Data bases 2

Gamified Marketing application

Group 35

Specifications - User

An application deals with gamified consumer data collection. A user registers with a username, a password and an email. A registered user logs in and accesses a HOME PAGE where a “Questionnaire of the day” is published.

The HOME PAGE displays the name and the image of the “product of the day” and the product reviews by other users. The HOME PAGE comprises a link to access a QUESTIONNAIRE PAGE with a questionnaire divided in two sections: a section with a variable number of marketing questions about the product of the day (e.g., Q1: “Do you know the product?” Q2: Have you purchased the product before?” and Q3 “Would you recommend the product to a friend?”) and a section with fixed inputs for collecting statistical data about the user: age, sex, expertise level (low, medium high). The user fills in the marketing section, then accesses (with a next button) the statistical section where she can complete the questionnaire and submit it (with a submit button), cancel it (with a cancel button), or go back to the previous section and change the answers (with a previous button). All inputs of the marketing section are mandatory. All inputs of the statistical section are optional.

After successfully submitting the questionnaire, the user is routed to a page with a thanks and greetings message.

The database contains a table of offensive words. If any response of the user contains a word listed in the table, the transaction is rolled back, no data are recorded in the database, and the user’s account is blocked so that no questionnaires can be filled in by such account in the future.

When the user submits the questionnaire one or more trigger compute the gamification points to assign to the user for the specific questionnaire, according to the following rule:

1. One point is assigned for every answered question of section 1 (remember that the number of questions can vary in different questionnaires).
2. Two points are assigned for every answered optional question of section 2.

When the user cancels the questionnaire, no responses are stored in the database. However, the database retains the information that the user X has logged in at a given date and time.

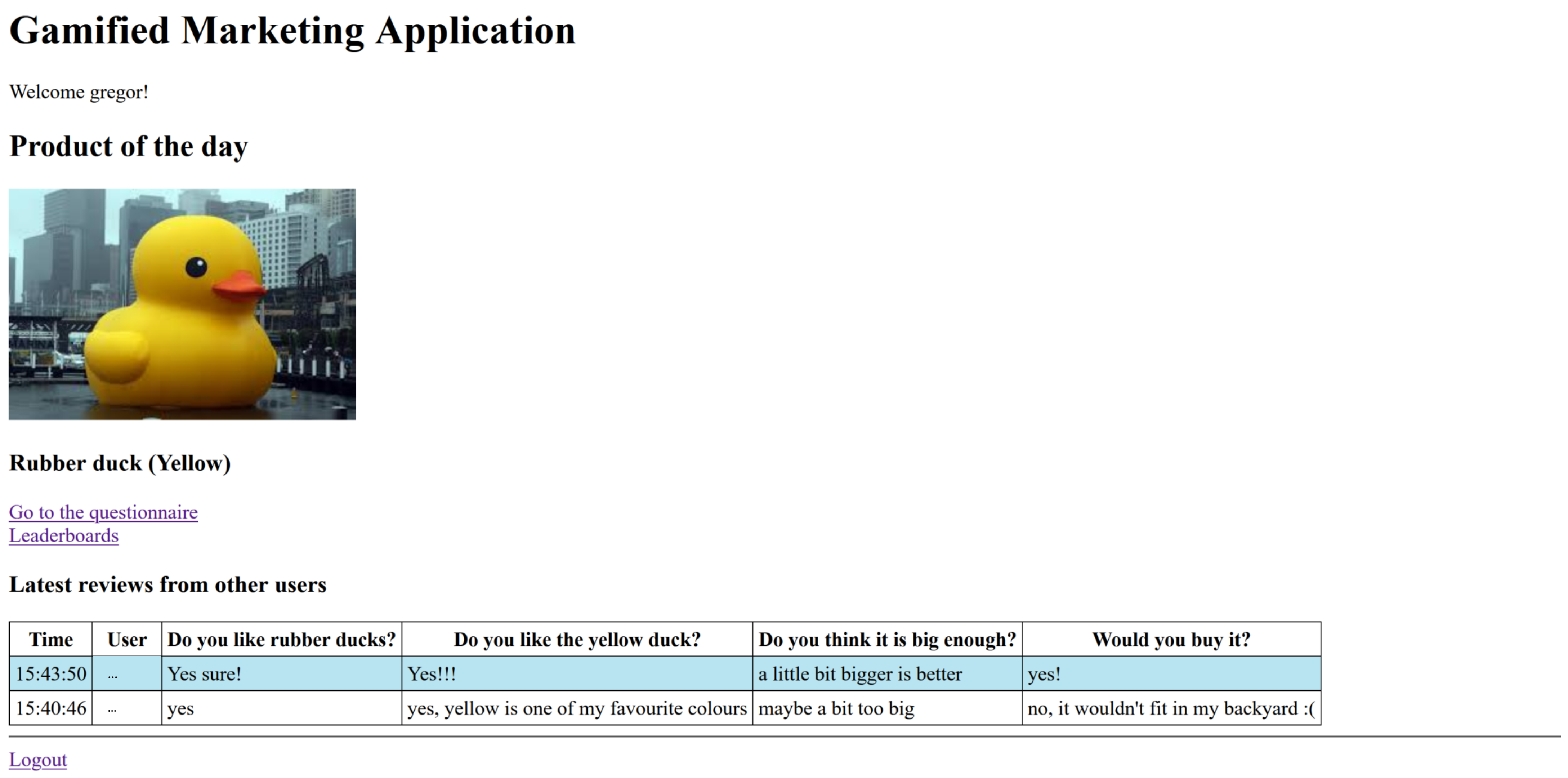
The user can access a LEADERBOARD page, which shows a list of the usernames and points of all the users who filled in the questionnaire of the day, ordered by the number of points (descending).

