B0B36DBS: **Database Systems**

Practical Class

JDBC, JPA 2.1

Author: **Martin Řimnáč**

Czech Technical University in Prague, Faculty of Electrical Engineering

JDBC

Java Database Connecvity

Database Connection

Dynamically load the JDBC driver for PostgreSQL

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

Open a new database conneticon

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

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

Close the database connecon

```
connection.close();
```

Table Definion

Create a schema for a table of books

- Include the following columns
  - id : book iden fier, integer, primary key
  - title : book tle, varchar

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

Insert sample data into our table of books

```
Statement s2 = connection.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 = connection.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 the parametrized query

```
ResultList getBooksByTitle (String bookTitle) {
        PreparedStatement ps = connection.prepareStatement(
                "SELECT * FROM book WHERE title = ?" );
        ps.setString(1, bookTitle);
        return ps.executeQuery();
}
```

SQL Injection

Retrieve books based on their titles

- Use a prepared statement for parametrized query

```
ResultList getBooksByTitle (String bookTitle) {
        PreparedStatement ps = connection.prepareStatement(
                "SELECT * FROM book WHERE title = ?" );
        ps.setString(1, bookTitle);
        return ps.executeQuery();
}
```

- Don’t use a SQL query string concatenation

```
ResultList getBooksByTitle (String bookTitle) {
        Statement st = connection.createStatement();
        st.execute(
                "SELECT * FROM book WHERE title = ‘“ + bookTitle + “’“ );
        return ps.executeQuery();
}
```

Dao.getBooksByTitle(“‘ OR “=‘“) **??!!**

JPA 2.1

Java Persistence API

Java Persistence API

Your interface to data sources - Persistence

- generic data access object (DAO) for processing data sources
