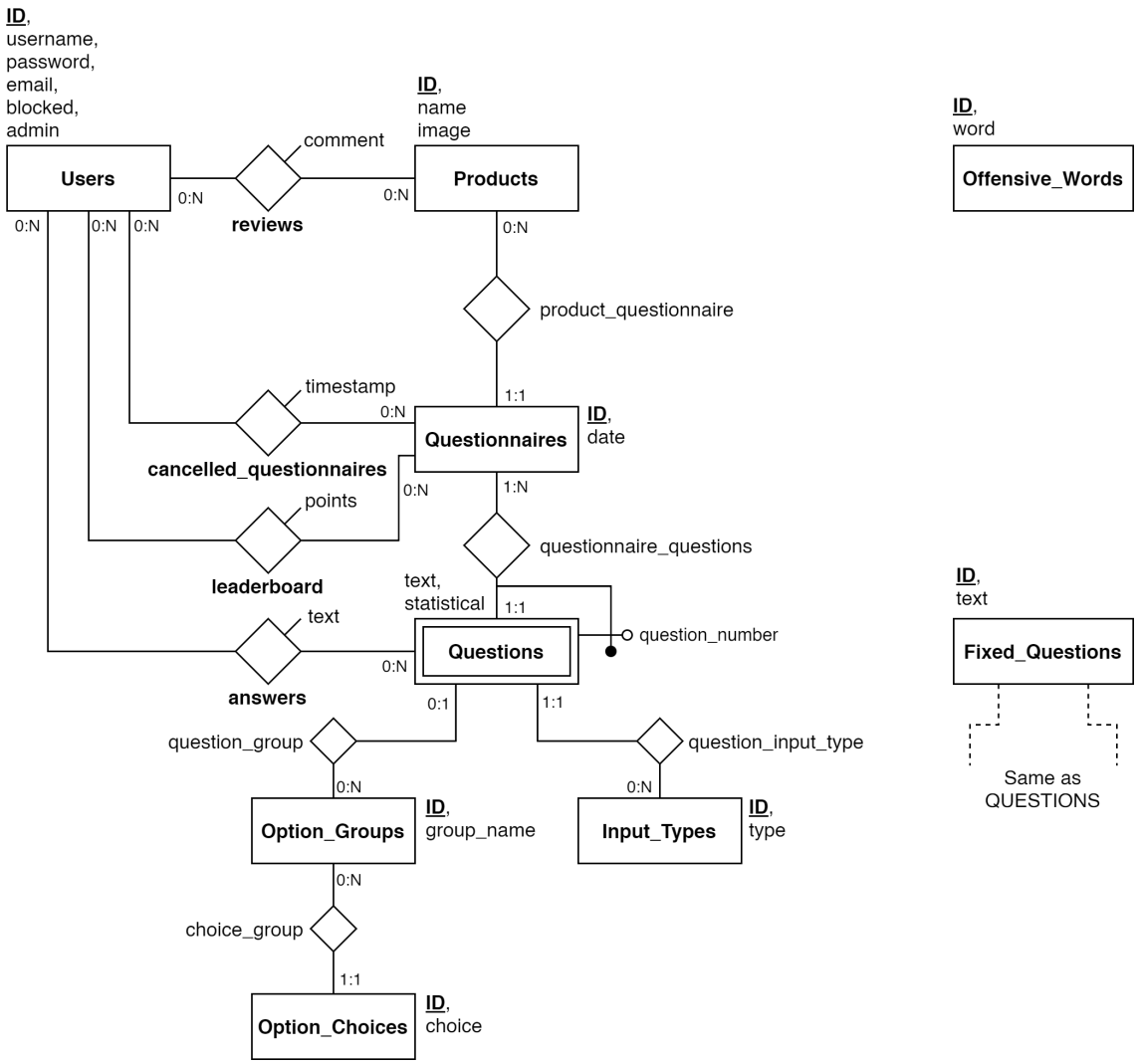
Data Bases 2

Optional Project

Lampis Andrea 10622804

Entity Relationship



Relational model

Users(ID, username, password, email, blocked, admin)

Reviews(user_id, product_id, comment)

Products(ID, name, image)

Cancelled_questionnaires(questionnaire_id, user_id, timestamp)

Leaderboard(questionnaire_id, user_id, points)

Questionnaires(ID, date, product)

Questions(questionnaire_id, question_number, text, is_statistical,
input_type_id, option_group_id)

Answers(user_id, questionnaire_id, question_number, text)