Specifications - Administrator

The administrator can access a dedicated application on the same database, which features the following pages

- A CREATION page for inserting the product of the day for the current date or for a posterior date and for creating a variable number of marketing questions about such product.
- An INSPECTION page for accessing the data of a past questionnaire. The visualized data for a given questionnaire include o List of users who submitted the questionnaire. o List of users who cancelled the questionnaire. o Questionnaire answers of each user.
- A DELETION page for ERASING the questionnaire data and the related responses and points of all users who filled in the questionnaire. Deletion should be possible only for a date preceding the current date.

Interface example



Gamified Marketing Application

Welcome gregor!

Product of the day

Rubber duck (Yellow)

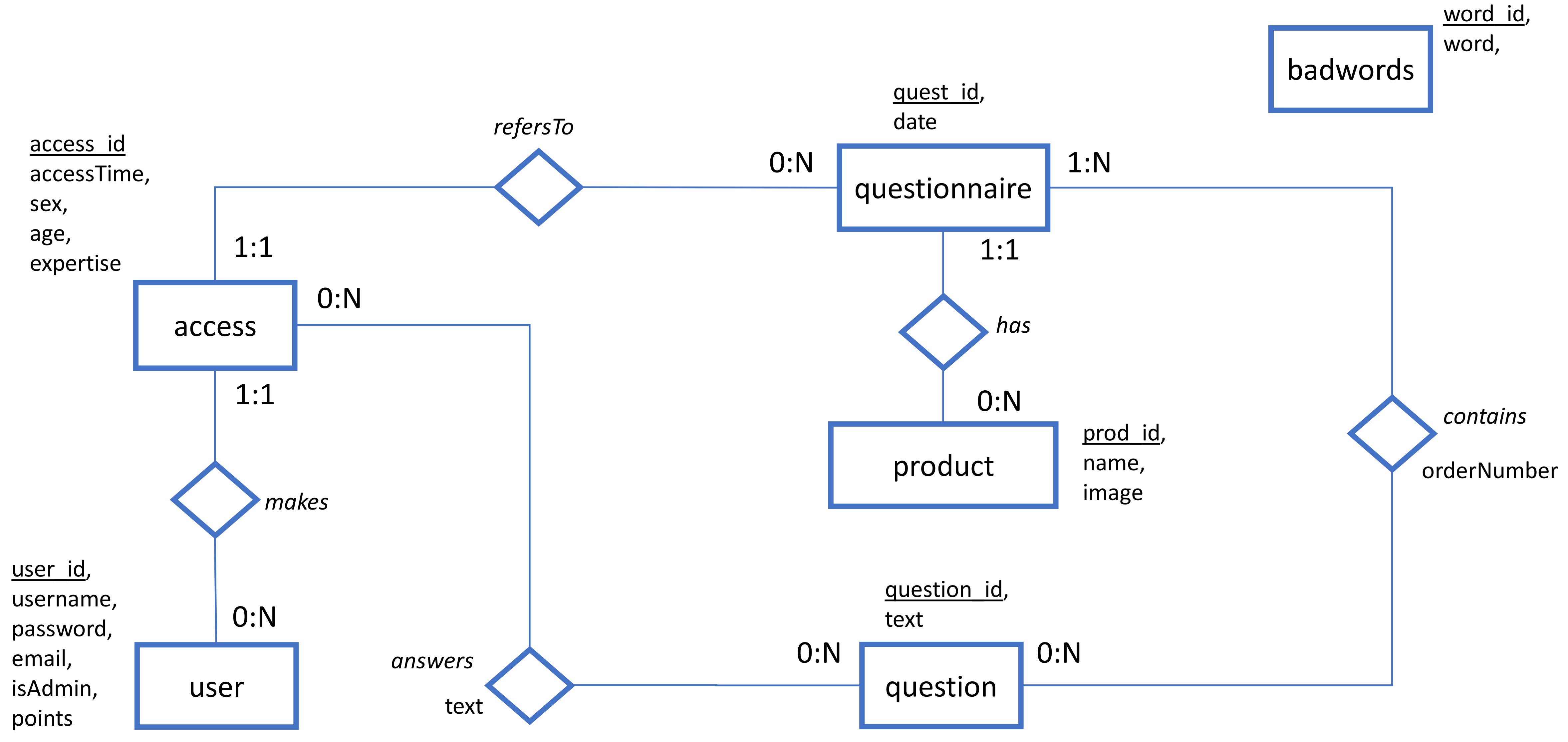
Go to the questionnaire
Leaderboards

Latest reviews from other users

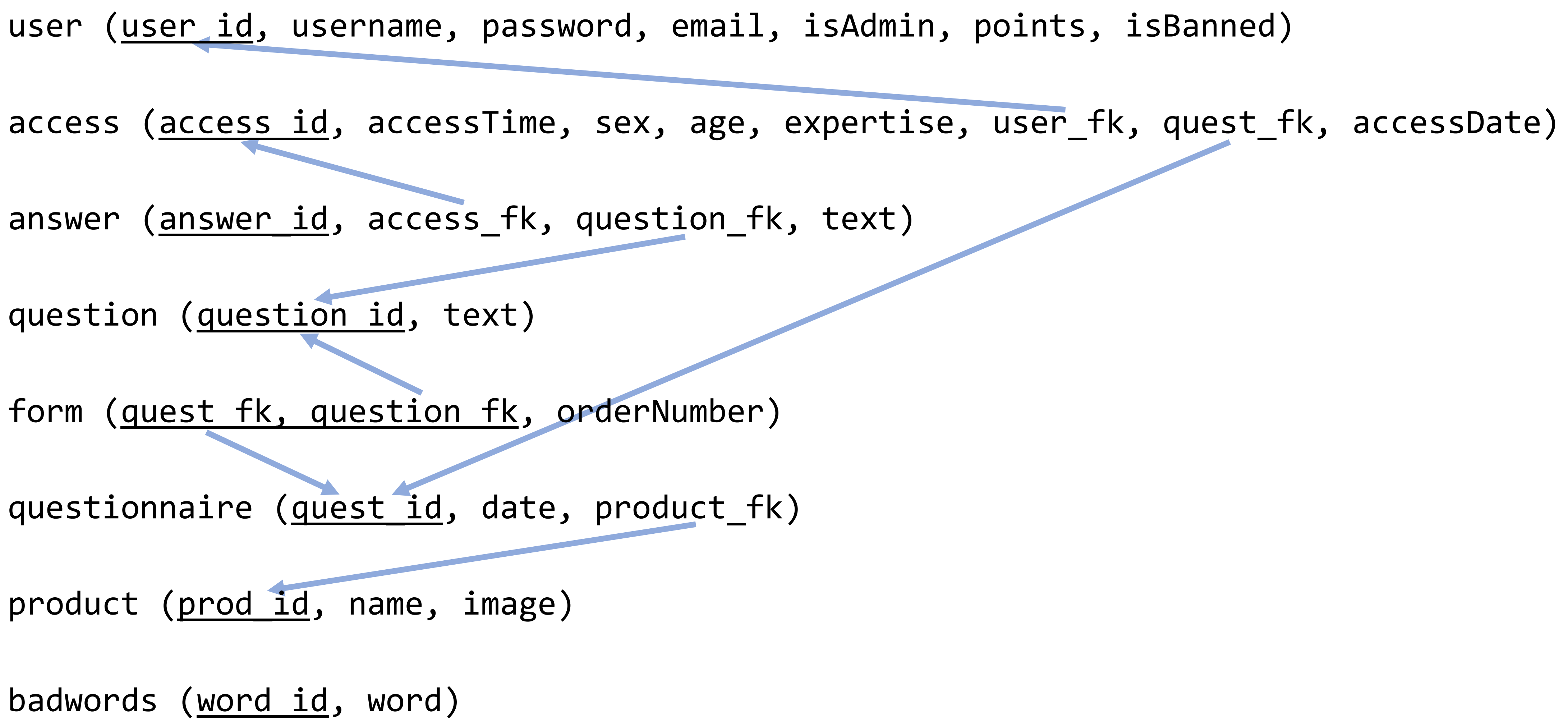
Time	User	Do you like rubber ducks?	Do you like the yellow duck?	Do you think it is big enough?	Would you buy it?
15:43:50	...	Yes sure!	Yes!!!	a little bit bigger is better	yes!
15:40:46	...	yes	yes, yellow is one of my favourite colours	maybe a bit too big	no, it wouldn't fit in my backyard :(

