with database.
- *Using the facilities of both*: the application program can act as an intermediary between the user at a terminal and the DBMS

SQL and Host Language

- SQL statements can be incorporated into an application program in three different ways:
  - ***Statement Level Interface*** (SLI): Application program is a mixture of host language statements and SQL statements and directives
  - ***Call Level Interface*** (CLI): Application program is written entirely in host language
    - SQL statements are values of string variables that are passed as arguments to host language (library) procedure
  - ***Stored Procedure*** (SP): Portion of application program written in specialized language and stored on database manager.

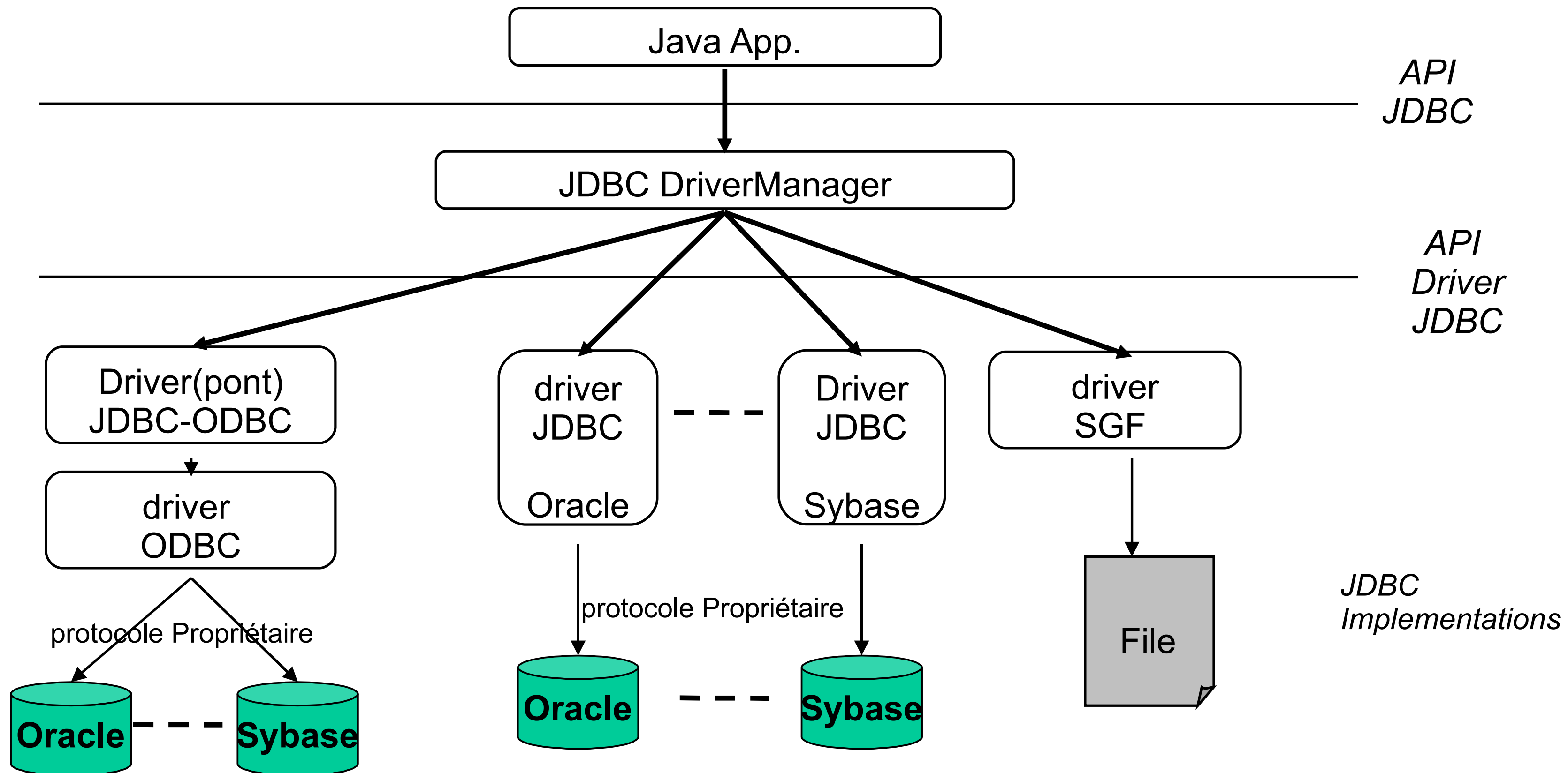
The "Impedance" Mismatch

- One cannot write a complete application in SQL, so SQL statements are embedded in a host language, like C or Java.
- **SQL**: Set-oriented, works with relations, uses high-level operations over them.
- **Host language**: Record-oriented, does not understand relations and high-level operations on them.
- **SQL**: Declarative.
- **Host language**: Procedural.
- Embedding SQL in a host language involves ugly adaptors (iterators) – a direct consequence of the above mismatch of properties between SQL and the host languages. It was dubbed *"impedance" mismatch*.