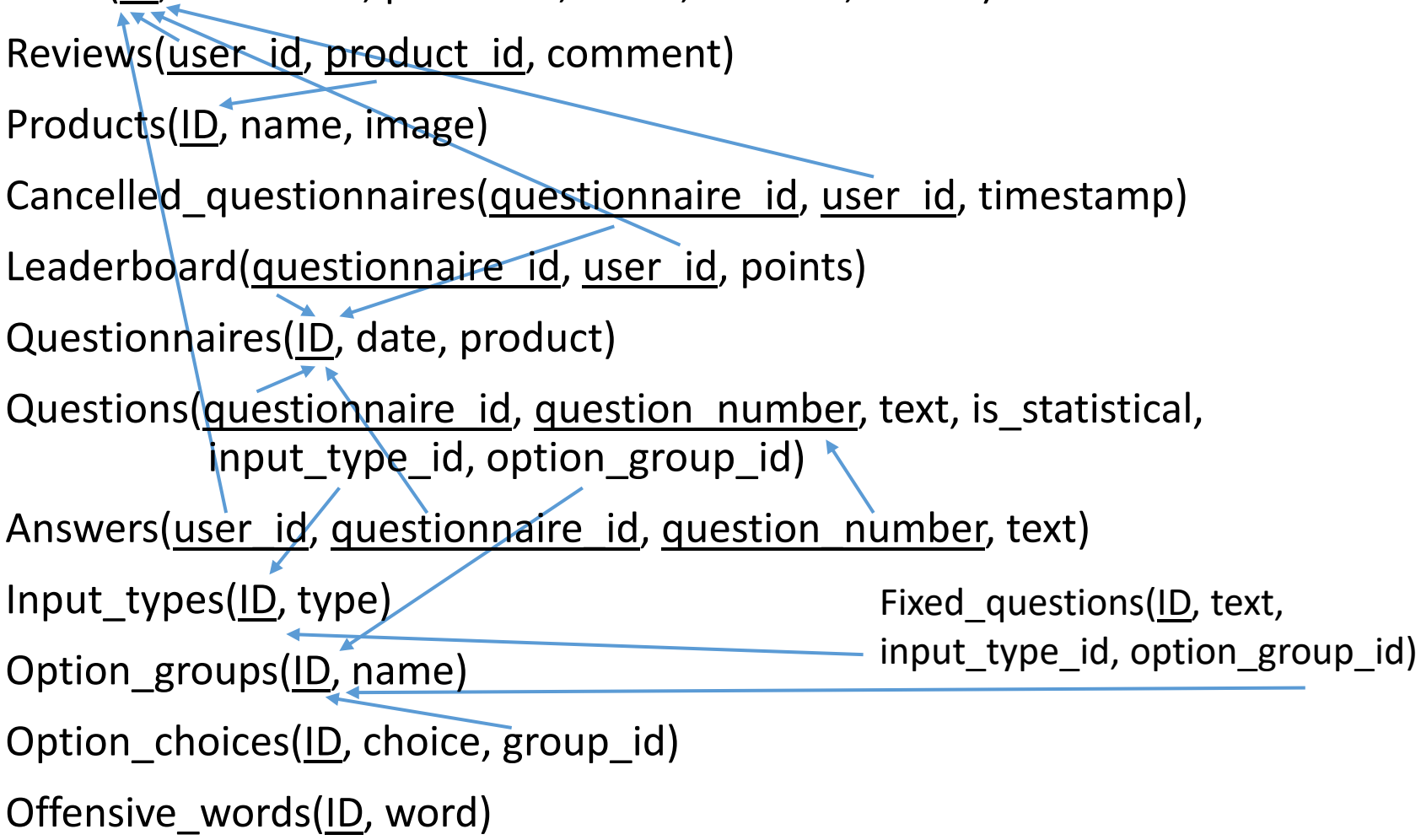
Input_types(ID, type)

Option_groups(ID, name)

Option_choices(ID, choice, group_id)

Offensive_words(ID, word)

Fixed_questions(ID, text,
input_type_id, option_group_id)



Motivation

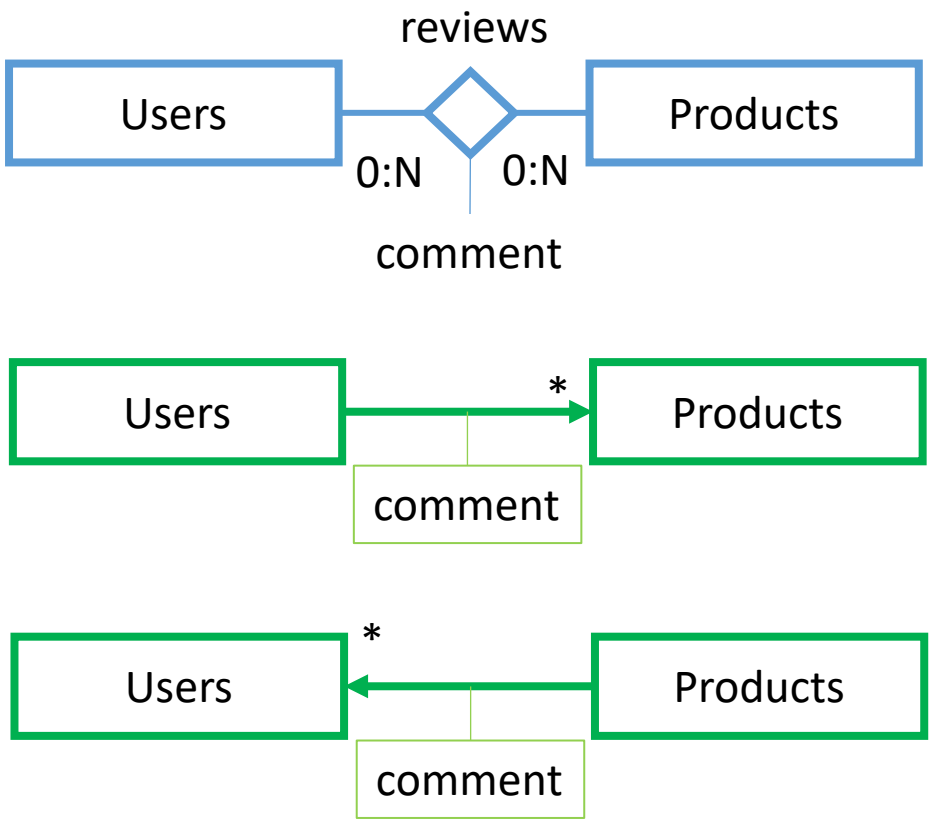
The statistical questions have been implemented through a dedicated table “Fixed_questions”.

Each time a questionnaire is created, those fixed questions are copied into the “Questions” table (through JPA code).