Logout

Entity Relationship

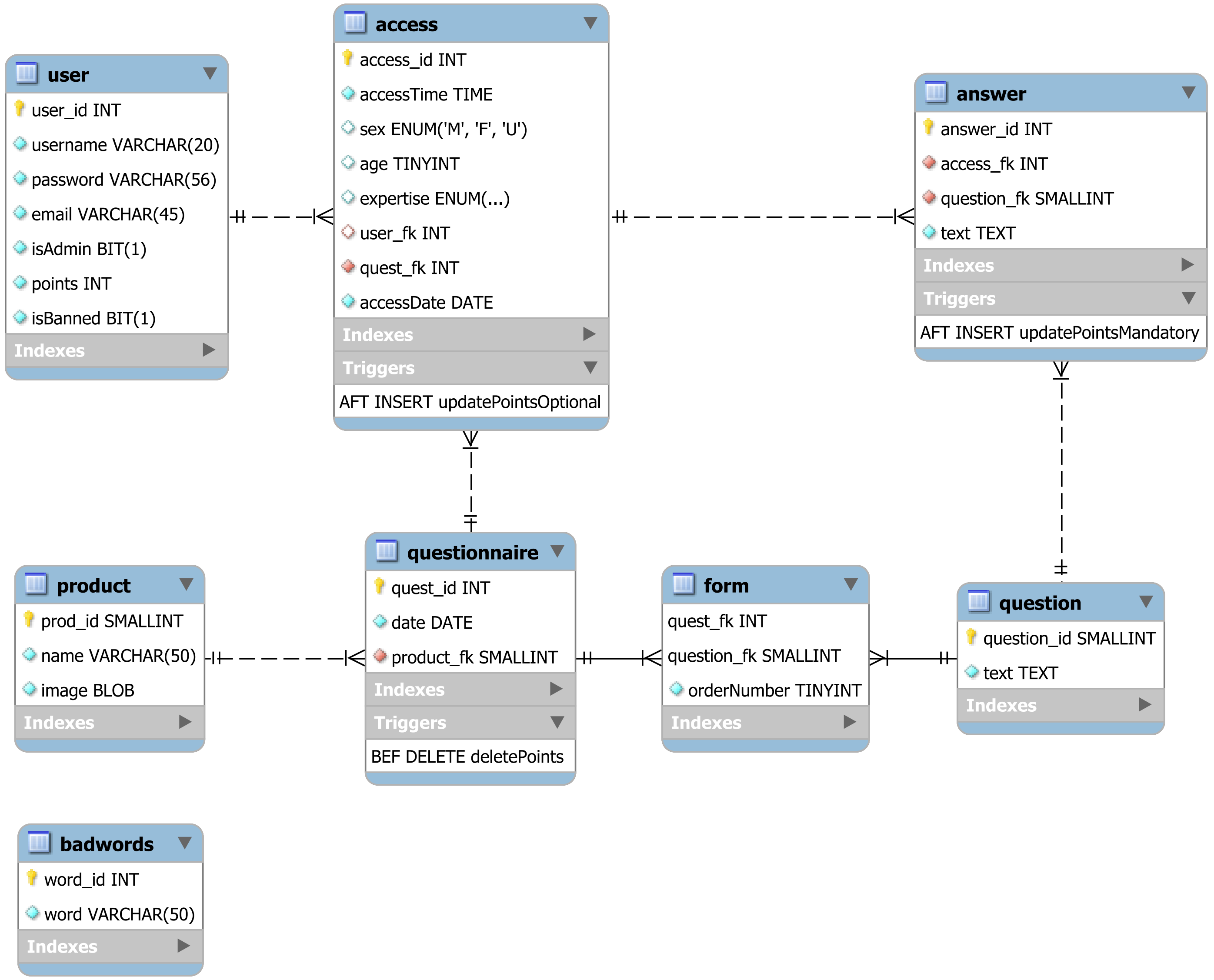


Relational model



```
user (user_id, username, password, email, isAdmin, points, isBanned)

access (access_id, accessTime, sex, age, expertise, user_fk, quest_fk, accessDate)

answer (answer_id, access_fk, question_fk, text)

question (question_id, text)

form (quest_fk, question_fk, orderNumber)

questionnaire (quest_id, date, product_fk)

product (prod_id, name, image)

badwords (word_id, word)
```

Model reverse engineered from MySQL WB



Motivations

- The *access* entity contains also optional questions (that are always the same) and is connected to *questions* through the *answer* table (in order to have multiple mandatory questions); submitted and cancelled questionnaires will have an access (they are told apart by the presence or absence of connected tuples in the *answer* table)
- Questions and products can be reused by multiple questionnaires, or even by none if all of their questionnaires were deleted
- The *badwords* table does not have any relationships, but is kept in the database instead of in a file in order to exploit the concurrency properties of MySQL

Triggers

Trigger to set mandatory points after a questionnaire insert

```
CREATE TRIGGER `updatePointsMandatory`

AFTER INSERT ON `answer`
FOR EACH ROW
BEGIN
    UPDATE user SET points=points+1
    WHERE user_id=(
        SELECT user_fk FROM access
        WHERE access_id=new.access_fk
    );
END
```

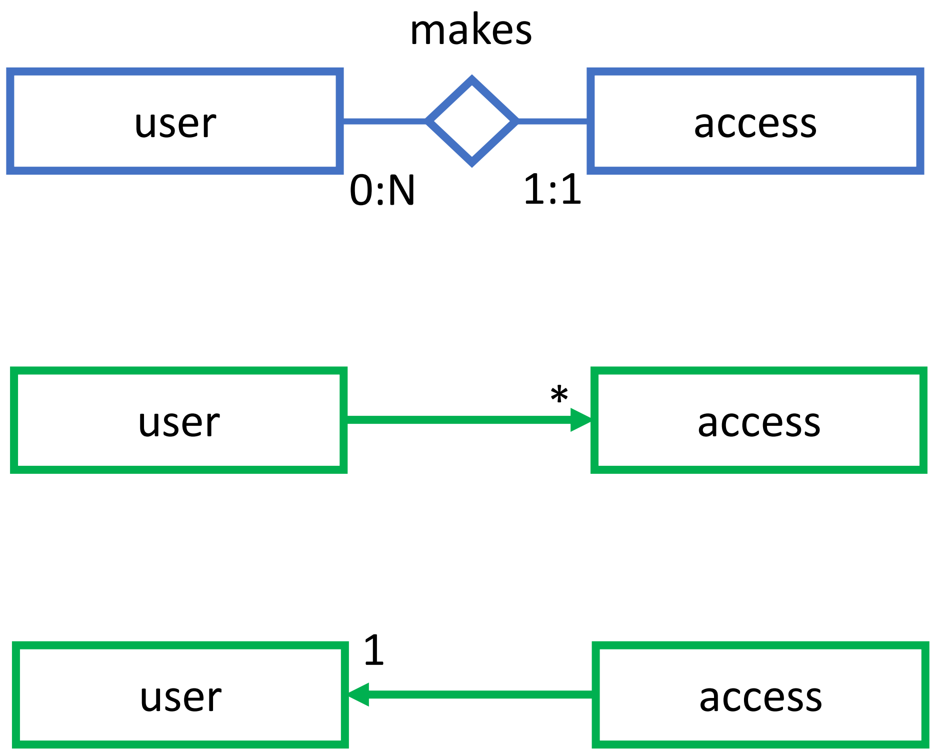
Trigger to set optional points after a questionnaire insert

```
CREATE TRIGGER `updatePointsOptional`
AFTER INSERT ON `access`
FOR EACH ROW BEGIN
    DECLARE p integer;
    SET p:=0;

    IF(new.sex IS NOT NULL AND new.sex <> 'U') THEN
        SET p:=p+2;
    END IF;
    IF(new.age IS NOT NULL AND new.age <> 0) THEN
        SET p:=p+2;
    END IF;
    IF(new.expertise IS NOT NULL AND new.expertise <> 'Unknown') THEN
        SET p:=p+2;
    END IF;

    UPDATE user SET points=points+p WHERE user_id=new.user_fk;
END
```

