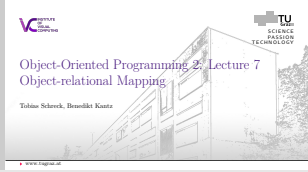
INSTITUTE OF VISUAL COMPUTING

TU Graz
SCIENCE
PASSION
TECHNOLOGY

Object-Oriented Programming 2: Lecture 7 Object-relational Mapping

Tobias Schreck, Benedikt Kantz



Code Examples

We will cover some code examples from last week(s) first!

Code Examples

We will cover some code examples from last week(s) first!

Have you ever used a Database for an application?

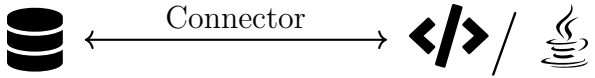
$\Longrightarrow$ Noticed any problems?

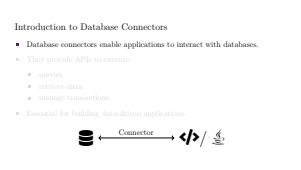
Have you ever used a Database for an application?
$\Longrightarrow$ Noticed any problems?

- Ask students for their experiences, write down relevant answers on blackoard - goal would be to make students aware of DRY violations

Introduction to Database Connectors

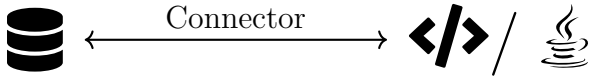
- Database connectors enable applications to interact with databases.
- They provide APIs to execute
  - queries
  - retrieve data
  - manage transactions
- Essential for building data-driven applications.

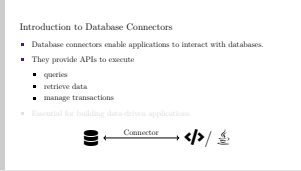




Introduction to Database Connectors

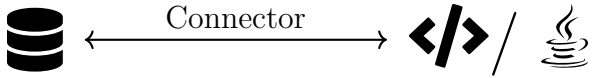
- Database connectors enable applications to interact with databases.
- They provide APIs to execute
  - queries
  - retrieve data
  - manage transactions
- Essential for building data-driven applications.





Introduction to Database Connectors

- Database connectors enable applications to interact with databases.
- They provide APIs to execute
  - queries
  - retrieve data
  - manage transactions
- Essential for building data-driven applications.



Introduction to Database Connectors
- Database connectors enable applications to interact with databases.
- They provide APIs to execute
  - queries
  - retrieve data
  - manage transactions
- Essential for building data-driven applications.
Connector

sqlite3

Python → SQLite

- Built-in Python module
- Lightweight, file-based database
- Great for prototyping and small apps

```python
import sqlite3
conn = sqlite3.connect('example.db')
cur = conn.cursor()
cur.execute("SELECT * FROM users;")
for row in cur:
        print(row)
conn.close()
```


sqlite3
Python → SQLite
- Built-in Python module
- Lightweight, file-based database
- Great for prototyping and small apps
import sqlite3
conn = sqlite3.connect('example.db')
cur = conn.cursor()
cur.execute("SELECT * FROM users;")
for row in cur:
print(row)
conn.close()

psycopg
Python → PostgreSQL

- Popular PostgreSQL adapter for Python
- Used in Django, SQLAlchemy, etc.
- Supports advanced PostgreSQL features

```
import psycopg
conn = psycopg.connect(
        "dbname=test user=postgres
        ↪  password=secret")
cur = conn.cursor()
cur.execute("SELECT * FROM users;")
for row in cur:
        print(row)
conn.close()
```


psycopg
Python → PostgreSQL
- Popular PostgreSQL adapter for Python
- Used in Django, SQLAlchemy, etc.
- Supports advanced PostgreSQL features

JDBC

Java → Many Databases (PostgreSQL, MySQL, etc.)

- Java Database Connectivity API
- Works with many RDBMS via drivers
- Standard for Java database access

```
import java.sql.*;
Connection conn = DriverManager.getConnection(
        "jdbc:postgresql://localhost/test",
        "user", "password");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(
        "SELECT * FROM users");
while (rs.next()) {
        System.out.println(
            rs.getString("name"));
}
conn.close();
```


JDBC: More Details I

Core Components:

- `DriverManager`: Manages database drivers.
- `Connection`: Represents a session with the database.
- `Statement` and `PreparedStatement`: Execute SQL queries.
- `ResultSet`: Holds data returned by queries.

JDBC: More Details I

Core Components:

- `DriverManager`: Manages database drivers.
- `Connection`: Represents a session with the database.
- `Statement` and `PreparedStatement`: Execute SQL queries.
- `ResultSet`: Holds data returned by queries.

JDBC: More Details II

- Similar principles in other languages' connectors:
  - Connection pooling
  - Transaction management
  - Query execution

JDBC: More Details II

- Similar principles in other languages' connectors:
  - Connection pooling
  - Transaction management
  - Query execution

JDBC/Connectors: Advantages vs. Limitations

Advantages

- Database-agnostic API (for Java applications).
- Supports transactions, batch updates, and metadata inspection.