In this way, we can:

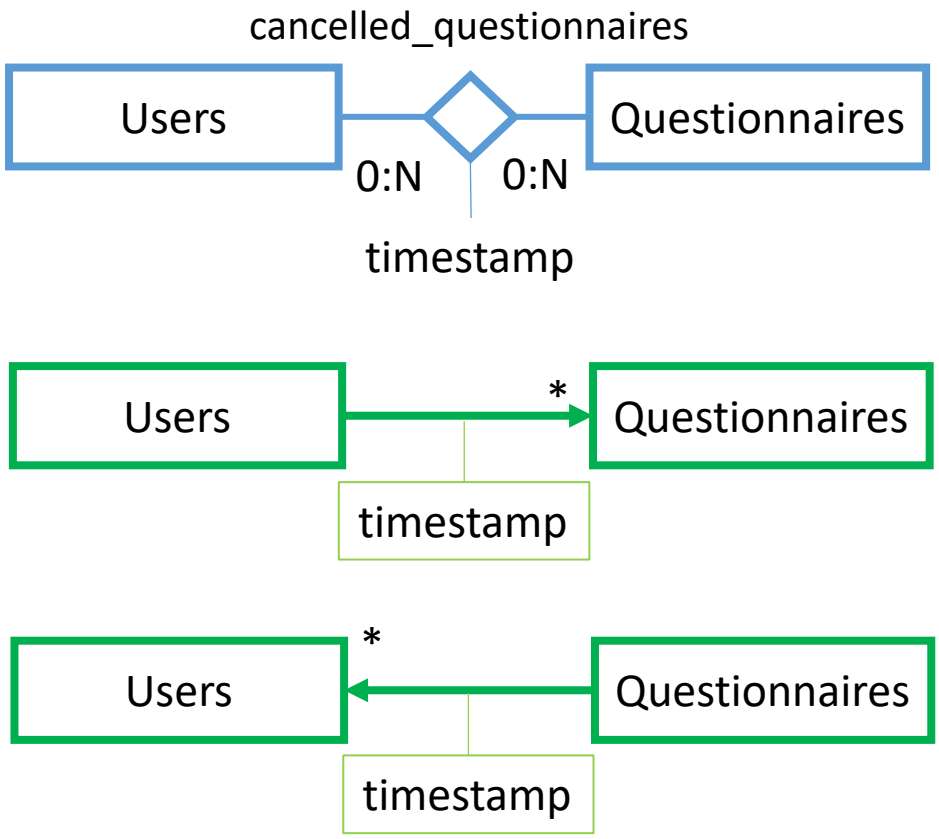
1) Add at any time new statistical questions simply adding a new tuple in the “Fixed_questions” table
2) Treat “Fixed_questions” as normal Questions in our backend code

Relationship “reviews”



- Users → Products
  @ElementCollection or @ManyToMany, not necessary but implemented for future uses
  - FetchType.LAZY
- Products → Users
  @ManyToMany, @ElementCollection necessary to show the reviews of a product with their owner
  - FetchType.EAGER to navigate the relationship at client side

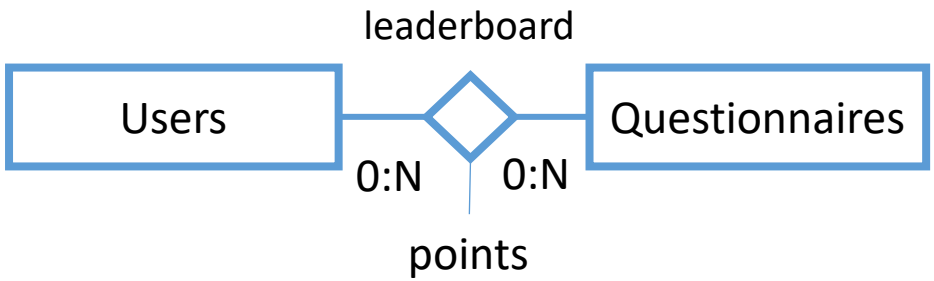
Relationship “cancelled_questionnaires”



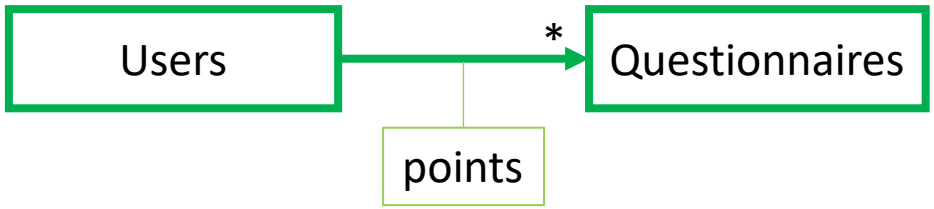
- Users → Questionnaires
  @ElementCollection or @ManyToMany, not necessary

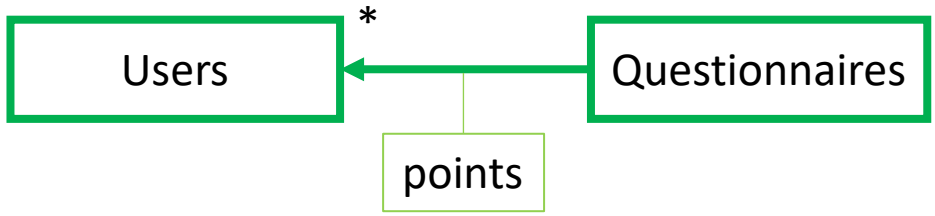
- Questionnaires → Users
  @ManyToMany, @ElementCollection necessary to show to the admin the users who cancelled a questionnaire, with the cancellation timestamp

Relationship “leaderboard”



