Large-Scale and Multi-Structured Databases

JDBC and Maven

Prof Pietro Ducange

Eng. José Corcuera

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Objective of this Class

- Let the student to think and remember his/her skills about programming in JAVA and using API for manipulating a relational database.
- We will recall some basics elements of JDBC.
- We will briefly introduce MAVEN
- Students will be asked to solve some programming exercises using an IDE (Eclipse or IntelliJ).

Motivations

Let suppose that our JAVA application needs to access and manipulate data stored in a ***Database Management System*** (DBMS).

A number of difficulties may appear (***Impedance Mismatch***), depending on:

- ***Different access modes***:
  SQL is a set-oriented language (operates on sets of tuples), while applications operate on a single tuple at a time
- ***Different types of data***:
  Each programming language defines its own types of data, which may differ from those of the SQL (and DBMS)
  Es: Java defines only one type to represent the strings (java.lang.String), SQL defines several: CHAR, VARCHAR, TEXT, LONGTEXT, ...

In case we try to map the concepts of the DB with in object-oriented languages there are also other difficulties called ***Object-Relational Impedance Mismatch.***

Connectors

DBMSs provide specific ***connectors*** (drivers) for certain programming languages.

Connectors are characterized by a ***low portability***: each DBMS has its own API, so the ***application*** becomes ***dependent*** on the specific DBMS.

In 1992 Microsoft introduced ***ODBC*** (Open DataBase Connectivity), which defines a unified API for accessing databases in C.

Java introduces ***JDBC*** (Java DataBase Connectivity): it is almost equivalent to ODBC, but provides a Java-specific API (java.sql.*)