Trigger to delete points from both optional and mandatory points on a questionnaire deletion

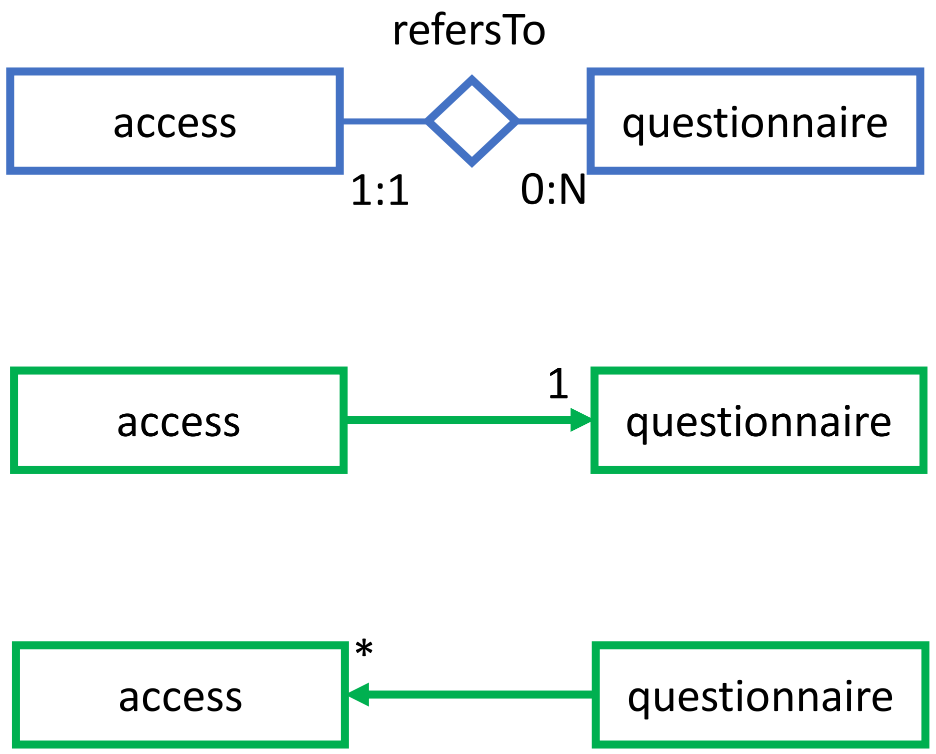
```
CREATE TRIGGER `deletePoints`
BEFORE DELETE ON `questionnaire`
FOR EACH ROW
BEGIN
    UPDATE user AS u JOIN (
        SELECT user_fk,
        CASE WHEN sex<>'U' THEN 2 ELSE 0 END
        + CASE WHEN age<>0 THEN 2 ELSE 0 END
        + CASE WHEN expertise<>'Unknown' THEN 2 ELSE 0 END
        + (
            SELECT COUNT(*) FROM answer
            WHERE access_fk=a.access_id
        ) AS points
        FROM access AS a
        WHERE access_id in (
            SELECT access_id
            WHERE quest_fk=old.quest_id
        )
    ) AS p ON user_fk=user_id
    SET u.points=u.points-p.points;
END
```

Relationship “makes”



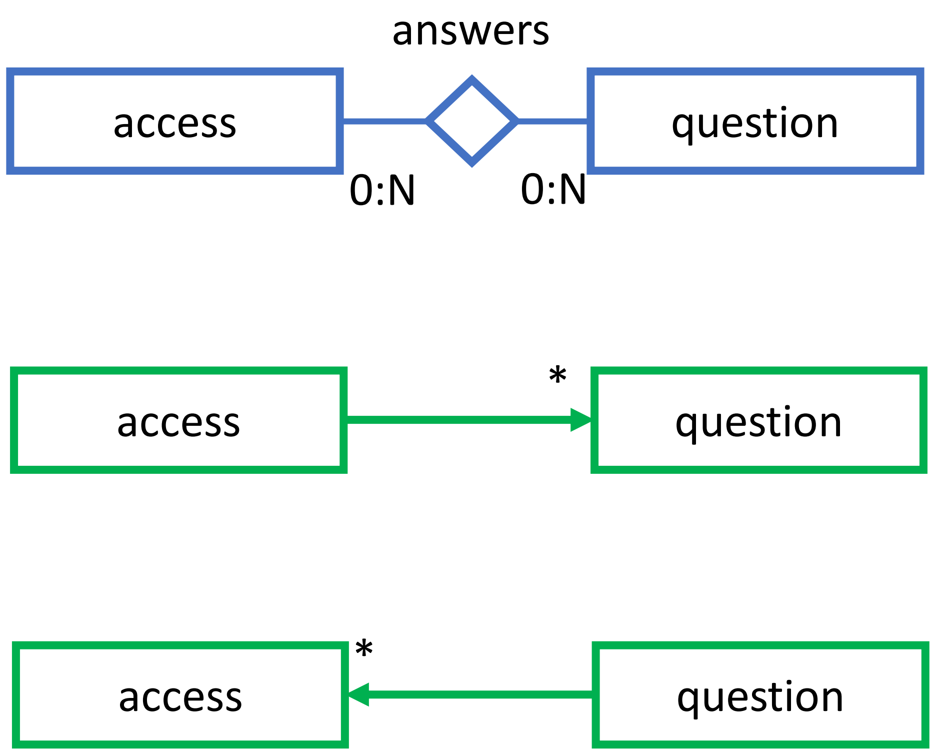
- user → access @OneToMany not mapped.
- access → user @ManyToOne unidirectional required to compute the users score shown in the leaderboard page, to show recent review by other users in the homepage and to avoid double submissions for the same questionnaire.