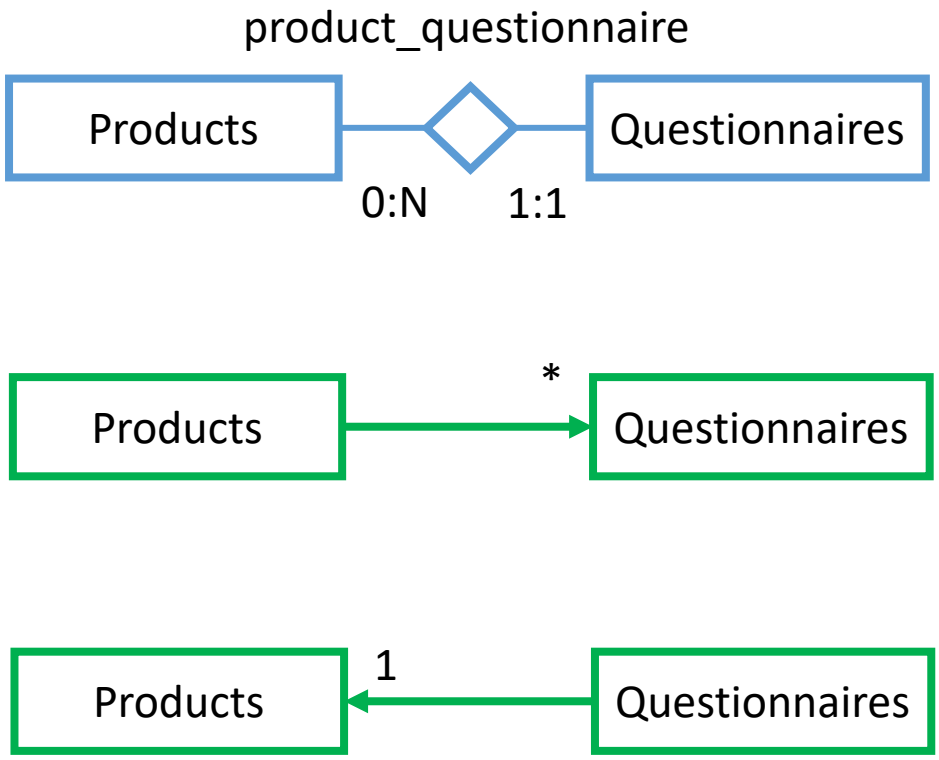
- Users → Questionnaires
  @ElementCollection or @ManyToMany, not necessary





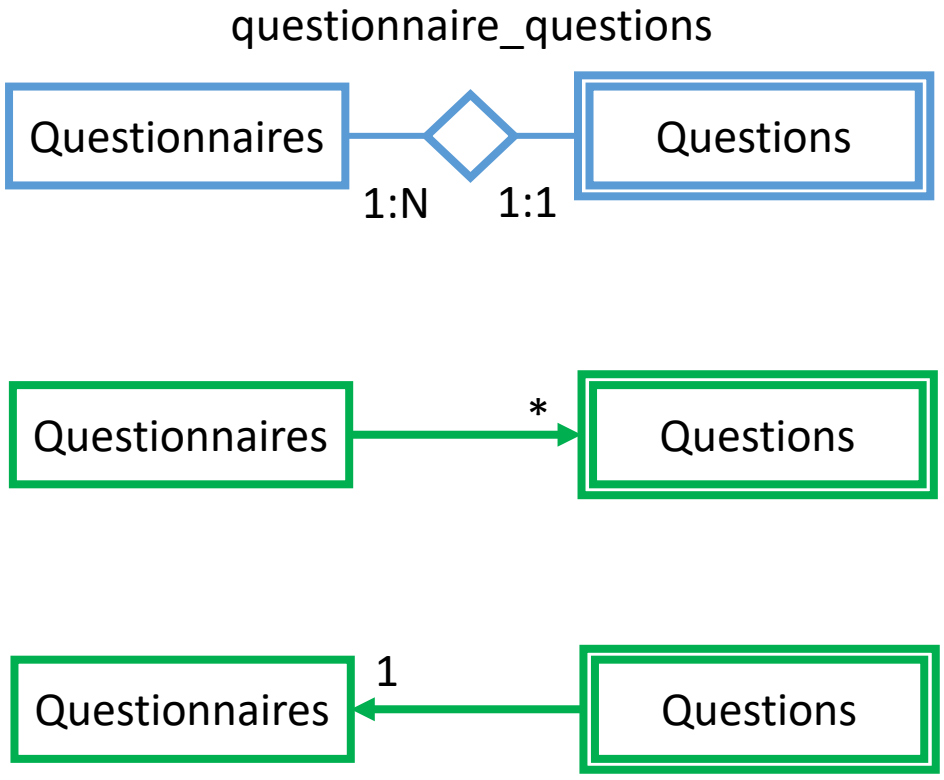
- Questionnaires → Users
  @ManyToMany, @ElementCollection necessary to show the leaderboard of a given questionnaire, ordered by the points in descending order (and also in the admin inspection page)

Relationship “product_questionnaire”