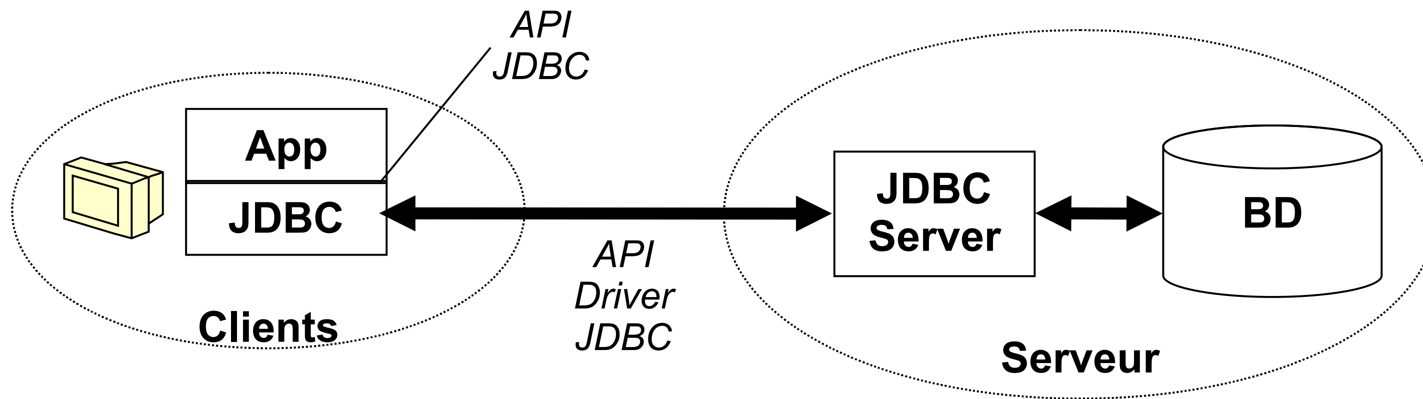
Outline

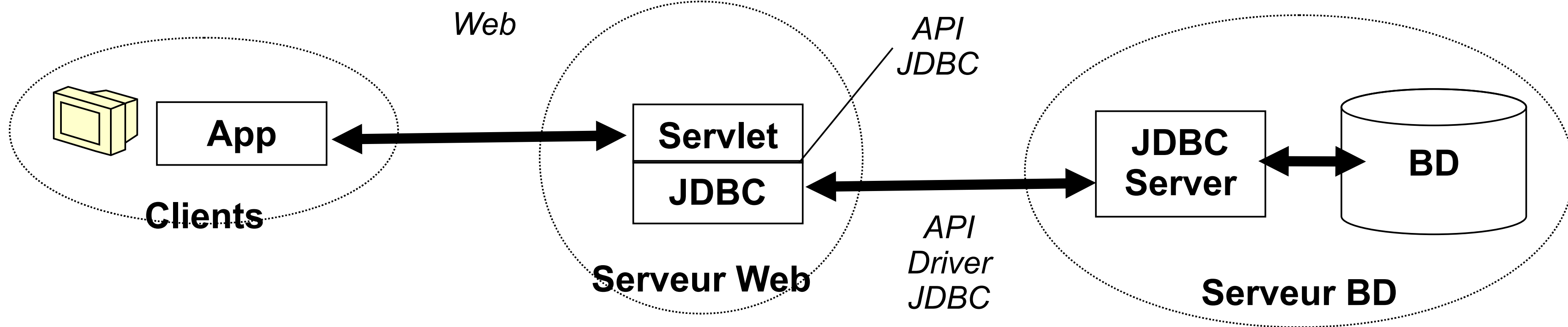
- Host-Language and SQL
  - Problems
  - SLI vs. CLI vs. SP
- JDBC (Java Database Connectivity)
  - Call-Level Interface
  - java.sql
- Object-Relational Mapping