JDBC Architecture

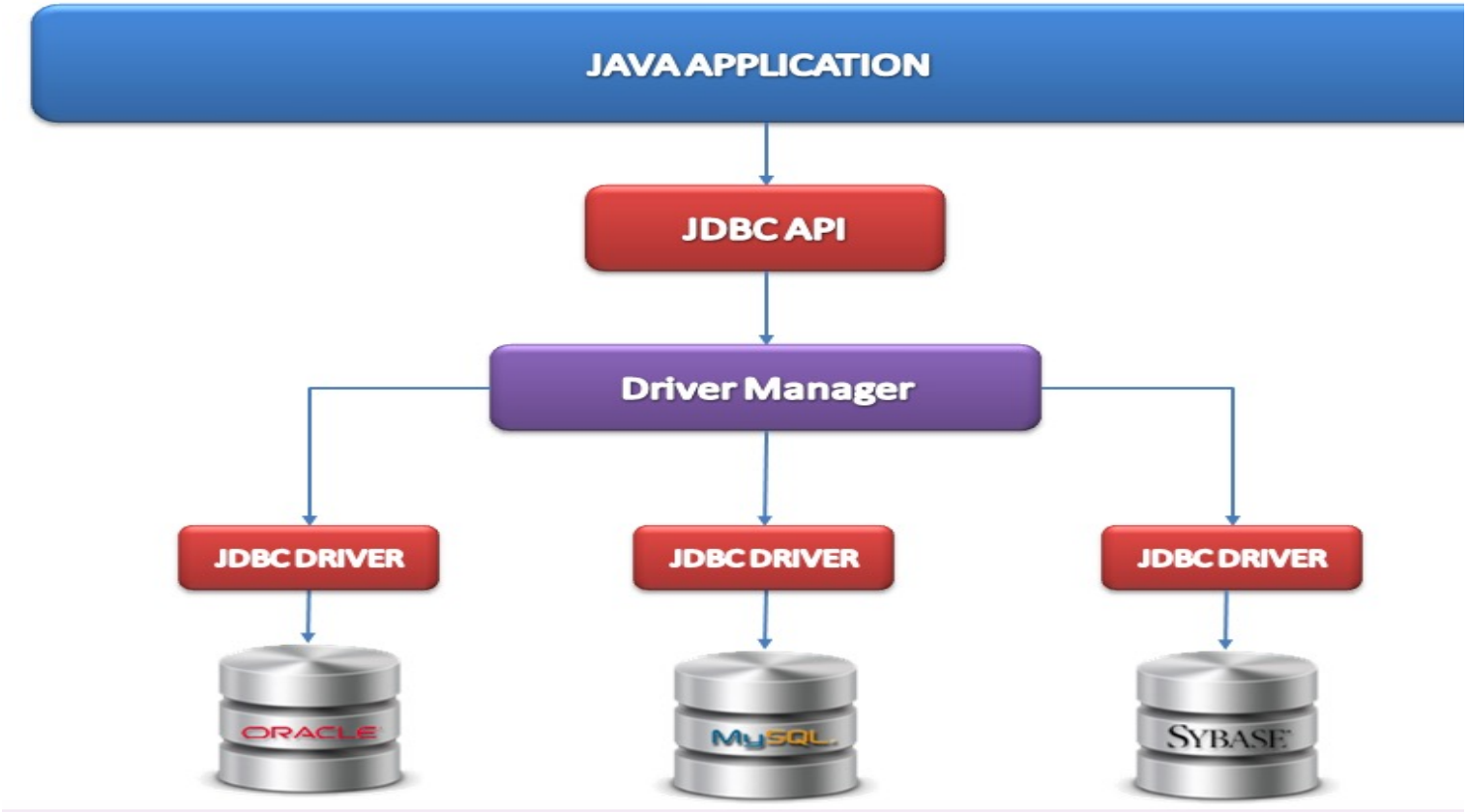


Image extracted from:

JDBC Driver Manager

The class ***DriverManager*** (java.sql.DriverManager) manages the JDBC drivers and ***creates connections*** to the DBMS.

A ***connection string*** is required to obtain a connection, in form of URN (***Uniform Resource Name***). The string is specific for DBMS and indicates:

- The type of driver
- The type of DBMS
- The host of the DBMS server
- Access credentials
- The database to which you can connect

Accessing the DB

```java
package it.unipi.lsmsd.example;

import java.sql.*;

public class ExampleConnect {

    public static void main(String[] args) {
        Connection connection = null;
        try{
            connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/unipi", user: "jose", password: "jose");
            // Do something...
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }  finally{
            if (connection != null) {
                try { connection.close();} catch (SQLException e) {}
            }
        }
    }
}
```

Returns from the DriverManager a connection to the DB specified by the connection string

At the end of all the operations we have to close the connection

Performing Queries

```
// Create a SQL statement
Statement stmt = conn.createStatement();
stmt.execute("SELECT * FROM user");

// Fetch results
ResultSet rs = stmt.getResultSet();
while (rs.next()){
    System.out.print(rs.getInt("id"));
    System.out.print(" ");
    System.out.print(rs.getString("first_name"));
    System.out.print(" ");
    System.out.println(rs.getString("last_name"));
}
rs.close();

// Close the statement and the connection
stmt.close();
conn.close();
```

The interface Statement creates an object used for executing a static SQL statement and returning the results it produces.

The ResultSet object allows to iterate on each tuple of the table returned by a query.

To obtain the values of the individual columns, specific methods are used for each data type