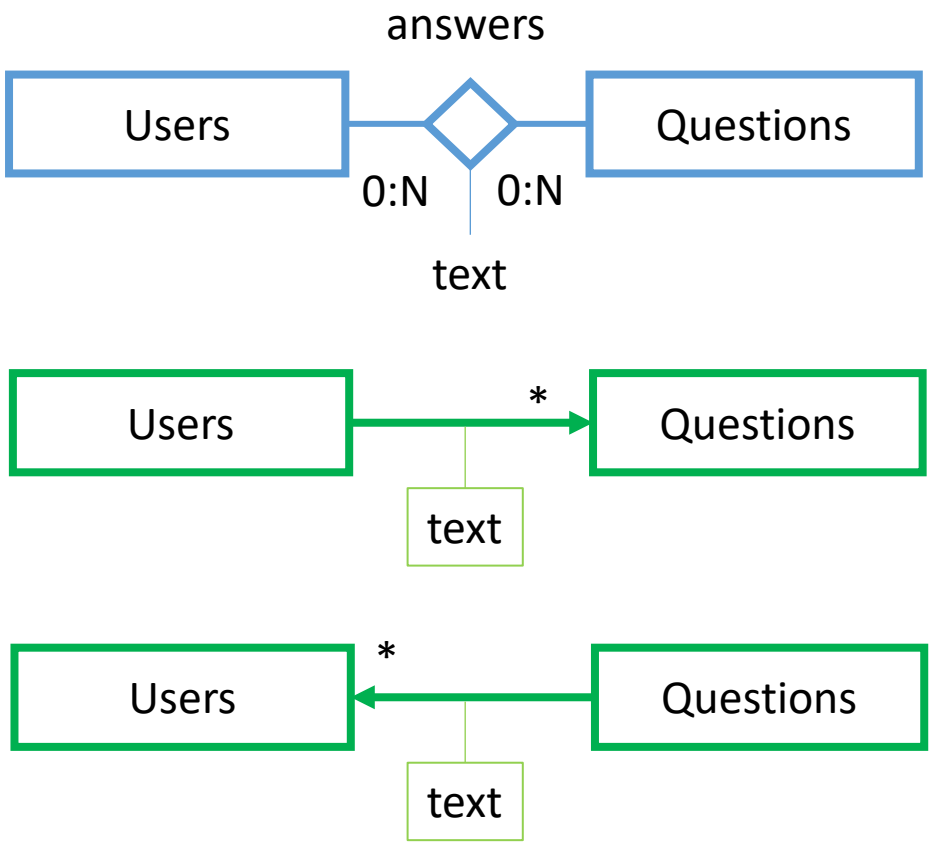
- Products → Questionnaires @OneToMany not necessary but implemented for future uses
  - FetchType.LAZY
- Questionnaires → Products @ManyToOne, necessary to show the product of the questionnaire of the day.
  - Owner side
  - FetchType.EAGER to navigate the relationship at client side

Relationship “questionnaire_questions”



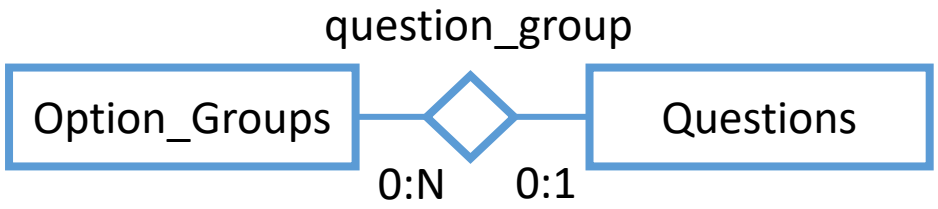
- Questionnaires → Questions @OneToMany necessary to retrieve the questions of a questionnaire
  - FetchType.LAZY because the questions are needed only in the compilation form
- Questions → Questionnaires @ManyToOne, implemented due to weak entity
  - Owner side

Relationship “answers”



- User → Questions
  @ElementCollection or
  @ManyToMany, not necessary
- Questions → Users
  @ManyToMany,
  @ElementCollection

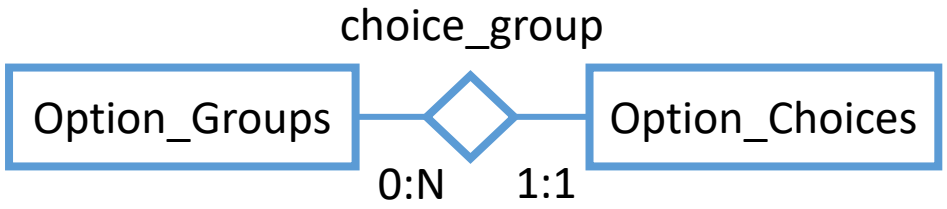
Relationship “question_group”



- Option_Groups → Questions
  @OneToMany, not necessary

- Questions → Option_Groups
  @ManyToOne, necessary to retrieve the option_group of a given question
  - Owner side

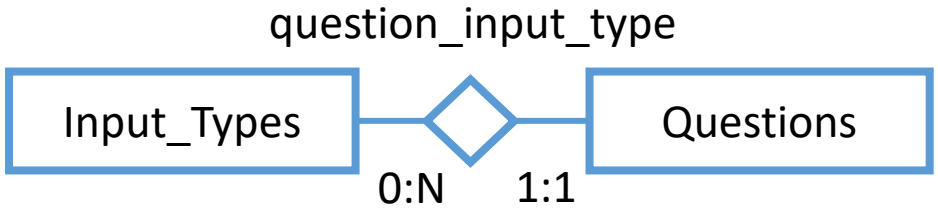
Relationship “choice_group”



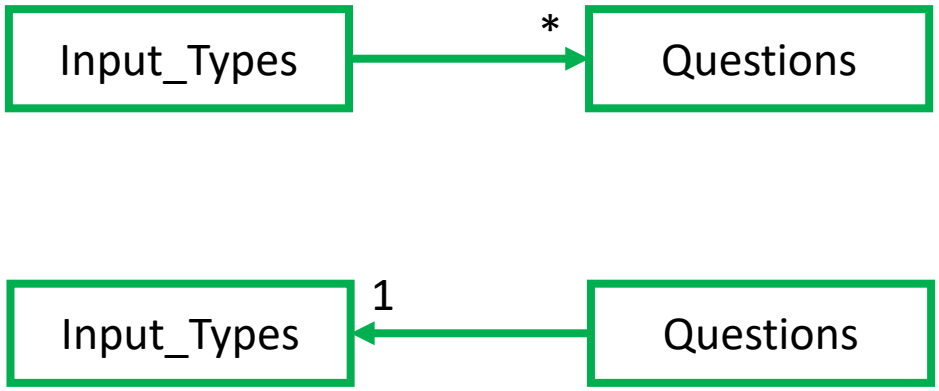
- Option_Groups → Option_Choices
  @OneToMany, necessary to retrieve the possible choices of a given group

- Option_Choices → Option_Groups
  @ManyToOne, not necessary but implemented for simplicity
  - Owner side

Relationship “question_input_type”



- Input_Types → Questions
  @OneToMany, not necessary



- Questions → Input_Types
  @ManyToOne, necessary to retrieve the input_type of a given question
  - Owner side

Motivations

All the N:N relationships have been implemented through a “support” entity, which uses a composite primary key.
In this way we can add more than one attribute to the relationship and also, from a coding perspective, the naming convention is more clear.