JDBC Architecture

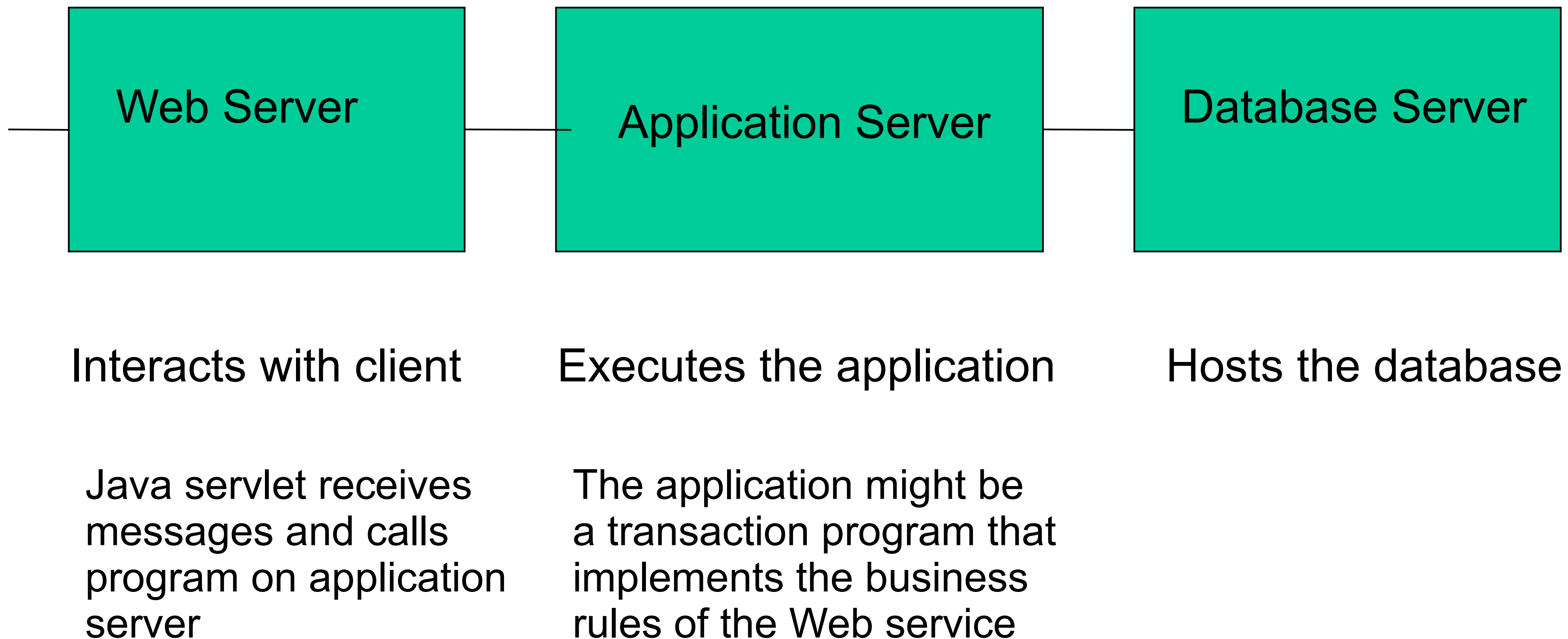


Distributed JDBC Driver





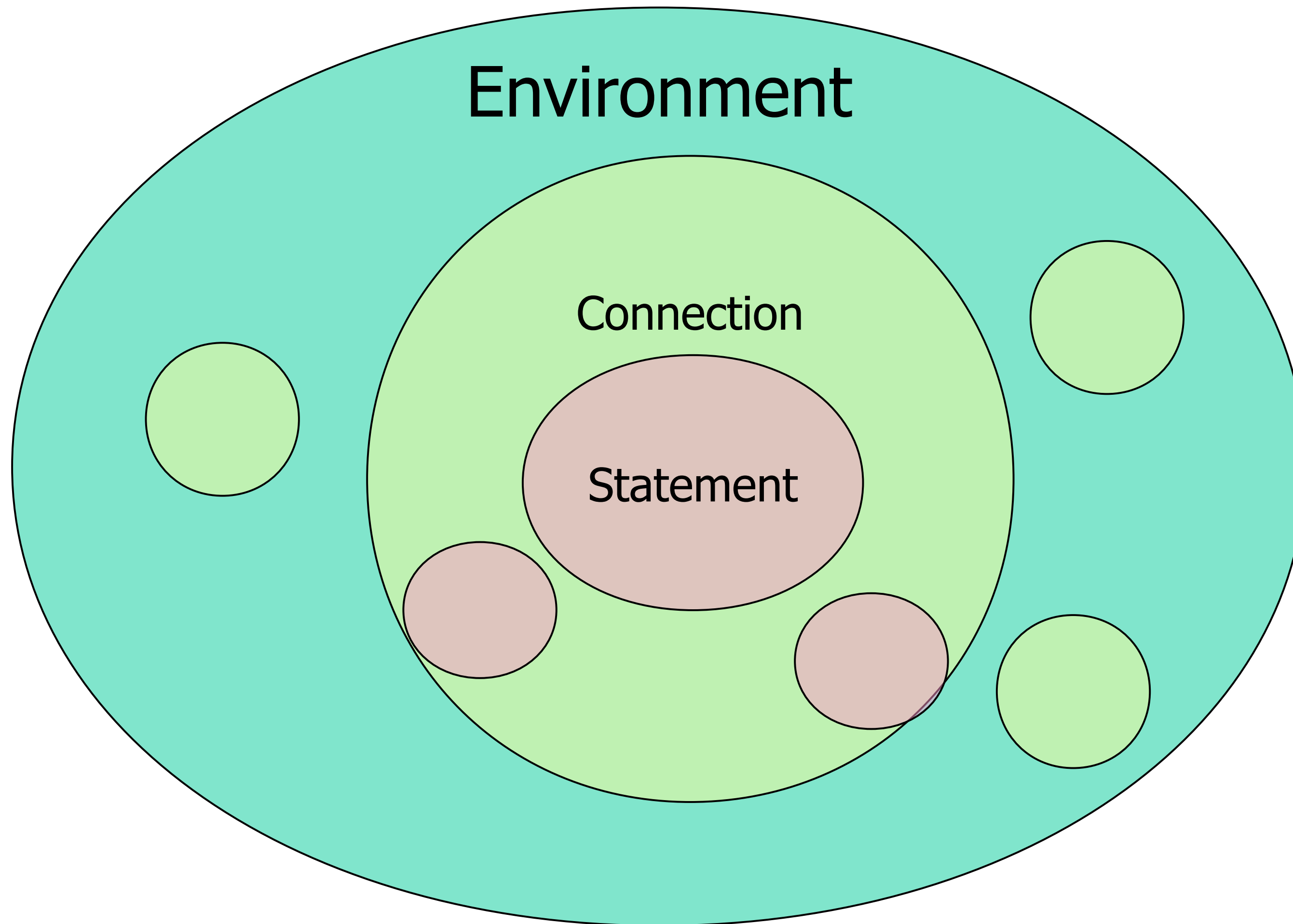
Architecture of a Web Application



JDBC Concepts

- The database is an *environment*.
- Database servers maintain some number of *connections*, so app servers can ask queries or perform modifications.
- The app server issues *statements* : queries and modifications, usually.

Diagram to Remember



Executing a Query

import java.sql.*; -- *import all classes in package java.sql*

Class.forName (*driver name*); // *static method of class Class*
// *loads specified driver*

Connection con = DriverManager.getConnection(*Url, Id, Passwd*);

- *Static method of class* DriverManager; *attempts to connect to DBMS*
- *If successful, creates a connection object,* con, *for managing the connection*

Statement stat = con.createStatement ();

- *Creates a statement object* stat

Static Features

```
String query = “SELECT   T.StudId  FROM  Transcript T” +
        “WHERE   T.CrsCode = ‘cse305’ ” +
        “AND  T.Semester = ‘S2000’ ”;
ResultSet  res = stat.executeQuery (query);
```

- *Creates a result set object,* res.
- *Prepares and executes the query.*
- *Stores the result set produced by execution in* res
- *The query string can be constructed at run time (as above).*
- *The input parameters are plugged into the query when*

Dynamic Features

```
String query = "SELECT T.StudId FROM Transcript T" +
        "WHERE T.CrsCode = ? AND T.Semester = ?";
```

```
PreparedStatement ps = con.prepareStatement ( query );
```

- *Prepares the statement*
- *Creates a prepared statement object,* ps, *containing the prepared statement*
- Placeholders *(?) mark positions of* in *parameters; special API is provided to plug the actual values in positions indicated by the ?'s*