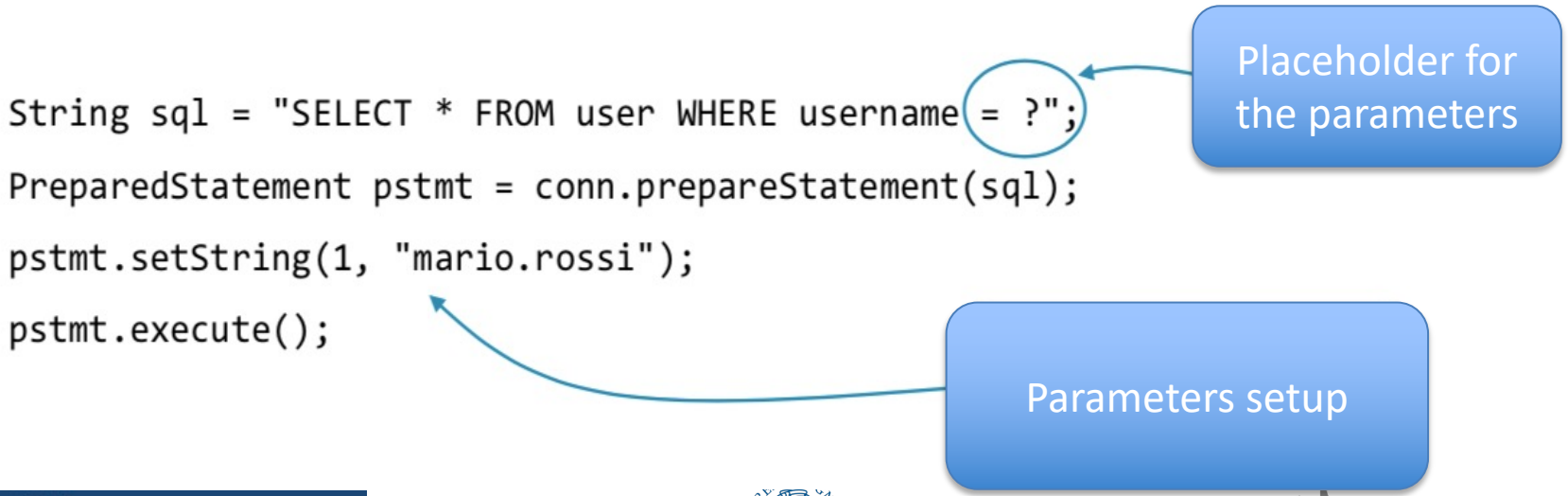
Also the ResultSet and the statement objects must be closed.

Prepared Statements

Using the interface ***java.sql.Statement*** the SQL statement is compiled at the execution time.

Using prepared statements allows us to ***pre-fill*** a statement and improves performance when the same query is ***executed multiple times***, even with different parameters.

The parameters are indicated by question marks and set with appropriate setters.



Updating Tables

INSERT, ***UPDATE*** and ***DELETE*** statements do not return a ResultSet object.

The ***executeUpdate()*** method executes the statement and returns the number of rows actually inserted/modified/deleted.

```
String sql = "INSERT INTO user SET username = ?, first_name = ?, last_name = ?, "
                   + "email = ?, password = SHA1(?)";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "alice.ferrari");
pstmt.setString(2, "Alice");
pstmt.setString(3, "Ferrari");
pstmt.setString(4, "alice.ferrari@example.com");
pstmt.setString(5, "1234");

int insertedRows = pstmt.executeUpdate();
System.out.println("Rows inserted: " + insertedRows);
```

Executes the statement and returns the number of inserted rows

Maven

"Apache Maven is a ***software project management*** and comprehension tool. Based on the concept of a project object model (***POM***).

Maven can manage a project's ***build***, ***reporting*** and ***documentation*** from a central piece of information." extracted from https://maven.apache.org/

We will use Maven for managing the dependencies in our JAVA programs.

Roughly speaking, we will avoid to manually add libraries and API.

POM file

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XML
  <modelVersion>4.0.0</modelVersion>
  <groupId>exercises</groupId>
  <artifactId>exercises</artifactId>
  <version>0.0.1-SNAPSHOT</version>
   <dependencies>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
      <version>8.0.17</version>
</dependency>
 </dependencies>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

It is an XML file that contains information about the project and configuration details used by Maven to build the project.

It contains default values for most projects.

Whenever we need a new dependency, we add new fields to the ***dependencies list***.

groupId uniquely identifies a project across all projects.

artifactId is the name of the jar without version.