With this implementation choice, the two original related entities will have a @OneToMany mapping, while the new support entity will have two @ManyToOne mappings.

Entity User

```
@Entity
@Table(name = "users", schema = "db_gma")
@NamedQueries({
      @NamedQuery(name = "User.checkCredentials",
            query = "SELECT r FROM User r WHERE r.username = ?1 and r.password = ?2"),
      @NamedQuery(name = "User.findUserByUsername", query = "SELECT u FROM User u WHERE u.username = ?1"),
      @NamedQuery(name = "User.findUserByEmail", query = "SELECT u FROM User u WHERE u.email = ?1")
})

public class User implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "iduser")
   private int id;

   private String username;
   private String password;
   private String email;
   private boolean blocked;
   private boolean admin;

   @OneToMany(fetch = FetchType.LAZY, mappedBy = "user", cascade = CascadeType.ALL)
   private Set<Review> reviews = new HashSet<>();

// constructor, setters and getters omitted...
```

Entity Product

```
@Entity
@Table(name = "products", schema = "db_gma")
@NamedQuery(name = "Product.getAllProducts", query = "SELECT p FROM Product p")
public class Product implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "idproduct")
   private int id;

   private String name;

   @Basic(fetch = FetchType.LAZY)
   @Lob
   private byte[] image;

   @OneToMany(fetch = FetchType.LAZY, mappedBy = "product", cascade = CascadeType.ALL)
   private List<Questionnaire> questionnaires = new ArrayList<>();

   @OneToMany(fetch = FetchType.EAGER, mappedBy = "product", cascade = CascadeType.ALL)
   private Set<Review> reviews = new HashSet<>();

// constructor, setters and getters omitted...
```

Entity Review

```java
@Entity
@Table(name = "reviews", schema = "db_gma")
public class Review implements Serializable {
   private static final long serialVersionUID = 1L;

   @EmbeddedId
   private ReviewPK id;

   @ManyToOne
   @MapsId("userId")
   @JoinColumn(name = "user_id", referencedColumnName = "iduser", nullable = false)
   private User user;

   @ManyToOne
   @MapsId("productId")
   @JoinColumn(name = "product_id", referencedColumnName = "idproduct", nullable = false)
   private Product product;

   private String comment;

// constructor, setters and getters omitted...
```

Entity Questionnaire

```java
@Entity
@Table(name = "questionnaires", schema = "db_gma")
@NamedQueries({
      @NamedQuery(name = "Questionnaire.findQuestionnaireOfTheDay", query = "SELECT q FROM Questionnaire q WHERE q.date = CURRENT_DATE"),
      @NamedQuery(name = "Questionnaire.findQuestionnaireByDay", query = "SELECT q FROM Questionnaire q WHERE q.date = ?1"),
      @NamedQuery(name = "Questionnaire.findAllPastQuestionnaires",query = "SELECT q FROM Questionnaire q WHERE q.date < CURRENT_DATE ORDER BY q.date DESC")
})
public class Questionnaire implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "idquestionnaire")
   private int id;

   @Temporal(TemporalType.DATE)
   private Date date;

   @ManyToOne
   @JoinColumn(name = "product", referencedColumnName = "idproduct", nullable = false)
   private Product product;

   @OneToMany(mappedBy = "questionnaire", cascade = {CascadeType.PERSIST, CascadeType.REMOVE, CascadeType.MERGE}, orphanRemoval = true)
   @OrderBy("questionNumber ASC")
   private List<Question> questions = new ArrayList<>();

   @OneToMany(fetch = FetchType.EAGER, mappedBy = "questionnaire", cascade = {CascadeType.REMOVE, CascadeType.REFRESH})
   @OrderBy("points DESC")
   private List<Leaderboard> leaderboard = new ArrayList<>();

   @OneToMany(fetch = FetchType.EAGER, mappedBy = "questionnaire", cascade = {CascadeType.REMOVE, CascadeType.REFRESH})
   @OrderBy("log_timestamp DESC")
   private List<CancelledQuestionnaire> cancelledQuestionnaires = new ArrayList<>();

   public Questionnaire(Product product, Date date, List<Question> questions) {
      this.product = product;
      this.date = date;

      int questionNumber = 1;
      for (Question q : questions) {
         q.setPK(this, questionNumber);
         questionNumber++;
      }

      this.questions = questions;
   }

// setters and getters omitted...
```