Relationship “refersTo”



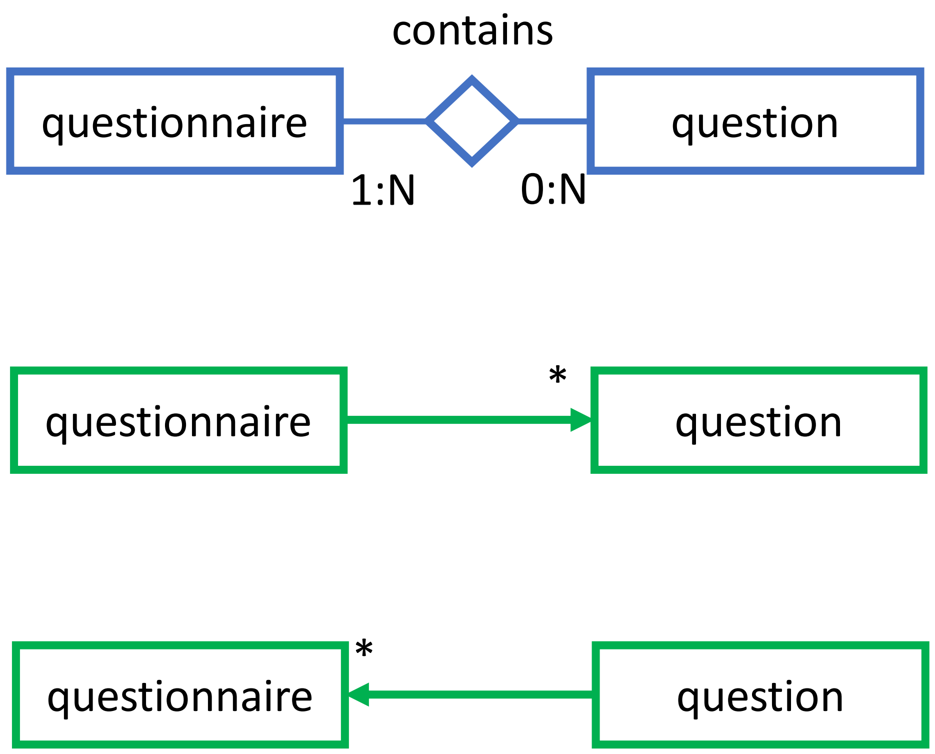
- access → questionnaire @ManyToOne mapped for simplicity. FetchType: lazy.
- questionnaire → access @OneToMany required to print latest reviews in the homepage and to print users who submitted/cancelled questionnaires in the inspect pages

Relationship “answers”



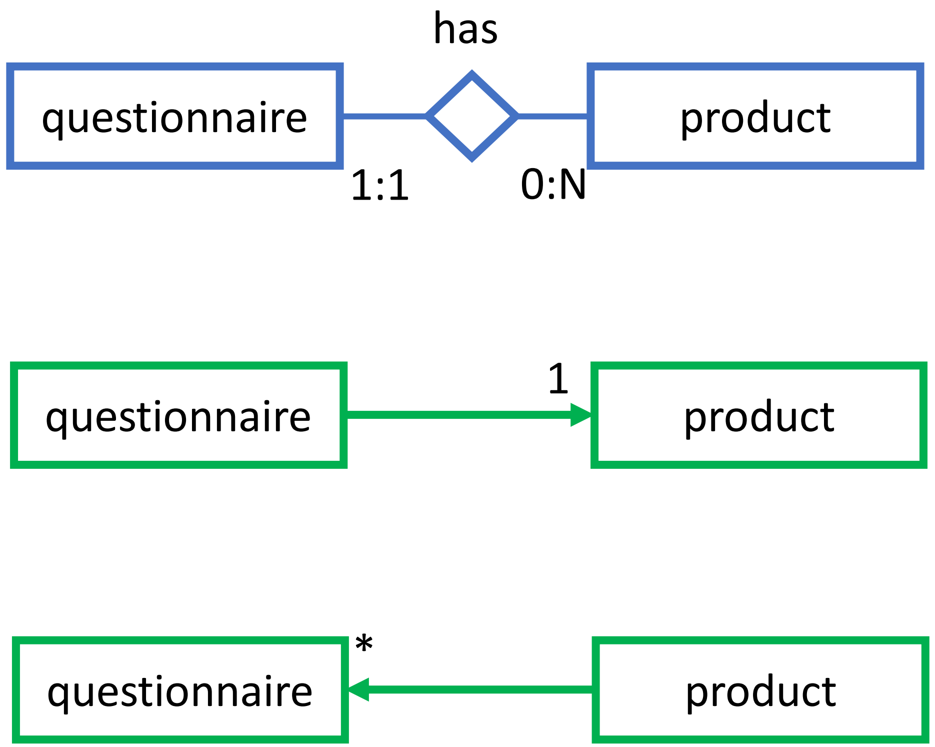
- access → question
  @ElementCollection
  @CollectionTable (name = "answer",
  joinColumns = @JoinColumn (name = "access_fk"))
  @MapKeyJoinColumn (name = "question_fk")
  @Column(name = "text")
  required to know answers of a user to a specific question to print reviews and in the inspect page
- question → access
  @ElementCollection (fetch = FetchType.LAZY)
  @CollectionTable (name = "answer",
  joinColumns = @JoinColumn (name = "question_fk"))
  @MapKeyJoinColumn (name = "access_fk")
  @Column(name = "text")
  not mandatory, mapped for simplicity.

