Welcome to the

Java Course

Module 4 – Day 03

Content of the course

- Introduction to Database Theory and SQL Basics
- Table Management and Relationships
- **Data Normalization**
- Advanced SQL Queries and Integration with Java
- Final Project and Optimization Techniques

Event Manager Project - Step 2

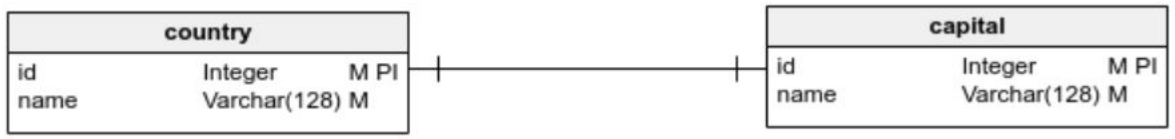
- **Write SQL statements** to create the database and tables necessary to store the Event Manager data as planned in your drawing of the previous step. Execute these SQL statements using pgAdmin.

- **Populate the database** with made up data using SQL INSERT statements in pgAdmin.

Relationships

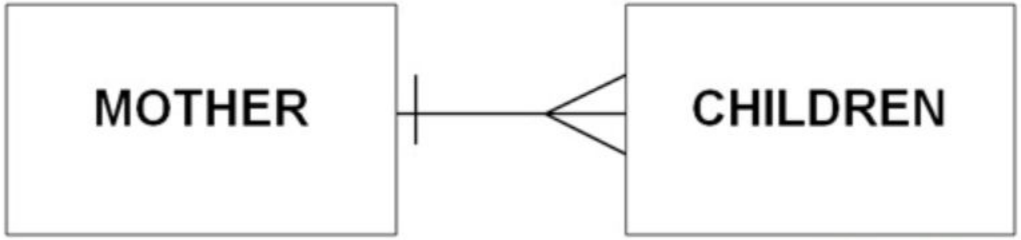
One-to-One Relationship

In a **one-to-one relationship**, each record in one table is related to exactly one record in another table, and vice versa.



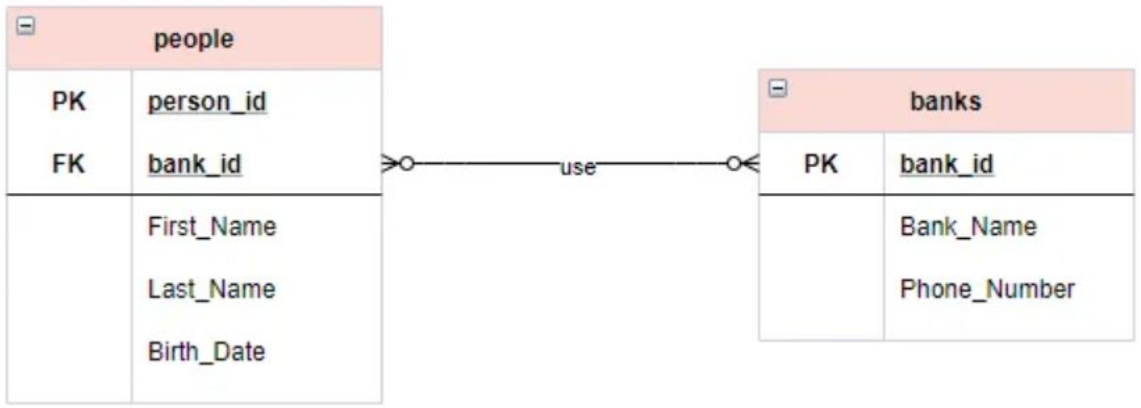
One-to-Many Relationship

In a **one-to-many relationship**, each record in one table can be related to one or more records in another table, but each record in the second table is related to only one record in the first table.



Many-to-Many Relationship

In a **many-to-many relationship**, each record in one table can be related to one or more records in another table, and vice versa.



Now YOUR TURN !

Grab your phone!

JPA

Java Persistence API

JPA

Java Persistence API is a specification that defines a standard interface for Java applications to interact with relational databases.

It simplifies database access by providing a set of annotations and APIs for mapping Java objects to database tables.

JPA is not an implementation but rather a set of rules that frameworks, like Hibernate, implement.