- for each data source – creates EntityManagerFactory

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

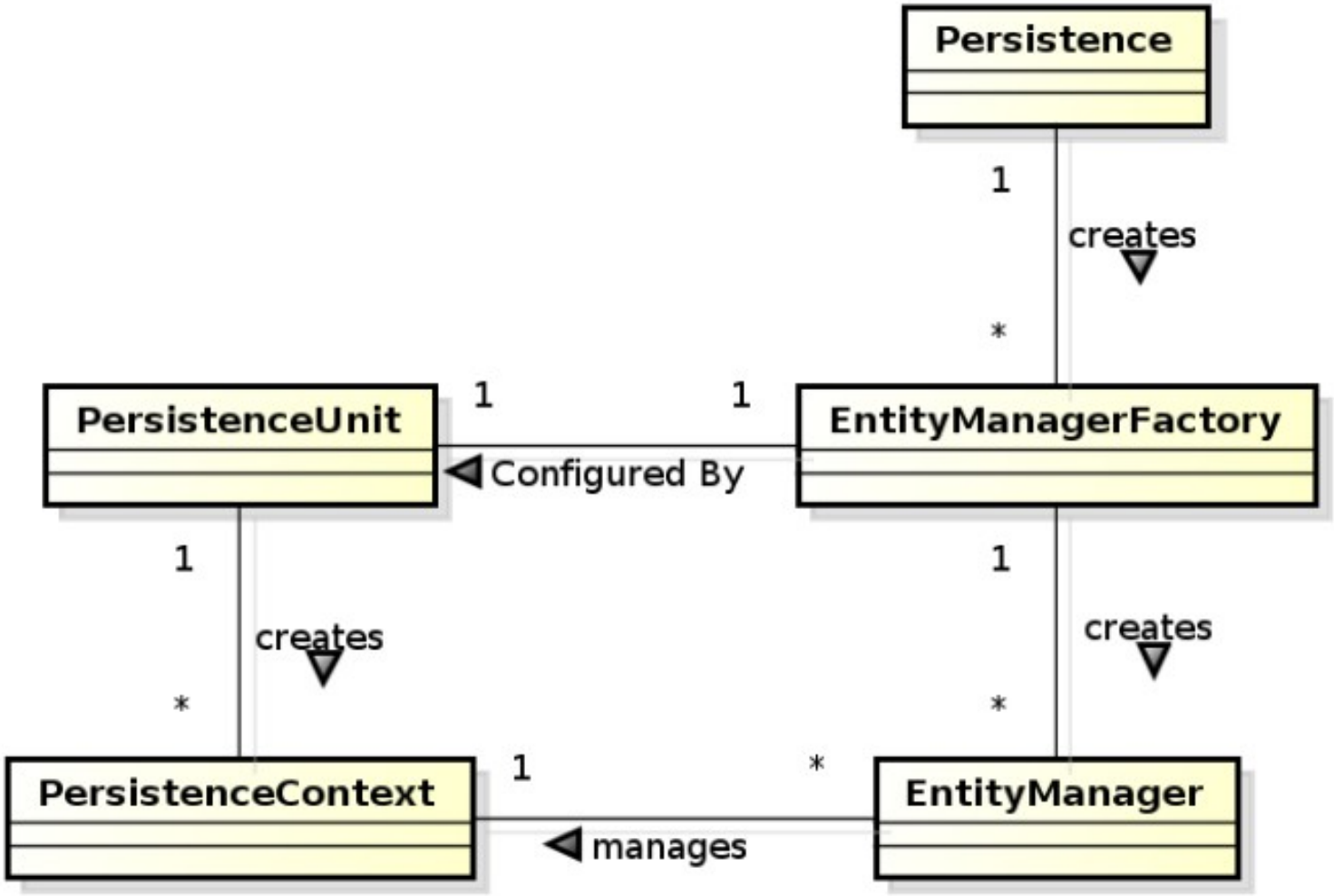
- the data source configured by – **Persistence Unit**
  (create own Persitence unit – e.g. BooksPU)

- EntityManagerFactory creates **EntityManager** for data processing

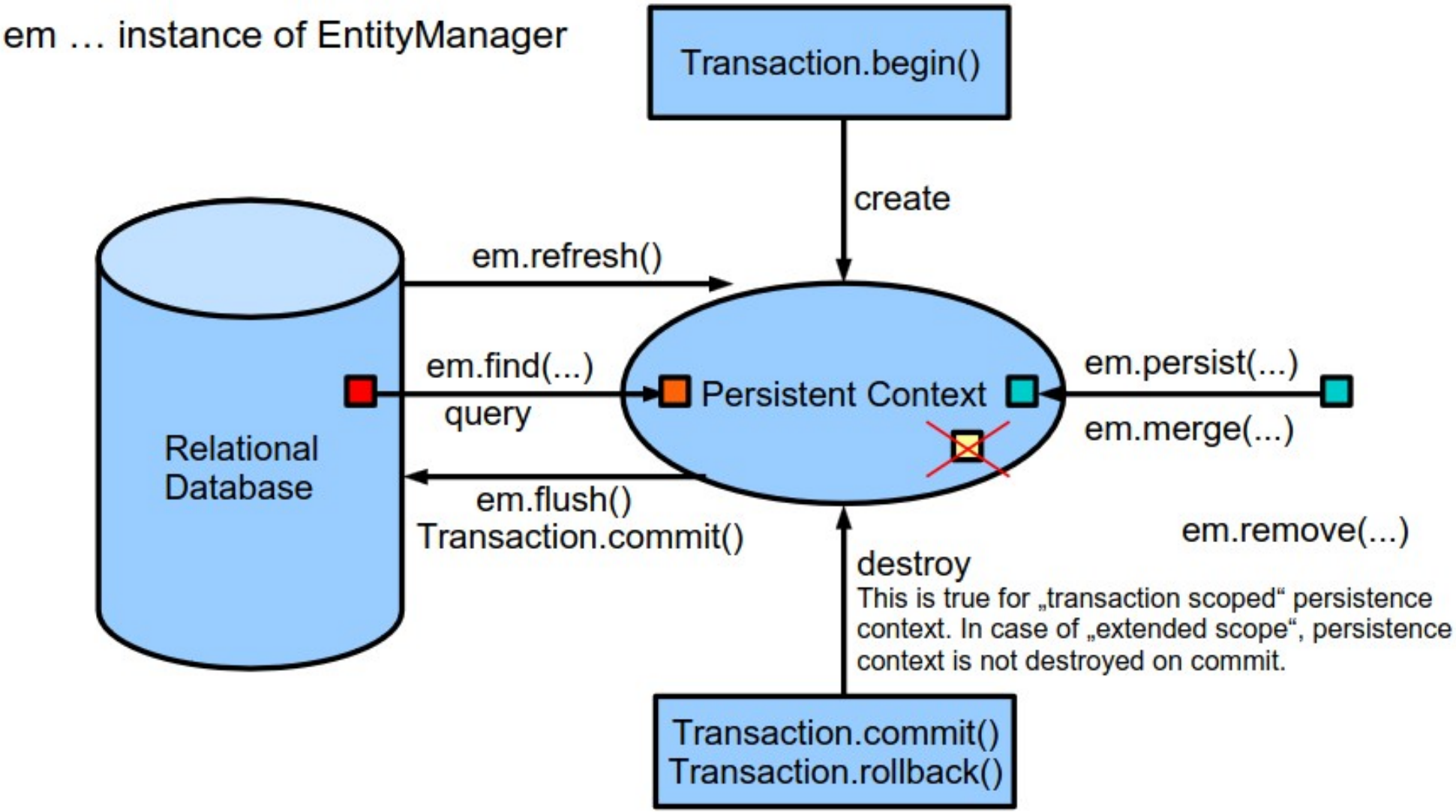
```
EntityManager em = emf.createEntityManager();
```

- in DAO – use the Entity manager to
  - find()/refresh() the entity instance
  - persist()/merge()/remove() the entity instance

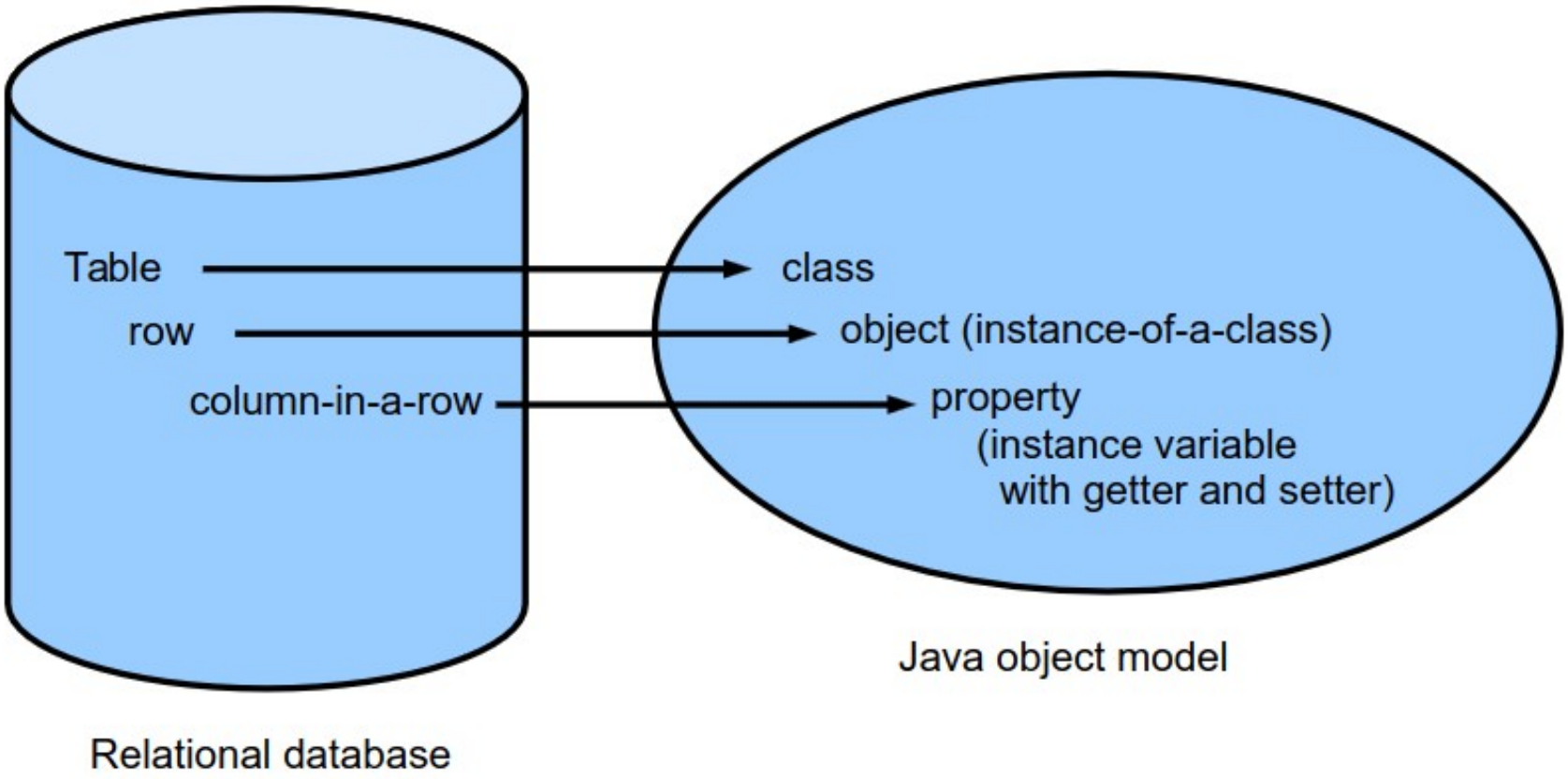
Java Persistence API



Persistence Context



Object – Relational Mapping (ORM)



Database Connection

Configure your database connection in IDE

- Create and configure a new connecon
  - *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
- Alternative: Create a new project with existing database

Entity Definition – from scratch

POJO class for entity modelling

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