Relationship “contains”



- questionnaire → question
  @ElementCollection
  @CollectionTable (name = "form",
  joinColumns = @JoinColumn (name = "quest_fk"))
  @MapKeyJoinColumn (name = "question_fk")
  @Column(name = "orderNumber")
  required to show the questions in the compile questionnaire page. Also it is used to print headers in reviews and inspect tables.
- question → questionnaire
  not mapped, not required

Relationship "has"



- questionnaire → product @ManyToOne unidirectional required to show product image and name in homepage for questionnaire of the day.
- product → questionnaire @OneToMany not mapped.

Entity Access

Equals and simple getters and setters will not be reported

```
@Table(name = "access")
@NamedQuery(name = "Access.findByUser",
      query = "select a from Access a where a.user = ?1 and a.accessDate = ?2")
@Entity
public class Access {
   @ManyToOne(optional = false)
   @JoinColumn(name = "user_fk", nullable = false)
   private User user;

   @ManyToOne(fetch = FetchType.LAZY, optional = false)
   @JoinColumn(name = "quest_fk", nullable = false)
   private Questionnaire questionnaire;

   @ElementCollection
   @CollectionTable (name = "answer",
         joinColumns = @JoinColumn (name = "access_fk"))
         @MapKeyJoinColumn (name = "question_fk")
         @Column(name = "text")
   private Map<Question, String> answers = new HashMap<>();

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "access_id", nullable = false)
   private Integer id;

   @Column(name = "accessTime")
   private LocalTime accessTime;

   @Column(name = "accessDate")
   private LocalDate accessDate;

   @Enumerated(EnumType.STRING)
   @Column(name = "sex")
   private Sex sex;

   @Column(name = "age")
   private Short age;

   @Enumerated(EnumType.STRING)
   @Column(name = "expertise")
   private Expertise expertise;
}
```

Entity Badwords

```java
@Table(name = "badwords")
@NamedQuery(name="Badwords.findAll", query = "select b from Badwords b")
@Entity
public class Badwords {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "word_id", nullable = false)
   private Integer word_id;

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

Entity Product

```java
@Table(name = "product")
@NamedQuery(name = "Product.findAll",
      query = "SELECT p FROM Product p")
@Entity
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "prod_id", nullable = false)
   private Integer prod_id;

   @Lob
   @Basic(fetch = FetchType.LAZY)
   @Column(name = "image", nullable = false)
   private byte[] image;

   @Column(name = "name", nullable = false, length = 50)
   private String name;
}
```

Entity Question

```
@Table(name = "question")
@NamedQuery(name = "Question.findAll",
      query = "SELECT q FROM Question q")
@NamedQuery(name = "Question.findByText",
      query = "SELECT q FROM Question q WHERE q.text = ?1")
@Entity
public class Question {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "question_id", nullable = false)
   private Integer question_id;

   @Column(name = "text", nullable = false, length = 65536)
   private String text;
}
```

Entity Questionnaire

```java
@Table(name = "questionnaire")
@NamedQuery(name = "Questionnaire.findQuestionnaireByDate",
      query = "SELECT q FROM Questionnaire q " +
            "WHERE q.date=?1")
@NamedQuery(name = "Questionnaire.findAll",
      query = "SELECT q FROM Questionnaire q ORDER BY q.date DESC ")
@NamedQuery(name = "Questionnaire.findUntil",
      query = "SELECT q FROM Questionnaire q " +
            "WHERE q.date<=?1 ORDER BY q.date DESC")
@NamedQuery(name = "Questionnaire.findAllDates",
      query = "SELECT q.date FROM Questionnaire q")
@Entity
public class Questionnaire {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "quest_id", nullable = false)
   private Integer quest_id;

   @Column(name = "date", nullable = false)
   private LocalDate date;

   @OneToMany(mappedBy = "questionnaire")
   private List<Access> accesses;

   @ElementCollection
   @CollectionTable (name = "form",
         joinColumns = @JoinColumn (name = "quest_fk"))
   @MapKeyJoinColumn (name = "question_fk")
   @Column(name = "orderNumber")
   private Map<Question, Integer> questions = new HashMap<>();

   @ManyToOne(optional = false)
   @JoinColumn(name = "product_fk", nullable = false)
   private Product product;

   public List<Question> getQuestions() {
      return questions.entrySet().stream().sorted(Map.Entry.comparingByValue())
            .map(e -> e.getKey()).collect(Collectors.toList());
   }

   public List<Access> getCanceled() {
      return getAccesses().stream()
            .filter(a -> a.getSex() == null && a.getAge() == null && a.getExpertise() == null)
            .collect(Collectors.toList());
   }

   public List<Access> getSubmitted() {
      return getAccesses().stream()
            .filter(a -> a.getSex() != null && a.getAge() != null && a.getExpertise() != null)
            .sorted(Comparator.comparing(Access::getAccessTime).reversed())
            .collect(Collectors.toList());
   }
}
```

Entity User

```
@Table(name = "user")
@NamedQuery(name = "User.login",
      query = "SELECT u FROM User u WHERE u.username=?1 AND u.password=?2")
@NamedQuery(name = "User.findByName",
      query = "SELECT u FROM User u WHERE u.username=?1")
@NamedQuery(name = "User.getLeaderboard",
      query = "SELECT u FROM User u WHERE NOT u.isAdmin ORDER BY u.points DESC")
@Entity
public class User {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "user_id", nullable = false)
   private Integer id;