Dynamic Features

```
String  crs_code, semester;
………
ps.setString(1, crs_code);     // set value of first in
parameter
ps.setString(2, semester);     // set value of second in
parameter

ResultSet  res = ps.executeQuery ( );
```

- *Creates a result set object,* res
- *Executes the query*
- *Stores the result set produced by execution in* res

```
while  ( res.next ( ) )  {                        // advance the
cursor
      j = res.getInt ("StudId");          // fetch output int-value
      …process output value…
```

Dynamic Features

```
ResultSet rs =  stmt.executeQuery(”select * from R”);
ResultSetMetaData rsmd = rs.getMetaData();

int numbOfColumns = rsmd.getColumnCount();
String columnName = rsmd.getColumnName(1);
String TypeName = rsmd.getColumnTypeName(1);
```

Result Sets and Cursors

- Three types of result sets in JDBC:
  - *Forward-only*: not scrollable
  - *Scroll-insensitive*: scrollable; changes made to underlying tables after the creation of the result set are not visible through that result set
  - *Scroll-sensitive*: scrollable; updates and deletes made to tuples in the underlying tables after the creation of the result set are visible through the set

Result Set

```
Statement stat = con.createStatement (
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE  );
```

- Any result set type can be declared *read-only* or *updatable* – CONCUR_UPDATABLE (assuming SQL query satisfies the conditions for updatable views)
- *Updatable*: Current row of an updatable result set can be changed or deleted, or a new row can be inserted. Any such change causes changes to the underlying database table

```
res.updateString ("Name", "John" );    // change the attribute "Name" of
                                        // current row in the row buffer.
res.updateRow ( );   // install changes to the current row buffer
```

Handling Exceptions

```
try {
    ...Java/JDBC code...
}  catch ( SQLException  ex ) {
    …exception handling code...
}
```

- try/catch is the basic structure within which an SQL statement should be embedded
- If an exception is thrown, an exception object, *ex*, is created and the catch clause is executed
- The exception object has methods to print an error message, return SQLSTATE, etc.

Transactions in JDBC

- Default for a connection is
  - Transaction boundaries
    - *Autocommit mode*: each SQL statement is a transaction.
    - To group several statements into a transaction use con.setAutoCommit (false)
  - Isolation
    - default isolation level of the underlying DBMS
    - To change isolation level use

con.setTransactionIsolationLevel (TRANSACTION_SERIALIZABLE)

- With autocommit off:
  - transaction is committed using con.commit().
  - next transaction is automatically initiated (chaining)
- Transactions on each connection committed separately

Outline

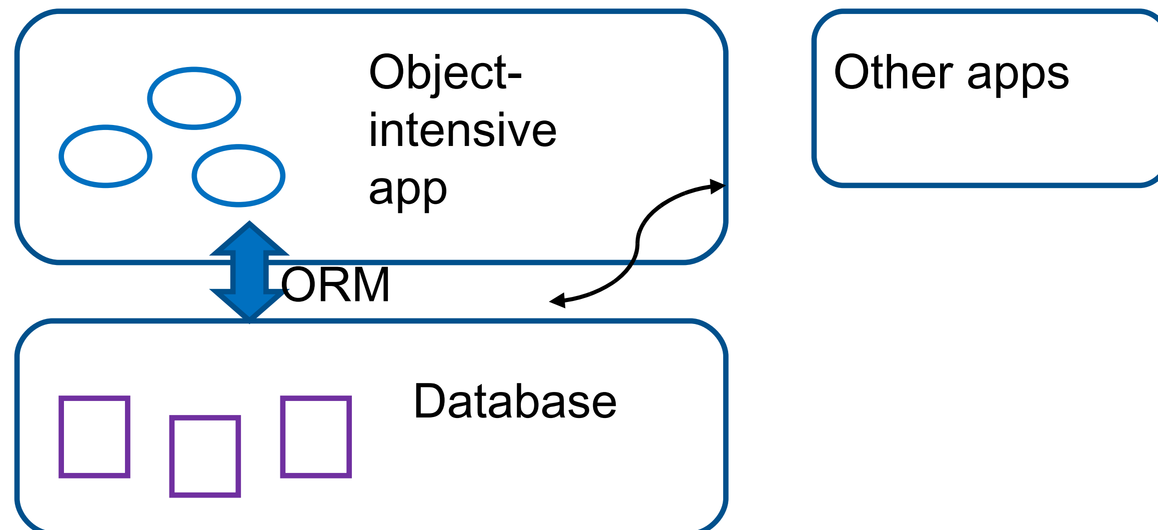
- Host-Language and SQL
  - Problems
  - SLI vs. CLI vs. SP
- JDBC (Java Database Connectivity)
  - Call-Level Interface
  - java.sql
- Object-Relational Mapping

Programming with Objects

- Programmers love objects
- Objects are normally ephemeral
- Programming languages provide object persistence, but it is fragile
- Databases provide robust data persistence.
- So… need way to persist object data in the database
- Or think bigger: use data model across object-DB boundaries.