JDBC/Connectors: Advantages vs. Limitations

Advantages

- Database-agnostic API (for Java applications).
- Supports transactions, batch updates, and metadata inspection.

JDBC/Connectors: Advantages vs. Limitations

Advantages

- Database-agnostic API (for Java applications).
- Supports transactions, batch updates, and metadata inspection.

Limitations

- Manual resource management (connections, statements).
- Verbose compared to higher-level ORMs.

JDBC/Connectors: Advantages vs. Limitations

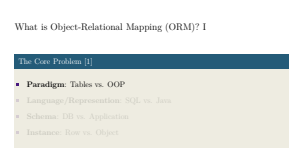
Advantages
- Database-agnostic API (for Java applications).
- Supports transactions, batch updates, and metadata inspection.

Limitations
- Manual resource management (connections, statements).
- Verbose compared to higher-level ORMs.

What is Object-Relational Mapping (ORM)? I

The Core Problem [1]

- **Paradigm**: Tables vs. OOP
- **Language/Represention**: SQL vs. Java
- **Schema**: DB vs. Application
- **Instance**: Row vs. Object



Object-Oriented Programming (OOP) uses objects, while most popular databases are Relational (RDBMS) and use tables. These two paradigms don't naturally align perfectly. This difference is often called the **Object-Relational Impedance Mismatch**.

What is Object-Relational Mapping (ORM)? I

The Core Problem [1]

- **Paradigm**: Tables vs. OOP
- **Language/Represention**: SQL vs. Java
- **Schema**: DB vs. Application
- **Instance**: Row vs. Object

What is Object-Relational Mapping (ORM)? I
The Core Problem [1]
- **Paradigm**: Tables vs. OOP
- **Language/Represention**: SQL vs. Java
- **Schema**: DB vs. Application
- **Instance**: Row vs. Object

Object-Oriented Programming (OOP) uses objects, while most popular databases are Relational (RDBMS) and use tables. These two paradigms don't naturally align perfectly. This difference is often called the **Object-Relational Impedance Mismatch**.

What is Object-Relational Mapping (ORM)? I

The Core Problem [1]

- **Paradigm**: Tables vs. OOP
- **Language/Represention**: SQL vs. Java
- **Schema**: DB vs. Application
- **Instance**: Row vs. Object

What is Object-Relational Mapping (ORM)? I
The Core Problem [1]
- **Paradigm**: Tables vs. OOP
- **Language/Represention**: SQL vs. Java
- **Schema**: DB vs. Application
- **Instance**: Row vs. Object

Object-Oriented Programming (OOP) uses objects, while most popular databases are Relational (RDBMS) and use tables. These two paradigms don't naturally align perfectly. This difference is often called the **Object-Relational Impedance Mismatch**.

What is Object-Relational Mapping (ORM)? I

The Core Problem [1]

- **Paradigm**: Tables vs. OOP
- **Language/Represention**: SQL vs. Java
- **Schema**: DB vs. Application
- **Instance**: Row vs. Object

What is Object-Relational Mapping (ORM)? I

The Core Problem [1]

- **Paradigm**: Tables vs. OOP
- **Language/Represention**: SQL vs. Java
- **Schema**: DB vs. Application
- **Instance**: Row vs. Object

Object-Oriented Programming (OOP) uses objects, while most popular databases are Relational (RDBMS) and use tables. These two paradigms don't naturally align perfectly. This difference is often called the **Object-Relational Impedance Mismatch**.

What is Object-Relational Mapping (ORM)? II

The Solution: **Object-Relational Mapping (ORM)**

→ solve these issue (semi-) automatically

- Interact with DB with the syntax of (most) OOP languages.
- Repetitive SQL boilerplate code abstracted (CRUD operations).

What is Object-Relational Mapping (ORM)? II
The Solution: Object-Relational Mapping (ORM)
→ solve these issue (semi-) automatically
- Interact with DB with the syntax of (most) OOP languages.
- Repetitive SQL boilerplate code abstracted (CRUD operations).

CRUD - Create, Read, Update, Delete

The Mapping

Object Model (OOP)

User
+ id : int (PK) + username : string + email : string
+ save() + delete()

The Mapping

Object Model (OOP)
User
+ id : int (PK)
+ username : string
+ email : string
+ save()
+ delete()

The Mapping

Object Model (OOP)

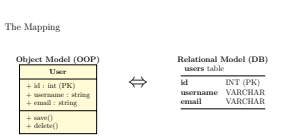
User
+ id : int (PK) + username : string + email : string
+ save() + delete()

⇔

Relational Model (DB)

users table

id	INT (PK)
username	VARCHAR
email	VARCHAR



The Mapping

Object Model (OOP)

User
+ id : int (PK) + username : string + email : string
+ save() + delete()

$\Leftrightarrow$

Relational Model (DB)

users table

id	INT (PK)
username	VARCHAR
email	VARCHAR

$\Longrightarrow$ The ORM handles the translation between these representations.

