B0B36DBS, BD6B36DBS: **Database Systems**

http://www.ksi.mff.cuni.cz/~svoboda/courses/172-B0B36DBS/

Practical Class 10

JDBC, JPA 2.1

Author: **Martin Svoboda**, martin.svoboda@fel.cvut.cz
Tutors: **J. Ahmad**, **R. Černoch**, **M. Řimnáč**, **M. Svoboda**, **G. Šourek**

24. 4. 2018

Czech Technical University in Prague, Faculty of Electrical Engineering

JDBC

Java Database Connectivity

First Steps

Create a new NetBeans project

- Select *Java Application* as a project type
- Create a main class `Main1`
  - Include `public static void main` method
- Add **PostgreSQL JDBC Driver** library
  - *Projects* window → your *Project* node → *Libraries* context menu → *Add Library* item

Database Connection

Dynamically load the JDBC driver for PostgreSQL

```
Class.forName("org.postgresql.Driver");
```

Open a new database connection

```
import java.sql.*;
```

```
Connection c = DriverManager.getConnection(
  "jdbc:postgresql://slon.felk.cvut.cz:5432/database",
  "user",
  "password"
);
```

Close the database connection

```
c.close();
```

Table Definition

Create a schema for a table of books

- Include the following columns
  - `id`: book identifier, integer, primary key
  - `title`: book title, varchar

```
Statement s1 = c.createStatement();
s1.execute(
  "CREATE TABLE book ..."
);
```

Insert sample data into our table of books

```
Statement s2 = c.createStatement();
s2.execute(
  "INSERT INTO book VALUES (1, 'Proces'), (2, 'Zámek')"
);
```

Data Querying

Select and process all books

- Print book identifiers and titles to the standard output

```
Statement s3 = c.createStatement();
ResultSet rs = s3.executeQuery(
  "SELECT id, title FROM book"
);
while (rs.next()) {
  System.out.println(
    "Id: " + rs.getInt("id") + " | title: " + rs.getString(2)
  );
}
```

Prepared Statements

Retrieve books based on their titles

- Use a prepared statement for this purpose

```
PreparedStatement ps = c.prepareStatement(
  "SELECT * FROM book WHERE (title = ?)"
);
ps.setString(1, "Zámek");
```

JPA 2.1

Java Persistence API

First Steps

Extend our NetBeans project

- Create a new main class `Main2`
  - Include `public static void main` method
  - Make this class the main class of the project
  - *Projects* window → your *Project* node context menu → *Properties* item → *Run* section → *Main Class* field
- Add **EclipseLink JPA 2.1** library
  - *Projects* window → your *Project* node → *Libraries* context menu → *Add Library* item

Database Connection

Configure your database connection in NetBeans

- Create and configure a new connection
  - *Services* window → *Databases* node context menu → *New Connection* item
    - *Driver*: PostgreSQL
    - *Host*: `slon.felk.cvut.cz`
    - *Port*: `5432`
    - Fill in your *Database*, *User name*, and *Password*

Persistence Unit

Create and configure a new *persistence unit*

- I.e. add a new `persistence.xml` file into your project
  - *Projects* window → your *Project* node context menu → *New* item → *Other* subitem
    - *Category*: Persistence
    - *File type*: Persistence Unit
- Configure the persistence unit
  - Set *Persistence Unit Name*
  - Select your *Database Connection*
  - *Table Generation Strategy*: Create

Browse XML source code of your persistence unit

Entity Definition

Create a new entity for books

- I.e. add a new Java class into your project
- Include the following attributes
  - `id`: book identifier, integer, primary key
  - `title`: book title, string
  - `comment`: auxiliary comment, string

```
public class Book {
  public Integer id;
  public String title;
  public String comment;
  public Book() {
    this.id = null;
    this.title = null;
    this.comment = null;
  }
}
```

Entity Annotations

Use annotations to make sure that…

- Class *Book* is treated as a database entity
- *book2* is a name of the associated table
- Values of *id* column are automatically generated
- *booktitle* is a name of column for book titles
- `NULL` values are not permitted for book titles
- Attribute *comment* is not included in the database

```
import javax.persistence.*;
```

Entity Registration

Register the book entity with our persistence unit

- *persistence.xml* file → *Design* view → *General* section → *Include Entity Classes* list → *Add Class…* button

Entity Manager

Create a new instance of an entity manager factory

```
EntityManagerFactory emf =
  Persistence.createEntityManagerFactory("PersistenceUnitName");
```

Create a new instance of an entity manager

```
EntityManager em =
  emf.createEntityManager();
```

Close the database connection

```
em.close();
emf.close();
```

Data Querying

Retrieve a book according to its identifier

- Print book identifier and title to the standard output

```
Book b1 = em.find(Book.class, 2);
```

JPQL Statements

Retrieve and process all books

- Print book identifiers and titles to the standard output

```
import java.util.List;
```

```
TypedQuery<Book> q2 = em.createQuery(
  "SELECT b FROM Book AS b",
  Book.class
);
List<Book> l = q2.getResultList();
for (Book b : l) {
  ...
}
```

Parameterized Queries

Retrieve a book using a parameterized query

```
TypedQuery<Book> q3 = em.createQuery(
  "SELECT b FROM Book AS b WHERE (b.id = :BookId)",
  Book.class
);
q3.setParameter("BookId", 2);
Book b3 = q3.getSingleResult();
```

Entity Transactions

Initialize a new entity transaction

```
EntityTransaction et = em.getTransaction();

et.begin();
...
et.commit();
```

Entity Manipulation

Create a new book entity

```
et.begin();
Book b4 = new Book();
b4.title = "Amerika";
em.persist(b4);
et.commit();
```

Update the previous book entity

```
et.begin();
b4.title = "Nezvěstný";
et.commit();
```

Remove the previous book entity

```
et.begin();
em.remove(b4);
et.commit();
```