Object-Relational Mapping (ORM)

- A software system that shuttles data back and forth between database rows and objects
- Appears as a normal database user to the database
- Can share the database and tables with other apps



Object-Relational Mapping

- Has a history, but widely adopted only since open-source Hibernate project started, 2001-2002.
- The Hibernate project was founded and led by Gavin King, a Java/J2EE software developer..
- Microsoft has adopted a comparable approach with EDM, Entity Data Model and its Entity Framework.
- Both Hibernate and EDM support (or will support) multiple databases: Oracle, DB2, SQL Server, …

Java Persistence Architecture

- JPA is part of current JEE (previously J2EE), Sun's Java Enterprise Edition
- JPA is a standardized version of the **Hibernate** architecture
- EJB 3 (current Entity Java Beans) uses JPA for EJB persistence, i.e., persistence of "managed" objects
- JPA and EJB 3 are now available in major application servers: Oracle TopLink 11g, OpenJPA for WebSphere and WebLogic, Hibernate JPA for JBoss
- JPA can be used outside EJB 3 for persistence of ordinary objects, as Hibernate is used in this talk

Non trivial mapping

- In the simplest case, a program object of class A has fields x, y, z and a table B has columns x, y, z
  - Each instance of A has a row in B and vice versa, via ORM
  - Are we done?
  - If x is a unique id, and x, y, and z are simple types, yes.
  - --Or some unique id in (x, y, z), possibly composite
- If no unique id in (x, y, z), the object still has its innate identity, but corresponding rows involve duplicate rows, against relational model rules
- So in practice, we add a unique id in this case:
- Class A1 has id, x, y, z and table B1 has id, x, y, z

Persistent Objects & Identity

- A "persistent object" is an **ordinary program object tied via ORM to database data for its long-term state**
- The program objects come and go as needed
- Don't confuse this with language-provided persistence (Java/C#), a less robust mechanism
- Persistent objects have **field-materialized identity**
- It makes sense—Programming language object identity depends on memory addresses, a short-lived phenomenon
- So long-lived objects (could be years…) have to be identified by their fields, it's not the database's fault

Persistent Objects need tracking

- We want only one copy of each unique object in use by an app, a basic idea of OO programming
- **Each persistent object has a unique id**
- We can no longer depend on object location in memory to ensure non-duplication
- So we have a harder problem than Programming Languages —need an active agent tracking objects
- This agent is part of ORM's runtime system
- The ORM uses hashing to keep track of ids, detect duplicates

ORM Entities

- Like E/R entities, ORM entities model collections of real-world objects of interest to the app
- Entities have properties/attributes of database datatypes
- Entities participate in relationships—see soon (but relationships are not "first class" objects, have no attributes)
- Entities have unique ids consisting of one or more properties
- Entity instances (AKA entities) are persistent objects of persistent classes
- Entity instances correspond to database rows of matching unique id

Creating Unique IDs

- ID creation can't be done with standard SQL insert, which needs predetermined values for all columns
- Every production database has a SQL extension to do this
  - Oracle's sequences
  - SQL Server's auto-increment datatype
  - …
- The ORM system coordinates with the database to assign the id, in effect standardizing an extension of SQL
- Keys obtained this way have no meaning, are called "surrogate keys"
- Natural keys can be used instead if they are available.
  - Bad idea though...