```
@Entity
public class Book {
        @Id
        @GeneratedValue
        private Integer id;
        private String title;
        private String comment;

        public Book () { …}
        //setters/getters
}
```

Entity Manger - Data Querying

Retrieve a book according to its identifier (a value of PK)

- Get the instance of the book with the primary key value bookID

```
Book  getBookById (Integer bookID) {
        return em.find(Book.class, bookID).getSingleResult();
}
```

JPQL Statements

Retrieve all books using Java Persistency Query Language (JPQL)

```
List <Book>  getAllBooks () {
            return em.createQuery("SELECT b FROM Book AS b", Book.class)
                        .getResultList();
}
```

Retrieve all books of given title using JPQL (parametrized query)

```
List <Books> getBooksByTitle (string bookTitle) {
            return em.createQuery(
                        "SELECT b FROM Book AS b WHERE b.title=:BTitle", Book.class)
                        .setParameter("BTitle", bookTitle).
                        .getResultList();
}
```

Persistence Transactions

Initialize a new transaction

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

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

Entity Create/Update/Delete

```
Book createBookOfTitle( String bookTitle) {
        Book book = new Book();
        book.setTitle(bookTitle);
        em.persist(book);
        return book;
}
Book renameBookByID( Integer bookID, String bookTitle) {
        et.begin();
        Book book = this.getBookById(bookID);
        book.setTitle(bookTitle);
        em.merge(book);
        et.commit();
        return book;
}
void removeBookByID( Integer bookID) {
        et.begin();
        Book book = this.getBookById(bookID);
        em.remove(book);
        et.commit();
}
```