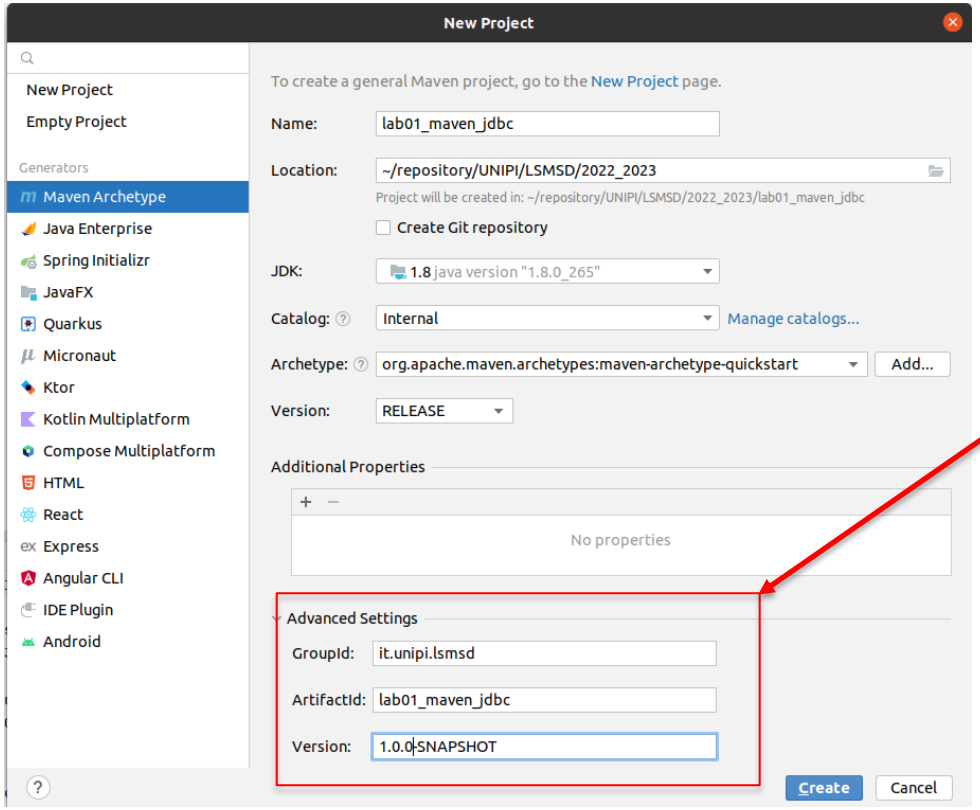
By default, By default, Maven will download from the central repository.

Creating a Maven Project (Using Eclipse)

1. Create a New Java Project
2. Create your first Package
3. Create your first Class (main class)
4. Right Click on your Project
5. Select Configure-> Convert to Maven Project
6. Add your dependencies
7. Right Click on your Project
8. Select Maven->Update Project

Creating a Maven Project (Using IntelliJ)