The Mapping

Object Model (OOP)
User
+ id : int (PK)
+ username : string
+ email : string
+ save()
+ delete()
⇔
Relational Model (DB)
users table
id INT (PK)
username VARCHAR
email VARCHAR
⟹ The ORM handles the translation between these representations.

Jakarta Persistence (JPA)

Overview[1]

Jakarta Persistence provides Java developers with an object/relational mapping facility for managing relational data in Java applications.

[1] https://jakarta.ee/

Jakarta Persistence (JPA)
Overview[1]
Jakarta Persistence provides Java developers with an object/relational mapping facility for managing relational data in Java applications.
[1] https://jakarta.ee/

Hibernate Example

Introduction

Description[2]

Hibernate makes relational data visible to a program written in Java, in a natural and type-safe form,

- making it easy to write complex queries and work with their results,
- letting the program easily synchronize changes made in memory with the database,
- respecting the ACID properties of transactions, ...

[2] https://hibernate.org/orm/

Hibernate Example
Introduction
Description[2]
Hibernate makes relational data visible to a program written in Java, in a natural and type-safe form,
- making it easy to write complex queries and work with their results,
- letting the program easily synchronize changes made in memory with the database,
- respecting the ACID properties of transactions, ...
[2] https://hibernate.org/orm/

Hibernate Example
Mapping a Class to a Table

SQL Table:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);
```


Hibernate Example
Mapping a Class to a Table
SQL Table:
CREATE TABLE users (
id SERIAL PRIMARY KEY,
username VARCHAR(50) NOT NULL,
email VARCHAR(100) NOT NULL
);

Hibernate Example

Mapping a Class to a Table

SQL Table:

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);
```

Java Class:

```java
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy =
    ↪  GenerationType.IDENTITY)
    private Long id;
    @Column(name = "username", nullable = false)
    private String username;
    @Column(name = "email", nullable = false)
    private String email;
}
```


Hibernate Example
Adding data

- Similar to JDBC:
  - Session & transaction start
  - Add the data
  - Commit!

```
Session session = factory.getCurrentSession();
User newUser = new User();
newUser.setUsername("MrP01");
newUser.setEmail("hello@example.com");

session.beginTransaction();
session.save(newUser);
session.getTransaction().commit();
```


Hibernate Example
Adding data

- Similar to JDBC:
  - Session & transaction start
  - Add the data
  - Commit!

```
Session session = factory.getCurrentSession();
User newUser = new User();
newUser.setUsername("MrP01");
newUser.setEmail("hello@example.com");

session.beginTransaction();
session.save(newUser);
session.getTransaction().commit();
```


Hibernate Example
Adding data

- Similar to JDBC:
  - Session & transaction start
  - Add the data
  - Commit!

```
Session session = factory.getCurrentSession();
User newUser = new User();
newUser.setUsername("MrP01");
newUser.setEmail("hello@example.com");

session.beginTransaction();
session.save(newUser);
session.getTransaction().commit();
```


Hibernate Example
Adding data

- Similar to JDBC:
  - Session & transaction start
  - Add the data
  - Commit!

```
Session session = factory.getCurrentSession();
User newUser = new User();
newUser.setUsername("MrP01");
newUser.setEmail("hello@example.com");

session.beginTransaction();
session.save(newUser);
session.getTransaction().commit();
```


Hibernate Example
Writing Queries

- Querying data:
  - Start a session
  - Write the query
  - Fetch results

```
Session session = factory.getCurrentSession();
session.beginTransaction();

List<User> users = session
    .createQuery("from User", User.class)
    .getResultList();

for (User user : users) {
    System.out.println(user.getUsername());
}

session.getTransaction().commit();
```


Hibernate Example
Writing Queries

- Querying data:
  - Start a session
  - Write the query
  - Fetch results

```
Session session = factory.getCurrentSession();
session.beginTransaction();

List<User> users = session
    .createQuery("from User", User.class)
    .getResultList();

for (User user : users) {
    System.out.println(user.getUsername());
}

session.getTransaction().commit();
```


Hibernate Example
Writing Queries

- Querying data:
  - Start a session
  - Write the query
  - Fetch results

```
Session session = factory.getCurrentSession();
session.beginTransaction();

List<User> users = session
    .createQuery("from User", User.class)
    .getResultList();

for (User user : users) {
    System.out.println(user.getUsername());
}

session.getTransaction().commit();
```


Hibernate Example
Writing Queries

- Querying data:
  - Start a session
  - Write the query
  - Fetch results

```
Session session = factory.getCurrentSession();
session.beginTransaction();

List<User> users = session
    .createQuery("from User", User.class)
    .getResultList();

for (User user : users) {
    System.out.println(user.getUsername());
}

session.getTransaction().commit();
```


SQLAlchemy Example
Introduction

Why SQLAlchemy? [3]

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

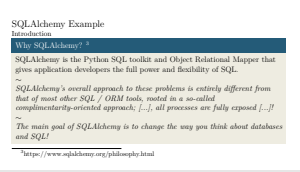
∼

SQLAlchemy's overall approach to these problems is entirely different from that of most other SQL / ORM tools, rooted in a so-called complimentarity-oriented approach; [...], all processes are fully exposed [...]!