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

   @Basic(fetch = FetchType.LAZY)
   @Column(name = "password", nullable = false)
   private String password;

   @Basic(fetch = FetchType.LAZY)
   @Column(name = "email", nullable = false, unique = true)
   private String email;

   @Column(name = "isAdmin", nullable = false)
   private Boolean isAdmin = false;

   @Column(name = "points", nullable = false)
   private Integer points;

   @Column(name = "isBanned", nullable = false)
   private Boolean isBanned;
}
```

Components

Format: (*source html page*) Component (*generated html page*)

- Client components (User)
    - CompileQuestionnaire
    - DiscardQuestionnaire
    - GoToHomePage (*home.html*)
    - GoToLeaderboard (*leaderboard.html*)
    - GoToQuestionnaire (*questionnaire.html*)
    - (*signup.html*) Signup
    - Submitted (*submitted.html*)
- Client components (Admin)
    - CreateQuestionnaire
    - DeleteQuestionnaire
    - GoToAdmin (*admin.html*)
    - GoToCreate (*newQuestionnaire.html*)
    - GoToDelete (*deleteList.html*)
    - GoToInspectList (*inspectList.html*)
    - GoToInspectQuestionnaire (*inspectQuestionnaire.html*)
- Components used by both
    - (*index.html*) Login
    - (*home.html*, *admin.html*) Logout

- Business Components
    - AccessBean (Stateless)
        - void createAccess(User u, Sex sex, Short age, Expertise ex, Map<Integer, String> answ)
        - Access findAccessByUser(LocalDate date, User user)
    - BadwordBean (Stateless)
        - List<Badwords> getBadwords()
        - Boolean checkForBadwords(Map<Integer, String> answ)
    - ProductBean (Stateless)
        - Product createProduct(String productName, Part imageFile)
        - Product findProductById(Integer id)
        - List<Product> findAllProducts()
    - QuestionBean (Stateless)
        - Question createQuestion(String text)
        - List<Question> findAllQuestions()
        - Question findQuestionByText(String text)
    - QuestionnaireBean (Stateless)
        - Questionnaire findQuestionnaireByDate(LocalDate date)
        - List<Questionnaire> findAll()
        - List<Questionnaire> findUntil(LocalDate date)
        - List<String> findAllDates()
        - Questionnaire findById(Integer id)
        - int deleteQuestionnaire(Integer id)
        - Questionnaire createQuestionnaire(LocalDate date, Product product, Map<Question, Integer> questionMap)
    - UserBean (Stateless)
        - User checkCredentials(String username, String password)
        - Boolean isNameTaken(String username)
        - User createUser(String user, String password, String email)
        - void banUser(User u)
        - List<UserLeaderboard> retrieveLeaderboard()

Business method to create a new access

Method in AccessBean: creates the access by linking answers

```
public void createAccess(User u, Sex sex, Short age, Expertise ex, Map<Integer, String> answ){
    Questionnaire questionnaire = questionnaireBean.findQuestionnaireByDate(LocalDate.now());
    access = new Access();
    access.setAccessTime(LocalTime.now());
    access.setAccessDate(LocalDate.now());
    access.setAge(age);
    access.setSex(sex);
    access.setExpertise(ex);
    access.setUser(u);
    access.setQuestionnaire(questionnaire);
    if(answ != null) {
        answ.forEach((key, text) -> access.addAnswer(em.find(Question.class, key), text));
    }
    em.persist(access);
}
```

Business method to obtain the leaderboards

Method in UserBean: finds points and users, maps it to a new object to be sent to the client

```java
public List<UserLeaderboard> retrieveLeaderboard() {
   return em.createNamedQuery("User.getLeaderboard", User.class)
         .setHint("javax.persistence.cache.storeMode", "REFRESH")
         .getResultList().stream().map(u-> new UserLeaderboard(u.getUsername(), u.getPoints()))
         .collect(Collectors.toList());
}
```

Business method to find already existing questions

Method in QuestionBean: finds questions by text in order to not duplicate them

```
public Question findQuestionByText(String text) {
   List<Question> questions =  em.createNamedQuery("Question.findByText", Question.class)
         .setParameter(1, text)
         .setHint("javax.persistence.cache.storeMode", "REFRESH")
         .getResultList();
   if(questions != null && !questions.isEmpty()) {
      return questions.get(0);
   }
   return null;
}
```

Business method to check for the presence of bad words in answers

Method in BadwordBean: checks for the presence of bad words in the given list of answers

```
public Boolean checkForBadwords(Map<Integer, String> answ) {
   List<Badwords> badwords = getBadwords();

   Boolean hasBadword = false;

   if (badwords != null && !badwords.isEmpty()) {
      Collection<String> answers = answ.values();
      Iterator<String> i = answers.iterator();
      String cur_answ;

      while(!hasBadword && i.hasNext()) {
         cur_answ = i.next();
         for (Badwords bad : badwords) {
            hasBadword = hasBadword || cur_answ.contains(bad.getWord());
         }
      }
   }

   return hasBadword;
}
```

Business method to find dates associated to questionnaires

Method in QuestionnaireBean: obtains and returns the dates

```
public List<String> findAllDates() {
    List<String> dates = em.createNamedQuery("Questionnaire.findAllDates", LocalDate.class)
            .getResultList().stream().map(LocalDate::toString).collect(Collectors.toList());
    if (dates.isEmpty()) {
        return new ArrayList<String>();
    }
    return dates;
}
```