1. File -> New Project
2. Enter/choose the values displayed in the next screenshot

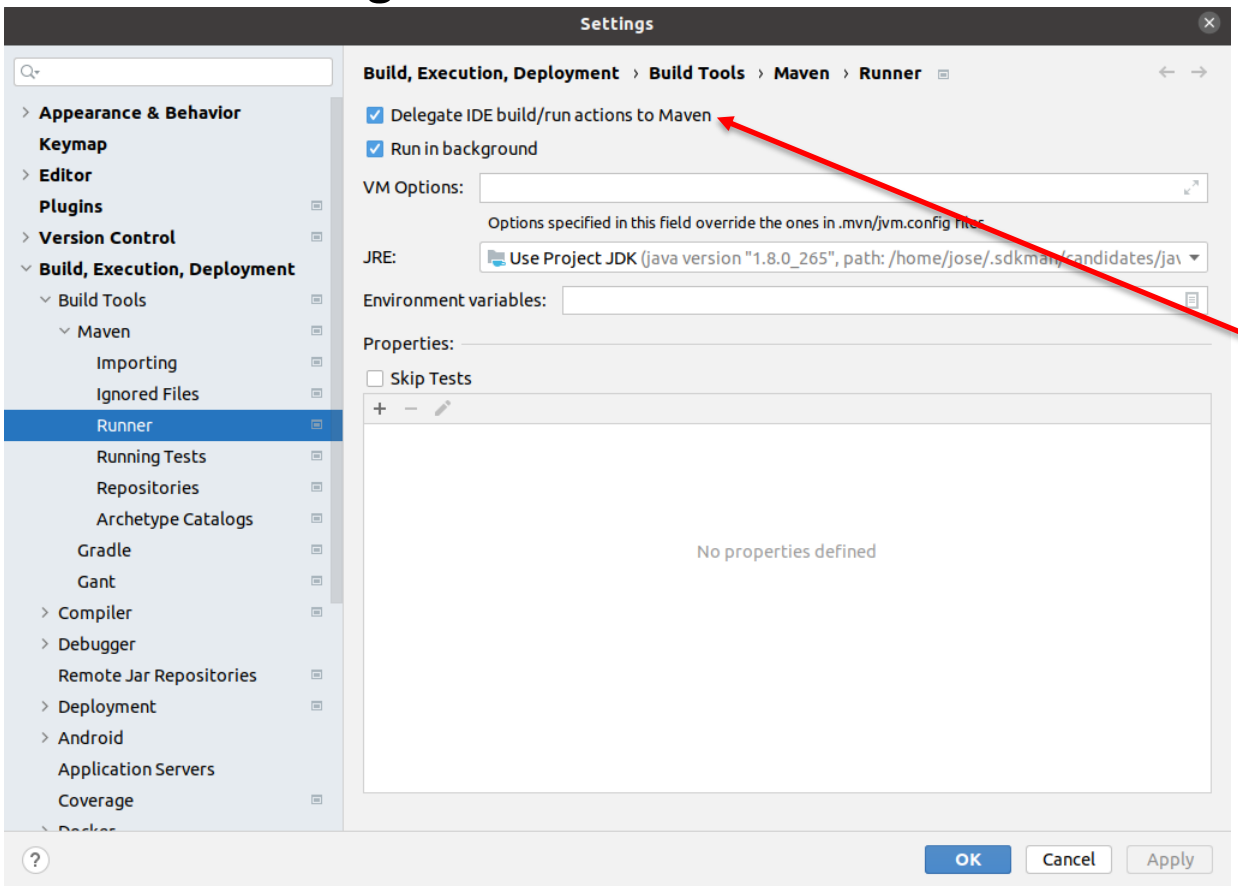


- Coordinates are made up of GroupId, ArtifactId and Version (Maven trinity!).
- Coordinates define the unique place of the project in the Maven universe.

Creating a Maven Project (Using IntelliJ)

Remember to delegate the build/run actions to Maven:

1. File -> Settings:



Check on this option.

Setting up the pom.xml file

For this lab session, we would like to interact with a MySQL database so it is required to add the MySQL JDBC Driver.

pom.xml (lab01_maven_jdbc)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>it.unipi.lsmsd</groupId>
  <artifactId>lab01_maven_jdbc</artifactId>
  <version>1.0.0-SNAPSHOT</version>

  <name>lab01_maven_jdbc</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.30</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>${artifactId}</finalName>
  </build>
</project>
```

First program with JDBC – Example 01