∼

The main goal of SQLAlchemy is to change the way you think about databases and SQL!

[3] https://www.sqlalchemy.org/philosophy.html



SQLAlchemy Example
Mapping a Class to a Table

SQL Table:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);
```


SQLAlchemy Example
Mapping a Class to a Table
SQL Table:


SQLAlchemy Example

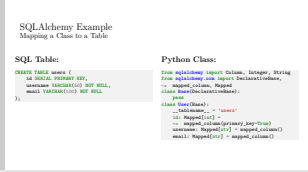
Mapping a Class to a Table

SQL Table:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);
```

Python Class:

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import DeclarativeBase,
↪  mapped_column, Mapped
class Base(DeclarativeBase):
    pass
class User(Base):
    __tablename__ = 'users'
    id: Mapped[int] =
    ↪  mapped_column(primary_key=True)
    username: Mapped[str] = mapped_column()
    email: Mapped[str] = mapped_column()
```

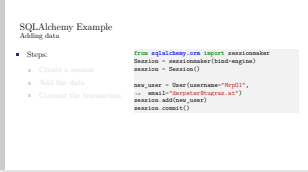



SQLAlchemy Example
Adding data

- Steps:
  - Create a session
  - Add the data
  - Commit the transaction

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()

new_user = User(username="MrpO1",
↪  email="derpeter@tugraz.at")
session.add(new_user)
session.commit()
```

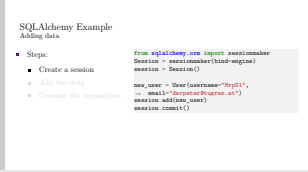



SQLAlchemy Example
Adding data

- Steps:
  - Create a session
  - Add the data
  - Commit the transaction

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()

new_user = User(username="MrpO1",
↪  email="derpeter@tugraz.at")
session.add(new_user)
session.commit()
```

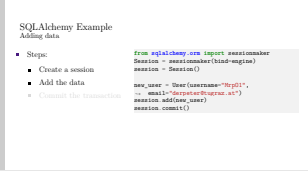



SQLAlchemy Example
Adding data

- Steps:
  - Create a session
  - Add the data
  - Commit the transaction

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()

new_user = User(username="MrpO1",
↪  email="derpeter@tugraz.at")
session.add(new_user)
session.commit()
```

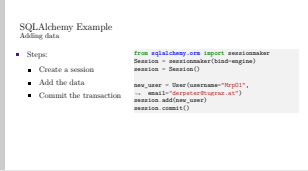



SQLAlchemy Example
Adding data

- Steps:
  - Create a session
  - Add the data
  - Commit the transaction

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
session = Session()

new_user = User(username="MrpO1",
↪  email="derpeter@tugraz.at")
session.add(new_user)
session.commit()
```

SQLAlchemy Example
Writing Queries

- Querying data:
  - Create a session
  - Write the query
  - Fetch results

```
session = Session()

users = session.query(User).all()

for user in users:
    print(user.username)
```


SQLAlchemy Example
Writing Queries
Querying data:
Create a session
Write the query
Fetch results
session = Session()
users = session.query(User).all()
for user in users:
print(user.username)

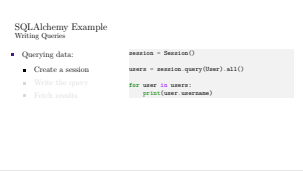
SQLAlchemy Example
Writing Queries

- Querying data:
  - Create a session
  - Write the query
  - Fetch results

```
session = Session()

users = session.query(User).all()

for user in users:
    print(user.username)
```

SQLAlchemy Example
Writing Queries

- Querying data:
  - Create a session
  - Write the query
  - Fetch results

```
session = Session()

users = session.query(User).all()

for user in users:
    print(user.username)
```

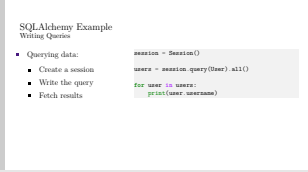
SQLAlchemy Example
Writing Queries

- Querying data:
  - Create a session
  - Write the query
  - Fetch results