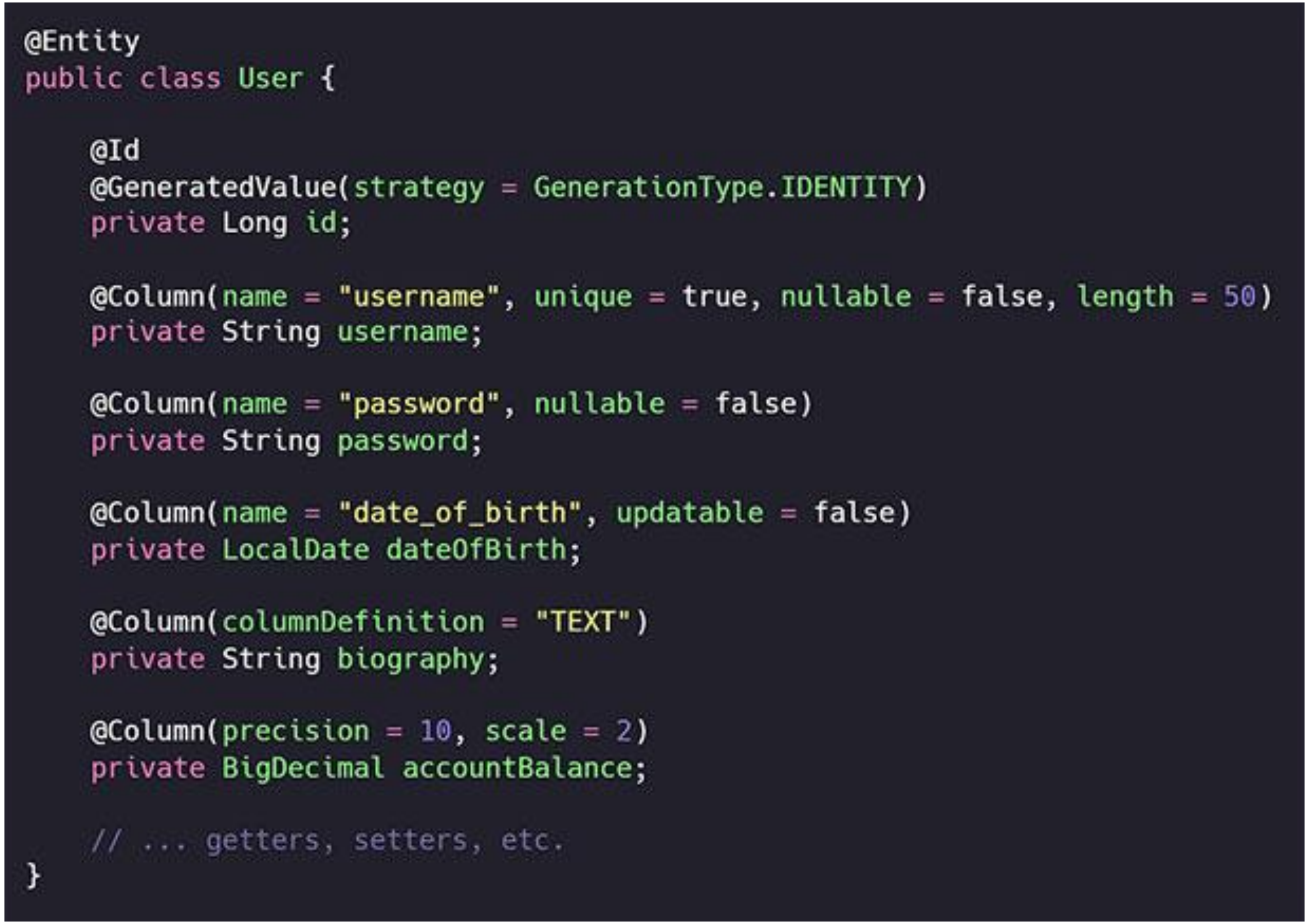
JPA vs JDBC

JDBC (Java Database Connectivity) is the standard Java API for connecting to relational databases.

Unlike JPA, JDBC is lower level and requires developers to write SQL queries and handle result sets manually.