Entity Question

```java
@Entity
@Table(name = "questions", schema = "db_gma")
public class Question {
   private static final long serialVersionUID = 1L;

   @EmbeddedId
   private QuestionPK id;

   @ManyToOne
   @MapsId("questionnaireId")
   @JoinColumn(name = "idquestionnaire", referencedColumnName = "idquestionnaire", nullable = false)
   private Questionnaire questionnaire;

   @MapsId("questionNumber")
   @Column(name = "question_number", nullable = false, insertable = false, updatable = false)
   private int questionNumber;

   private String text;

   @Column(name = "is_statistical")
   private boolean isStatistical;

   @ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.REFRESH})
   @JoinColumn(name = "option_group_id", referencedColumnName = "idoption")
   private OptionGroup optionGroup;

   @ManyToOne
   @JoinColumn(name = "input_type_id", referencedColumnName = "idinput", nullable = false)
   private InputType inputType;

   @OneToMany(fetch = FetchType.LAZY, cascade = {CascadeType.PERSIST, CascadeType.REMOVE, CascadeType.REFRESH, CascadeType.MERGE})
   @JoinColumns({
         @JoinColumn(name = "idquestionnaire", referencedColumnName = "idquestionnaire", insertable = false, updatable = false),
         @JoinColumn(name = "question_number", referencedColumnName = "question_number", insertable = false, updatable = false),
   })
   private Set<Answer> answers = new HashSet<>();

// constructor, setters and getters omitted...
```

Entity FixedQuestion

```java
@Entity
@Table(name = "fixed_questions", schema = "db_gma")
@NamedQuery(name = "FixedQuestion.getAllFixedQuestions", query = "SELECT q FROM FixedQuestion q")
public class FixedQuestion {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "idquestion")
   private int id;

   private String text;

   @ManyToOne
   @JoinColumn(name = "option_group_id", referencedColumnName = "idoption")
   private OptionGroup optionGroup;

   @ManyToOne
   @JoinColumn(name = "input_type_id", referencedColumnName = "idinput", nullable = false)
   private InputType inputType;

// constructor, setters and getters omitted...
```

Entity Answer

```
@Entity
@Table(name = "answers", schema = "db_gma")
@NamedQueries({
      @NamedQuery(name = "Answer.findAnswersForQuestionnaireByUser",
               query = "SELECT a FROM Answer a WHERE a.questionnaire.id = ?1 and a.user.id = ?2"),
      @NamedQuery(name = "Answer.findUsersWhoAnsweredQuestionnaire",
               query = "SELECT DISTINCT a.user FROM Answer a WHERE a.questionnaire.id = ?1"),
})
public class Answer {
   private static final long serialVersionUID = 1L;

   @EmbeddedId
   private AnswerPK id;

   @ManyToOne
   @MapsId("questionnaireId")
   @JoinColumn(name = "idquestionnaire", referencedColumnName = "idquestionnaire", nullable = false, insertable = false, updatable = false)
   private Questionnaire questionnaire;

   @ManyToOne
   @MapsId("userId")
   @JoinColumn(name = "iduser", referencedColumnName = "iduser", nullable = false)
   private User user;

   @MapsId("questionNumber")
   @Column(name = "question_number", nullable = false, insertable = false, updatable = false)
   private int questionNumber;

   private String answer;

   @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
   @JoinColumns({
         @JoinColumn(name = "idquestionnaire", referencedColumnName = "idquestionnaire"),
         @JoinColumn(name = "question_number", referencedColumnName = "question_number")
   })
   private Question question;

// constructor, setters and getters omitted...
```

Entity Leaderboard

```
@Entity
@Table(name = "leaderboard", schema = "db_gma")
@NamedQueries({
      @NamedQuery(name = "Leaderboard.findLeaderboardForQuestionnaire",
            query = "SELECT l FROM Leaderboard l WHERE l.questionnaire.id = ?1 ORDER BY l.points DESC"),
})
public class Leaderboard {
   private static final long serialVersionUID = 1L;

   @EmbeddedId
   private LeaderboardPK id;

   @ManyToOne
   @MapsId("userId")
   @JoinColumn(name = "iduser", referencedColumnName = "iduser", nullable = false)
   private User user;

   @ManyToOne
   @MapsId("questionnaireId")
   @JoinColumn(name = "idquestionnaire", referencedColumnName = "idquestionnaire", nullable = false)
   private Questionnaire questionnaire;

   private int points;

// constructor, setters and getters omitted...
```

Entity CancelledQuestionnaire

```java
@Entity
@Table(name = "cancelled_questionnaires", schema = "db_gma")
@NamedQuery(name = "CancelledQuestionnaire.findCancelledQuestionnairesForQuestionnaire",
            query = "SELECT cq FROM CancelledQuestionnaire cq WHERE cq.questionnaire.id = ?1 ORDER BY cq.log_timestamp DESC")
public class CancelledQuestionnaire {
   private static final long serialVersionUID = 1L;

   @EmbeddedId
   private CancelledQuestionnairePK id;

   @ManyToOne
   @MapsId("userId")
   @JoinColumn(name = "iduser", referencedColumnName = "iduser", nullable = false)
   private User user;

   @ManyToOne
   @MapsId("questionnaireId")
   @JoinColumn(name = "idquestionnaire", referencedColumnName = "idquestionnaire", nullable = false)
   private Questionnaire questionnaire;

   @Temporal(TemporalType.TIMESTAMP)
   private Date log_timestamp;

// constructor, setters and getters omitted...
```

Entity OffensiveWord

```
@Entity
@Table(name = "offensive_words", schema = "db_gma")
@NamedQuery(name = "OffensiveWord.getOffensiveWords", query = "SELECT o FROM OffensiveWord o")
@NamedNativeQuery(name = "OffensiveWord.getBannedWordsInString",
            query = "SELECT * FROM db_gma.offensive_words WHERE MATCH(word) AGAINST(?1 IN NATURAL LANGUAGE MODE)",
            resultClass = OffensiveWord.class)
public class OffensiveWord implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id")
   private int id;

   private String word;

// constructor, setters and getters omitted...
```