```python
session = Session()

users = session.query(User).all()

for user in users:
    print(user.username)
```


MY ORM!
I CANNOT TYPE QUERIES WITHOUT MY ORM


MY ORM!
I CANNOT TYPE QUERIES WITHOUT MY ORM

The Object-Relational Impedance Mismatch

Aspect	OOP Concept	DB Concept

The Object-Relational Impedance Mismatch
Aspect OOP Concept DB Concept

The Object-Relational Impedance Mismatch

Aspect	**OOP Concept**	**DB Concept**
Granularity	Small/large composites	Flat rows

The Object-Relational Impedance Mismatch

Aspect	OOP Concept	DB Concept
Granularity	Small/large composites	Flat rows

The Object-Relational Impedance Mismatch

Aspect	**OOP Concept**	**DB Concept**
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys

The Object-Relational Impedance Mismatch

Aspect	OOP Concept	DB Concept
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys

The Object-Relational Impedance Mismatch

Aspect	**OOP Concept**	**DB Concept**
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables

The Object-Relational Impedance Mismatch

Aspect	OOP Concept	DB Concept
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables

The Object-Relational Impedance Mismatch

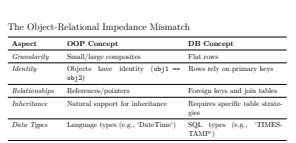
Aspect	**OOP Concept**	**DB Concept**
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables
Inheritance	Natural support for inheritance	Requires specific table strategies

The Object-Relational Impedance Mismatch

Aspect	OOP Concept	DB Concept
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables
Inheritance	Natural support for inheritance	Requires specific table strategies

The Object-Relational Impedance Mismatch

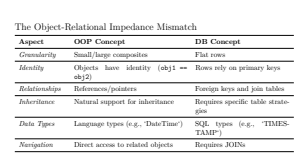
Aspect	**OOP Concept**	**DB Concept**
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables
Inheritance	Natural support for inheritance	Requires specific table strategies
Data Types	Language types (e.g., 'DateTime')	SQL types (e.g., 'TIMESTAMP')


The Object-Relational Impedance Mismatch

Aspect	OOP Concept	DB Concept
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (obj1 == obj2)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables
Inheritance	Natural support for inheritance	Requires specific table strategies
Data Types	Language types (e.g., 'DateTime')	SQL types (e.g., 'TIMESTAMP')

The Object-Relational Impedance Mismatch

Aspect	**OOP Concept**	**DB Concept**
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (`obj1 == obj2`)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables
Inheritance	Natural support for inheritance	Requires specific table strategies
Data Types	Language types (e.g., 'DateTime')	SQL types (e.g., 'TIMESTAMP')
Navigation	Direct access to related objects	Requires JOINs


The Object-Relational Impedance Mismatch

Aspect	OOP Concept	DB Concept
Granularity	Small/large composites	Flat rows
Identity	Objects have identity (obj1 == obj2)	Rows rely on primary keys
Relationships	References/pointers	Foreign keys and join tables
Inheritance	Natural support for inheritance	Requires specific table strategies
Data Types	Language types (e.g., 'DateTime')	SQL types (e.g., 'TIMESTAMP')
Navigation	Direct access to related objects	Requires JOINs

How ORM Works: Core Concepts [2] I

Mapping

- Define mappings between classes and tables.
- Map object attributes to table columns - datatypes!
- Define relationships (one-to-one, one-to-many, many-to-many)
- Often done via:
  - Decorators/Annotations (e.g., JPA, EF Core)
  - Configuration files (e.g., XML in older Hibernate)
  - Code-based configuration (e.g., SQLAlchemy mapping)

How ORM Works: Core Concepts [2] I
Mapping
- Define mappings between classes and tables.
- Map object attributes to table columns - datatypes!
- Define relationships (one-to-one, one-to-many, many-to-many)
- Often done via:
  - Decorators/Annotations (e.g., JPA, EF Core)
  - Configuration files (e.g., XML in older Hibernate)
  - Code-based configuration (e.g., SQLAlchemy mapping)

How ORM Works: Core Concepts [2] II

Querying

- Provide an API to query data using object-oriented concepts.
- Examples: LINQ (C#), JPQL/Criteria API (Java), SQLAlchemy's query syntax (Python).
- ORM translates these object queries into SQL.

How ORM Works: Core Concepts [2] II
Querying
- Provide an API to query data using object-oriented concepts.
- Examples: LINQ (C#), JPQL/Criteria API (Java), SQLAlchemy's query syntax (Python).
- ORM translates these object queries into SQL.

How ORM Works: Core Concepts [2] III

Identity Map