Tutorial Example

```
package org.hibernate.tutorial.domain;
import java.util.Date;

public class Event {
    private Long id;
    private String title;
    private Date date;
    public Event() {}
    public Long getId() {
        return id;
    }
    private void setId(Long id) {
        this.id = id;
    }
    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
}
```

Tutorial Example

```
<hibernate-mapping package="org.hibernate.tutorial.domain">

    <class name="Event" table="EVENTS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
        <property name="date" type="timestamp" column="EVENT_DATE"/>
        <property name="title"/>
    </class>

</hibernate-mapping>
```

Tutorial Example

```
package org.hibernate.tutorial.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
    private static final SessionFactory sessionFactory = buildSessionFactory();
    private static SessionFactory buildSessionFactory() {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            return new Configuration().configure().buildSessionFactory();
        }
        catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Tutorial Example

```
package org.hibernate.tutorial;
import org.hibernate.Session;
import java.util.*;
import org.hibernate.tutorial.domain.Event;
import org.hibernate.tutorial.util.HibernateUtil;
public class EventManager {
    public static void main(String[] args) {
        EventManager mgr = new EventManager();
        if (args[0].equals("store")) {
            mgr.createAndStoreEvent("My Event", new Date());
        }
        HibernateUtil.getSessionFactory().close();
    }
    private void createAndStoreEvent(String title, Date theDate) {
        Session session = HibernateUtil.getSessionFactory().getCurrentSession();
        session.beginTransaction();
        Event theEvent = new Event();
        theEvent.setTitle(title);
        theEvent.setDate(theDate);
        session.save(theEvent);
        session.getTransaction().commit();
    }
}
```

Tutorial Example

```
if (args[0].equals("store")) {
            mgr.createAndStoreEvent("My Event", new Date());
        }
        else if (args[0].equals("list")) {
            List events = mgr.listEvents();
            for (int i = 0; i < events.size(); i++) {
                Event theEvent = (Event) events.get(i);
                System.out.println(
                        "Event: " + theEvent.getTitle() + " Time: "
+ theEvent.getDate()
                );
            }
        }
```

Object-Orientation Considered Harmful

- authorized(user, type)
- doc(id, type, date)
- What are the document instances a user can see?
- SQL:

select doc.id, doc.date
from authorized, doc
where doc.type =
authorized.type
and authorized.user = <input>

- If each document is encapsulated in an object, the risk is the following:
  - Find types t authorized for user *input*

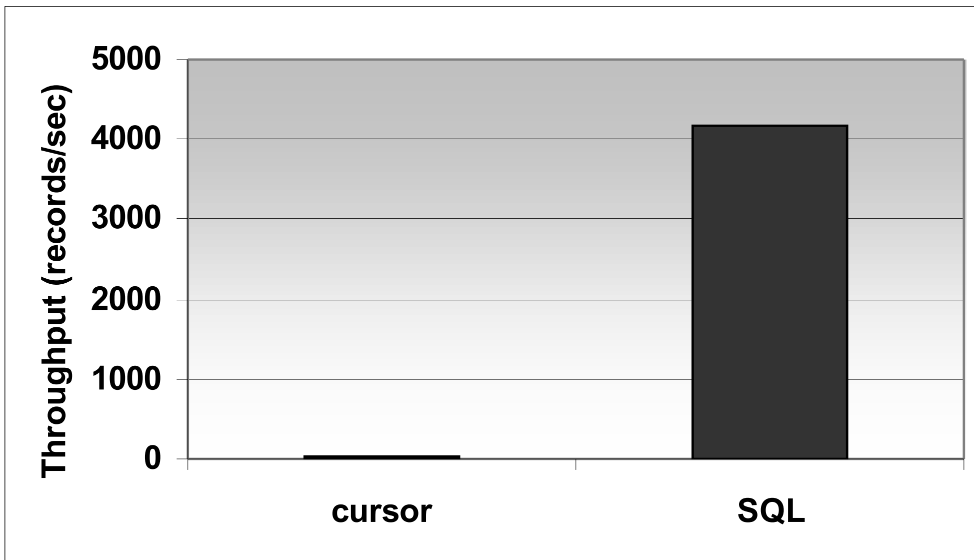
select doc.type as t
from authorized
where user = <input>

  - For each type *t* issue the query

select id, date
from doc
where type = <t>;

  - The join is executed in the application and not in the DB!

Cursors



- Query fetches 200000 56 bytes records
- Response time is a few seconds with a SQL query and more than an hour iterating over a cursor.

Hibernate QL

- SQL-like Query Language within Hibernate Framework
- Example queries:

```
select cat
from Cat cat
    join cat.kittens kitten
group by cat.id, cat.name, cat.other, cat.properties
having avg(kitten.weight) > 100
order by count(kitten) asc, sum(kitten.weight) desc
```

```
( (Integer) session.createQuery("select count(*)
from ....").iterate().next() ).intValue()
```