Entity InputType

```java
@Entity
@Table(name = "input_types", schema = "db_gma")
@NamedQuery(name = "InputType.getInputTypeByValue", query = "SELECT i FROM InputType i WHERE i.type = ?1")
public class InputType implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "idinput")
   private int id;

   @Enumerated(EnumType.STRING)
   private InputTypeValue type;

// constructor, setters and getters omitted...
```

```java
public enum InputTypeValue {
   select("select"),
   number("number"),
   text("text"),
   textarea("textarea");

   private final String name;

   InputTypeValue(String name) {
      this.name = name;
   }

   public String getName() {
      return name;
   }
}
```

Entity OptionChoice

```
@Entity
@Table(name = "option_choices", schema = "db_gma")
public class OptionChoice implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "idchoice")
   private int id;

   private String choice;

   @ManyToOne
   @JoinColumn(name = "group_id")
   private OptionGroup group;

// constructor, setters and getters omitted...
```

Entity OptionGroup

```
@Entity
@Table(name = "option_groups", schema = "db_gma")
public class OptionGroup implements Serializable {
   private static final long serialVersionUID = 1L;

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "idoption")
   private int id;

   private String name;

   @OneToMany(fetch = FetchType.EAGER, mappedBy = "group", cascade = CascadeType.ALL)
   private List<OptionChoice> choices;

   public OptionGroup(List<OptionChoice> choices) {
      for (OptionChoice c : choices) {
         c.setGroup(this);
      }
      this.choices = choices;
   }

// setters and getters omitted...
```

Clients Components

- Servlets
  - CheckLogin
  - Signup
  - GoToHomePage
  - Leaderboard
  - LoadQuestionnaire
  - SendQuestiionaire
  - Logout

  Admin servlets:
  - GoToAdminHome
  - CreateProduct
  - CreateQuestionnaire
  - CreationPage
  - DeleteQuestionnaire
  - DeletionPage
  - InspectAnswers
  - InspectionPage
  - InspectQuestionnaire

- HTML pages
  - Index.html
  - Home.html
  - Leaderboard.html
  - Questionnaire.html
  - Thanks.html
  - Error.html

  Admin HTML pages:
  - Admin.html
  - CreateProduct.html
  - Creation.html
  - Deletion.html
  - Inspection.html
  - InspectQuestionnaire.html
  - LoadAnswers.html

Business Components

All the components are **stateless** because is not necessary store any data of the session.

- AnswerService
  - public void addAnswers(Questionnaire questionnaire, User user, List<Answer> answers)
  - public Answer createAnswer(Questionnaire questionnaire, User user, int questionNumber, String answer)
  - public List<Answer> getAnswersForQuestionnaireByUser(Questionnaire questionnaire, User user)
- OffensiveWordService
  - public Set<String> getOffensiveWords()
  - public boolean containsOffensiveWord(String text)

Business Components

- QuestionnaireService
  - public Questionnaire getQuestionnaireOfTheDay()
  - public Questionnaire findQuestionnaireById(int questionnaireId)
  - public void logQuestionnaireAccess(Questionnaire questionnaire, User user)
  - public void removeQuestionnaireLog(Questionnaire questionnaire, User user)
  - public boolean questionnaireAlreadySubmittedByUser(Questionnaire questionnaire, User user)
  - public Questionnaire createQuestionnaire(Product product, Date date, List<Question> questions)
  - public Questionnaire getQuestionnaireByDay(Date day)
  - public List<Questionnaire> getAllPastQuestionnaires()
  - public List<User> getUsersWhoAnsweredQuestionnaire(Questionnaire questionnaire)
  - public Questionnaire deleteQuestionnaire(Questionnaire questionnaire)

Business Components

- ProductService
  - public Product findProductId(int productId)
  - public List<Product> getAllProducts()
  - public Product createProduct(String name, byte[] img)
- QuestionService
  - public InputType getInputTypeByValue(InputTypeValue typeValue)
  - public List<FixedQuestion> getAllFixedQuestions()
- UserService
  - public User createUser(String username, String password, String email)
  - public boolean checkUsernameExists(String username)
  - public boolean checkEmailExists(String email)
  - public User checkCredentials(String usrn, String pwd)
  - public void banUser(User u)
  - public User findUserById(int userId)

Triggers

```
CREATE TRIGGER `answers_update_points` AFTER INSERT ON `answers` FOR EACH ROW BEGIN
    IF 1=(select is_statistical FROM questions where (idquestionnaire = new.idquestionnaire and question_number = new.question_number)) THEN
        INSERT INTO leaderboard (idquestionnaire, iduser, points)
        VALUES (new.idquestionnaire, new.iduser, 2)
        ON DUPLICATE KEY UPDATE points = points + 2;
    ELSE
        INSERT INTO leaderboard (idquestionnaire, iduser, points)
        VALUES (new.idquestionnaire, new.iduser, 1)
        ON DUPLICATE KEY UPDATE points = points + 1;
    END IF;
END
```

```
CREATE TRIGGER `answers_decrease_points` AFTER DELETE ON `answers` FOR EACH ROW BEGIN
    DECLARE is_statistical_var TINYINT;
    DECLARE question_points INT;
    SELECT is_statistical INTO is_statistical_var FROM questions WHERE (idquestionnaire = old.idquestionnaire and question_number = old.question_number);

    IF is_statistical_var = 1 THEN
        SET question_points = 2;
    ELSE
        SET question_points = 1;
    END IF;

    IF 0 < (SELECT points-question_points FROM leaderboard WHERE (idquestionnaire = old.idquestionnaire AND iduser = old.iduser)) THEN
        UPDATE leaderboard SET points = points - question_points WHERE (idquestionnaire = old.idquestionnaire AND iduser = old.iduser);
    ELSE
        DELETE FROM leaderboard WHERE (idquestionnaire = old.idquestionnaire AND iduser = old.iduser);
    END IF;
END
```