- Ensures that each database row maps to exactly one object instance within a given session/context.
- Prevents inconsistencies if the same data is loaded multiple times.

How ORM Works: Core Concepts [2] III
Identity Map
- Ensures that each database row maps to exactly one object instance within a given session/context.
- Prevents inconsistencies if the same data is loaded multiple times.

How ORM Works: Core Concepts [2] IV

Unit of Work

- Tracks changes made to objects during a transaction.
- When the transaction commits, the ORM generates the necessary INSERT, UPDATE, DELETE SQL statements.
- Simplifies transaction management.

How ORM Works: Core Concepts [2] IV

Unit of Work

- Tracks changes made to objects during a transaction.
- When the transaction commits, the ORM generates the necessary INSERT, UPDATE, DELETE SQL statements.
- Simplifies transaction management.

How ORM Works: Core Concepts [2] V

Lazy/Eager Loading

- Control when related data is loaded from the database (on-demand vs. upfront).

How ORM Works: Core Concepts [2] V

Lazy/Eager Loading

- Control when related data is loaded from the database (on-demand vs. upfront).

Advantages of Using an ORM

- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- **Built-in Features:**
  - caching
  - transaction management
  - **security**

Advantages of Using an ORM
- **Increased Productivity:** Reduces boilerplate code (CRUD)
- Database Independence: Easier switch between different database systems (in theory).
- Improved Maintainability: Data access logic is more encapsulated.
- Built-in Features:
  - caching
  - transaction management
  - security

Problem: specific features/limits, e.g. pgvector, postgis, text limits

Advantages of Using an ORM

- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- **Built-in Features:**
  - caching
  - transaction management
  - **security**

Advantages of Using an ORM
- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- Improved Maintainability: Data access logic is more encapsulated.
- Built-in Features:
  - caching
  - transaction management
  - security

Problem: specific features/limits, e.g. pgvector, postgis, text limits

Advantages of Using an ORM

- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- **Built-in Features:**
  - caching
  - transaction management
  - **security**

Advantages of Using an ORM
- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- Built-in Features:
  - caching
  - transaction management
  - security

Problem: specific features/limits, e.g. pgvector, postgis, text limits

Advantages of Using an ORM

- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- **Built-in Features:**
  - caching
  - transaction management
  - **security**

Advantages of Using an ORM
- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- Built-in Features:
  - caching
  - transaction management
  - security

Problem: specific features/limits, e.g. pgvector, postgis, text limits

Advantages of Using an ORM

- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- **Built-in Features:**
  - caching
  - transaction management
  - **security**

Advantages of Using an ORM
- **Increased Productivity:** Reduces boilerplate code (CRUD)
- **Database Independence:** Easier switch between different database systems (in theory).
- **Improved Maintainability:** Data access logic is more encapsulated.
- **Built-in Features:**
  - caching
  - transaction management
  - **security**

Problem: specific features/limits, e.g. pgvector, postgis, text limits

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

Disadvantages of Using an ORM
- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

N+1: execute an additional query for each instance - slow!!

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- Leaky Abstraction: mapping not perfect, SQL still needed - debugging!
- Impedance Mismatch: OOP concepts (like complex inheritance) not mappable
- Potential for Inefficient Queries: e.g. "N+1 selects problem"

N+1: execute an additional query for each instance - slow!!

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

Disadvantages of Using an ORM
- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- Impedance Mismatch: OOP concepts (like complex inheritance) not mappable
- Potential for Inefficient Queries: e.g. "N+1 selects problem"

N+1: execute an additional query for each instance - slow!!

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

Disadvantages of Using an ORM
- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- Impedance Mismatch: OOP concepts (like complex inheritance) not mappable