```java
package it.unipi.lsmsd;

import java.sql.*;

public class Example01 {

    public static void main(String[] args) {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try{
            connection = DriverManager.getConnection( url: "jdbc:mysql://localhost:3306/unipi?" +
                    "zeroDateTimeBehavior=CONVERT_TO_NULL&serverTimeZone=CET", user: "jose", password: "jose");
            statement = connection.createStatement();
            resultSet = statement.executeQuery( sql: "select id, name from employee order by name asc");
            while (resultSet.next()) {
                Integer id = resultSet.getInt( columnLabel: "id");
                String name = resultSet.getString( columnLabel: "name");
                System.out.format("%d) %s \n", id, name);
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }  finally{
            if (resultSet != null) {
                try { resultSet.close();} catch (SQLException e) {}
            }
            if (statement != null) {
                try { statement.close();} catch (SQLException e) {}
            }
            if (connection != null) {
                try { connection.close();} catch (SQLException e) {}
            }
        }
    }
}
```

Too many lines of code to close these objects!!!

Let's organize the code.

First program with JDBC – Example 01

```java
package it.unipi.lsmsd;

import java.sql.*;
import java.util.Properties;

public class Example02 {
    private static final String MYSQL_HOST = "localhost";
    private static final Integer MYSQL_PORT = 3306;
    private static final String MYSQL_DATABASE = "unipi";
    private static final String MYSQL_USERNAME = "jose";
    private static final String MYSQL_PASSWORD = "jose";
    // format: mysql://<username>:<password>@<host>:<port>/<db_name>
    private static final String JDBC_URL = "jdbc:mysql://%s:%s@%s:%d/%s";

    public static void main(String[] args) {
        String jdbcUrl = String.format(JDBC_URL, MYSQL_USERNAME, MYSQL_PASSWORD, MYSQL_HOST, MYSQL_PORT, MYSQL_DATABASE);
        Properties properties = new Properties();
        properties.put("zeroDateTimeBehavior", "CONVERT_TO_NULL");
        properties.put("serverTimeZone", "CET");
        try(
                Connection connection = DriverManager.getConnection(jdbcUrl, properties);
                Statement    statement = connection.createStatement();
                ResultSet    resultSet = statement.executeQuery( sql: "select id, name from employee order by name asc");
          ){
            while (resultSet.next()) {
                Integer id = resultSet.getInt( columnLabel: "id");
                String name = resultSet.getString( columnLabel: "name");
                System.out.format("%d) %s \n", id, name);
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Try-with-resources statement.

For more information: https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Suggested Readings

https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-usagenotes-basic.html

https://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html

From https://mvnrepository.com/ the dependencies for several tools maybe retrieved and simply added to our JAVA project.

Exercise 1

From the command line:

1. Run the MySQL Server
2. Create a database
3. Create a table “company” with the following Attributes: name, address, employee_count and website.

From the Java Application Side, write a program that:

1. Makes an access to the database
1. Inserts 3 tuples , such as <Unipi, Lungarno Galilei, 5000, www.unipi.it>.
2. Retrieves all the tuples of the table “company” and prints them on the screen
3. Deletes all the inserted tuples.

Exercise 2: Book Shop Application (I)

Implement and test the *"**Book Shop Application**"*

Description:
The application must be used by the ***employees*** of a book shop, mainly for granting a complete and efficient tracing of the ***books*** stored in the book shop.

For each book, the following information are stored: title, author, publisher, price, number of pages, category, year of publication and number of books in stock. The application also maintains some information regarding the ***author(s)*** (first name, last name) and the ***publisher*** (name and location) to allow the salesman to query books, authors and/or publishers.

Exercise: Book Shop Application (II)

Main requirements:

The application allows the user to:

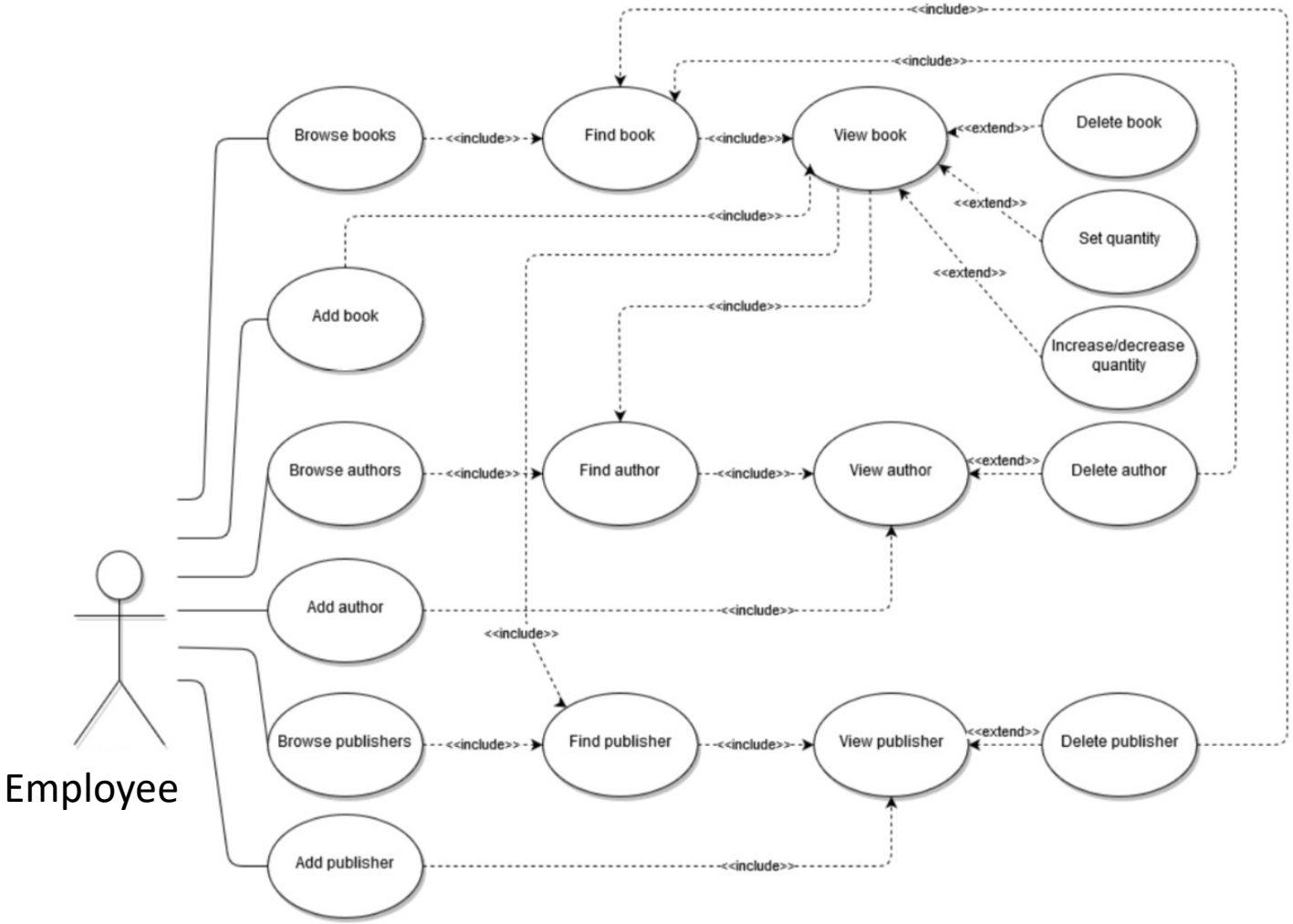
- Read the list of the books
- Insert a new book
- Remove a book
- Update the quantity of a book
- Increase/decrease the quantity of a book
- Read the list of the authors (for each author you must display the number of books)
- Insert an author
- Delete an author
- Read the list of the publishers
- Insert a publisher;
- Delete a publisher;

Some constrains:

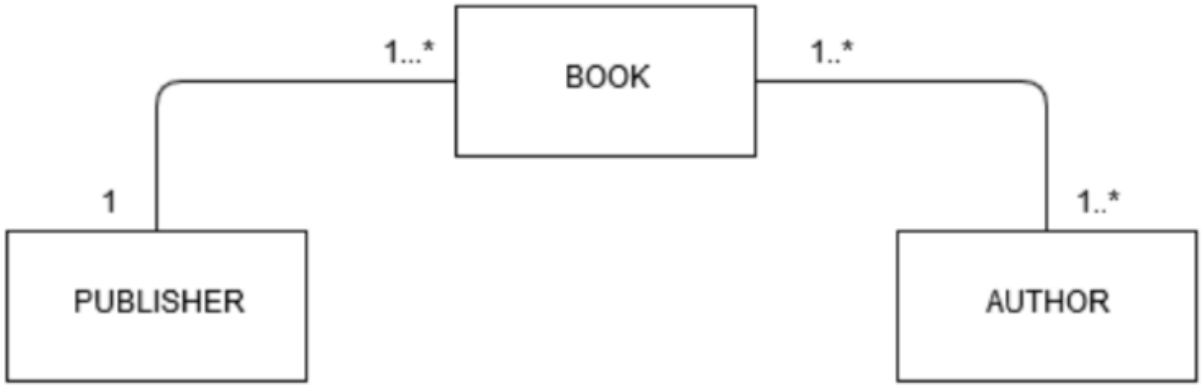
if an author, or a publisher, will be removed from the DB, also all the books associated with it must be removed.

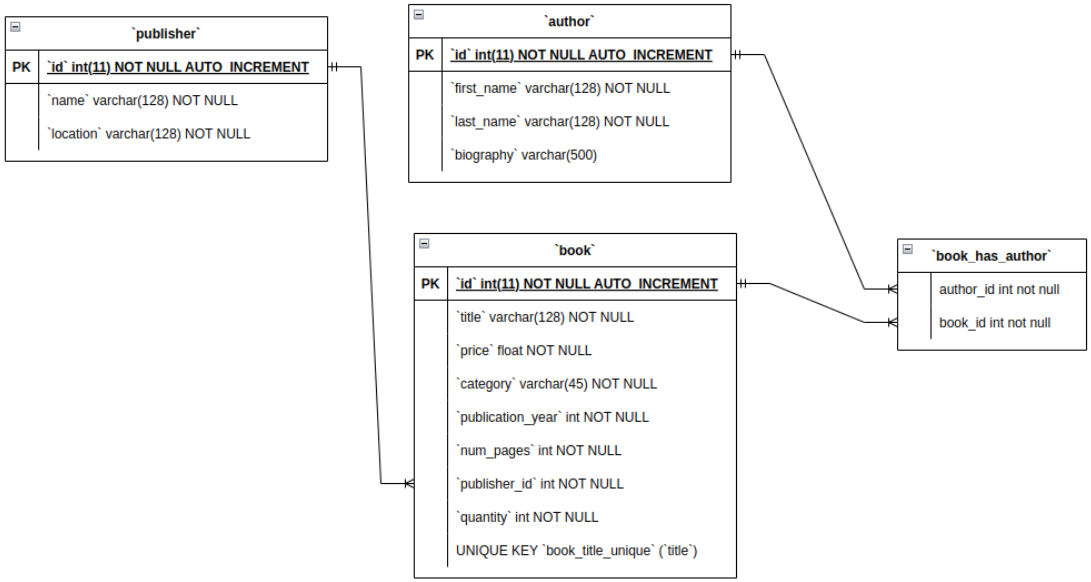
If a book is selected (viewed) also the publisher and the author(s) must be shown.

Use Cases Diagram



UML Class Analysis and DB Scheme





Acknowledgements

Credits to:

- Davide Coccomini
- Luigi Gjoni
- Marilisa Lippini
- Lorenzo Nelli
- Federico Fregosi
- Mirko Laruina
- Riccardo Mancini
- Gianmarco Petrelli

Students of 2019-2020 Academic Year.

They designed and developed the two applications discussed in the proposed exercises for their TASK1.

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0