While it offers more control, it also demands more code. JDBC is often used when fine-grained control over database interactions is required

JPA annotations



```java
@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "username", unique = true, nullable = false, length = 50)
    private String username;

    @Column(name = "password", nullable = false)
    private String password;

    @Column(name = "date_of_birth", updatable = false)
    private LocalDate dateOfBirth;

    @Column(columnDefinition = "TEXT")
    private String biography;

    @Column(precision = 10, scale = 2)
    private BigDecimal accountBalance;

    // ... getters, setters, etc.
}
```

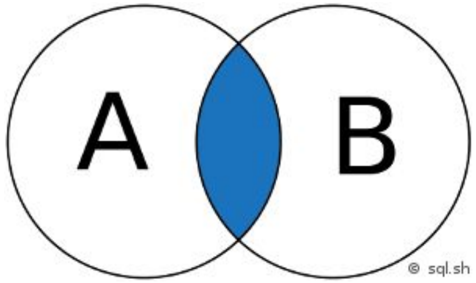
Many-to-One Relationship using JPA

```
@Entity
public class Comment {
  @Id
  private Long id;

  // Other fields

  @ManyToOne
  @JoinColumn(name = "blog_post_id")
  private BlogPost blogPost;

  // Getters and setters
}
```

Joins

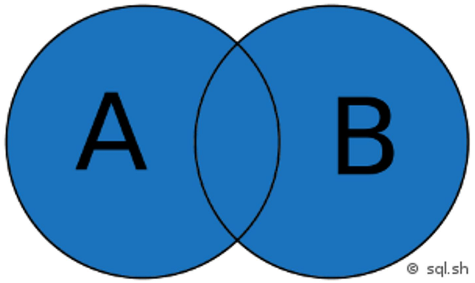
Joining data

INNER Join



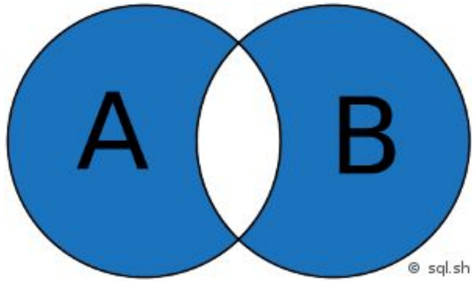
```
SELECT *
FROM A
INNER JOIN B ON A.key = B.key
```

FULL OUTER Join



```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
```

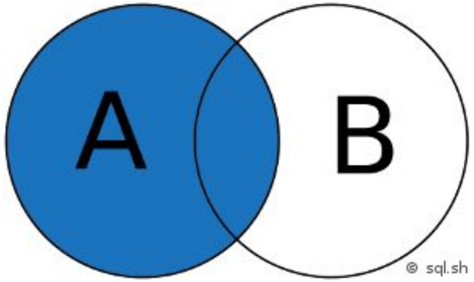
FULL OUTER Join without intersection



```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

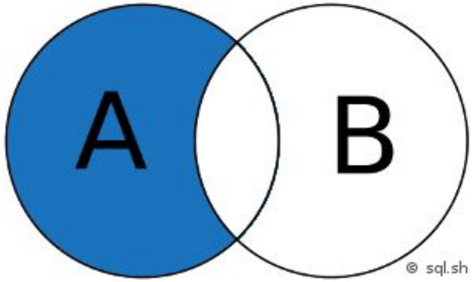
Joining data

LEFT OUTER Join



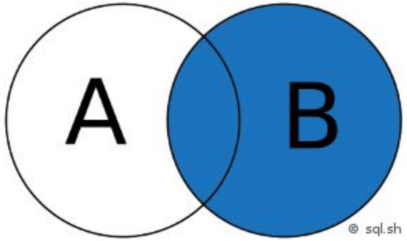
```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

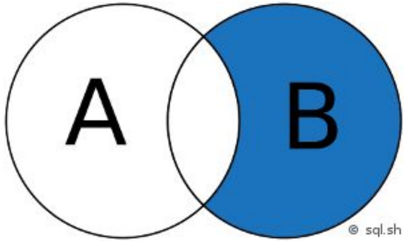
LEFT Join OUTER without **B**



```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

RIGHT OUTER Join





Joining data - example

Find all customers that live in Abu Dhabi

```
SELECT c.first_name, c.last_name
FROM customer c
JOIN address a ON c.address_id = a.address_id
JOIN city ct ON a.city_id = ct.city_id
WHERE ct.city = 'Abu Dhabi';
```

Now YOUR TURN !

Let’s do the exercises

Event Manager Project - Step 3

- Create a Java project for the Event Manager
- Create Java classes for the tables you created in the database
- Create a main program that will allow the user to view all existing events