- Potential for Inefficient Queries: e.g. "N+1 selects problem"

N+1: execute an additional query for each instance - slow!!

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

Disadvantages of Using an ORM
- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- Potential for Inefficient Queries: e.g. "N+1 selects problem"

N+1: execute an additional query for each instance - slow!!

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

Disadvantages of Using an ORM

- **Complexity / Learning Curve:** steep learning curve
- **Performance Overhead:** sometimes (for highly specialized code) slower, but usually well-optimised
- **Leaky Abstraction:** mapping not perfect, SQL still needed - debugging!
- **Impedance Mismatch:** OOP concepts (like complex inheritance) not mappable
- **Potential for Inefficient Queries:** e.g. "N+1 selects problem"

N+1: execute an additional query for each instance - slow!!

The $N + 1$ Problem
Problem

- ORM executes one query to fetch the parent objects

. . . then N additional queries to fetch related objects.

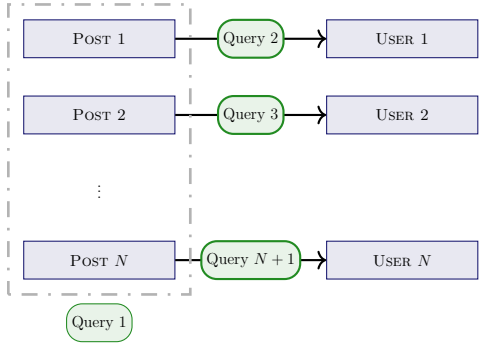
The $N + 1$ Problem
Problem
- ORM executes one query to fetch the parent objects
. . . then N additional queries to fetch related objects.

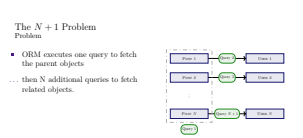
The $N + 1$ Problem

Problem

- ORM executes one query to fetch the parent objects

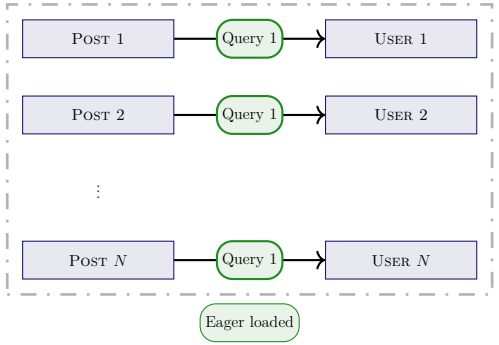
... then N additional queries to fetch related objects.





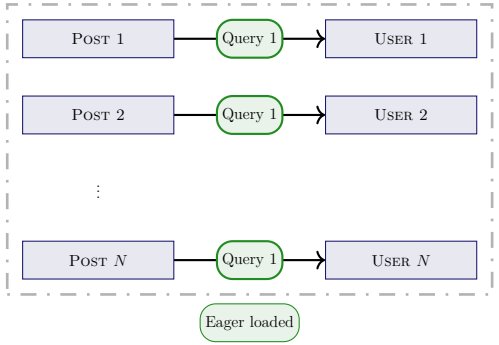
The N+1 Problem
(Potential) Solution

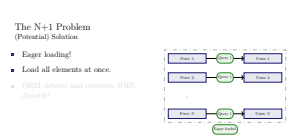
- Eager loading!
- Load all elements at once.
- ORM detects and executes JOIN directly!



The N+1 Problem
(Potential) Solution

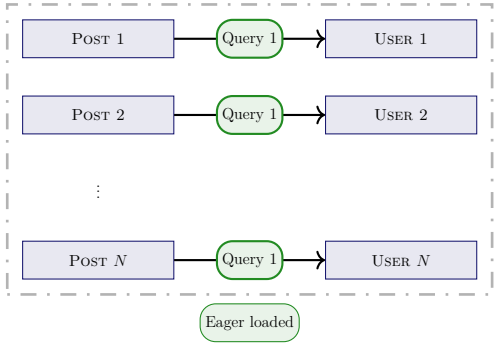
- Eager loading!
- Load all elements at once.
- ORM detects and executes JOIN directly!

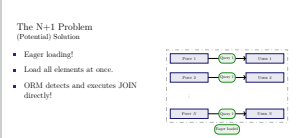




The N+1 Problem
(Potential) Solution

- Eager loading!
- Load all elements at once.
- ORM detects and executes JOIN directly!





Popular ORM Tools / Frameworks I

ORMs exist for almost every major programming language:

- **Python:**
  - SQLAlchemy (very powerful and flexible)
  - Django ORM (integrated into the Django web framework)
  - Peewee (lightweight)
- **Java:**
  - Hibernate (very mature and widely used)
  - JPA (Java Persistence API - a specification, Hibernate is an implementation)

Popular ORM Tools / Frameworks I
ORMs exist for almost every major programming language:
- **Python:**
  - SQLAlchemy (very powerful and flexible)
  - Django ORM (integrated into the Django web framework)
  - Peewee (lightweight)
- **Java:**
  - Hibernate (very mature and widely used)
  - JPA (Java Persistence API - a specification, Hibernate is an implementation)

Popular ORM Tools / Frameworks II

- **C# / .NET:**
  - Entity Framework Core (Microsoft's primary ORM)
  - NHibernate (port of Java's Hibernate)
  - Dapper (micro-ORM, focus on performance and SQL control)
- **Ruby:** ActiveRecord (part of Ruby on Rails)
- **PHP:** Doctrine, Eloquent (part of Laravel)
- **Node.js:** Sequelize, TypeORM, Prisma

Popular ORM Tools / Frameworks II
- **C# / .NET:**
  - Entity Framework Core (Microsoft's primary ORM)
  - NHibernate (port of Java's Hibernate)
  - Dapper (micro-ORM, focus on performance and SQL control)
- **Ruby:** ActiveRecord (part of Ruby on Rails)
- **PHP:** Doctrine, Eloquent (part of Laravel)
- **Node.js:** Sequelize, TypeORM, Prisma

Task Time! I

- Your Task: try out an ORM (on the object hierarchy)
- Use either:
  - SQLAlchemy
  - Hibernate
  - ?? (Language + ORM of choice - be creative!)

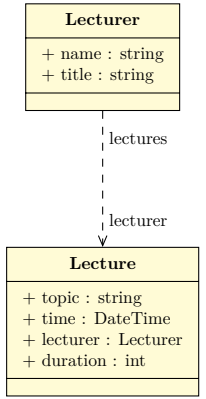


Figure: UML Diagram of a Lecturer and a Lecture

1.
2. Difficulties?
3. Setup/Installation?
4. How to specify mapping?
5. How can you map inheritance?
6. Is it possible to easily retrieve all lectures of a lecturer?

Task Time! I
- Your Task: try out an ORM (on the object hierarchy)
- Use either:
  - SQLAlchemy
  - Hibernate
  - ?? (Language + ORM of choice - be creative!)

Figure: UML Diagram of a Lecturer and a Lecture

Task Time! II

- What are your findings?

Task Time! II
- What are your findings?

Problems...

What was the main problem with ORMs?

Problems...
What was the main problem with ORMs?

Alternative: Object Stores / Object Databases

→ Store Objects directly!

- (this is difficult...) - why?
- Solutions:
  - Store Binary representations
  - Serialize

1. Problem: Impedance Mismatch - inheritance, ...
2. Problem: Versioning, Migration, ...
3. Binary: Problem: versioning
4. Serialize: Works quite well, interoperable

Alternative: Object Stores / Object Databases

→ Store Objects directly!

- (this is difficult...) - why?
- Solutions:
  - Store Binary representations
  - Serialize

1. Problem: Impedance Mismatch - inheritance, ...
2. Problem: Versioning, Migration, ...
3. Binary: Problem: versioning
4. Serialize: Works quite well, interoperable

Alternative: Object Stores / Object Databases

→ Store Objects directly!

- (this is difficult...) - why?
- Solutions:
  - Store Binary representations
  - Serialize

1. Problem: Impedance Mismatch - inheritance, ...
2. Problem: Versioning, Migration, ...
3. Binary: Problem: versioning
4. Serialize: Works quite well, interoperable

Alternative: Object Stores / Object Databases
→ Store Objects directly!
- (this is difficult...) - why?
- Solutions:
  1. Store Binary representations
  2. Serialize

Alternative: Object Stores / Object Databases

→ Store Objects directly!

- (this is difficult...) - why?
- Solutions:
  - Store Binary representations
  - Serialize

1. Problem: Impedance Mismatch - inheritance, ...
2. Problem: Versioning, Migration, ...
3. Binary: Problem: versioning
4. Serialize: Works quite well, interoperable

Alternative: Object Stores / Object Databases
→ Store Objects directly!
- (this is difficult...) - why?
- Solutions:
  - Store Binary representations
  - Serialize

Alternative: Object Stores / Object Databases

→ Store Objects directly!

- (this is difficult...) - why?
- Solutions:
  - Store Binary representations
  - Serialize

Alternative: Object Stores / Object Databases
→ Store Objects directly!
- (this is difficult...) - why?
- Solutions:
  - Store Binary representations
  - Serialize

1. Problem: Impedance Mismatch - inheritance, ...
2. Problem: Versioning, Migration, ...
3. Binary: Problem: versioning
4. Serialize: Works quite well, interoperable

Object Databases

- Problem: kind of “Out of fashion”, only commercial options . . .

Object Databases
- Problem: kind of “Out of fashion”, only commercial options . . .

Object Stores (Serialized)

Store	Features
MongoDB	A NoSQL database that stores data in flexible, JSON-like documents. Popular for its scalability and ease of use.
Couchbase	A distributed NoSQL database that supports JSON document storage and querying.
Redis	An in-memory data structure store that can be used for object storage with serialization (is Open Source *again*).
Postgres + JSONB	Extension for Postgres (SQL) to support serialized JSON data with querying, *Open Source!*

Object Stores (Serialized)

Store	Features
MongoDB	A NoSQL database that stores data in flexible, JSON-like documents. Popular for its scalability and ease of use.
Couchbase	A distributed NoSQL database that supports JSON document storage and querying.
Redis	An in-memory data structure store that can be used for object storage with serialization (is Open Source *again*).
Postgres + JSONB	Extension for Postgres (SQL) to support serialized JSON data with querying, *Open Source!*

Even more problems...

What problems do you still identify?

Even more problems...
What problems do you still identify?

Feedback

Feedback


Home
Mission
Why Data Veganism?
Blog
Contact Us
STOP JAVA.COM
GARBAGE COLLECTION IS DATA CRUELTY!
THINK! What's YOUR RAM footprint?
Java
The only "Garbage" around here is the Java virtual machine!
I'D RATHER GO BAREMETAL THAN VIRTUALIZE


STOP JAVA.COM
GARBAGE COLLECTION IS DATA CRUELTY!
I'D RATHER GO BAREMETAL THAN VIRTUALIZE

Bibliography I

References

[1] C. Ireland, D. Bowers, M. Newton, and K. Waugh, **A classification of object-relational impedance mismatch,** 2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications, 2009, pp. 36–43. DOI: 10.1109/DBKDA.2009.11.

[2] M. Nardone, L. Jungmann, M. Keith, and M. Schincariol, **Advanced object-relational mapping,** eng, *Pro Jakarta Persistence in Jakarta EE 10*, United States: Apress L. P, 2021, pp. 413–479, ISBN: 9781484274422.

Bibliography I

References

[1] C. Ireland, D. Bowers, M. Newton, and K. Waugh, **A classification of object-relational impedance mismatch,** 2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications, 2009, pp. 36–43. DOI: 10.1109/DBKDA.2009.11.

[2] M. Nardone, L. Jungmann, M. Keith, and M. Schincariol, **Advanced object-relational mapping,** eng, *Pro Jakarta Persistence in Jakarta EE 10*, United States: Apress L. P, 2021, pp. 413–479, ISBN: 9781484274422.