POLITECNICO
MILANO 1863

Gamified Market

Data Bases 2

Lecturer : Piero Fraternali

Students : Puoti Francesco, Ravella Elia

Specifications

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 and a section with fixed inputs for collecting statistical data about the user. The user fills in the marketing section, then accesses (with a *next* button) the statistical section where s/he 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
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

Gamified consumer data collection

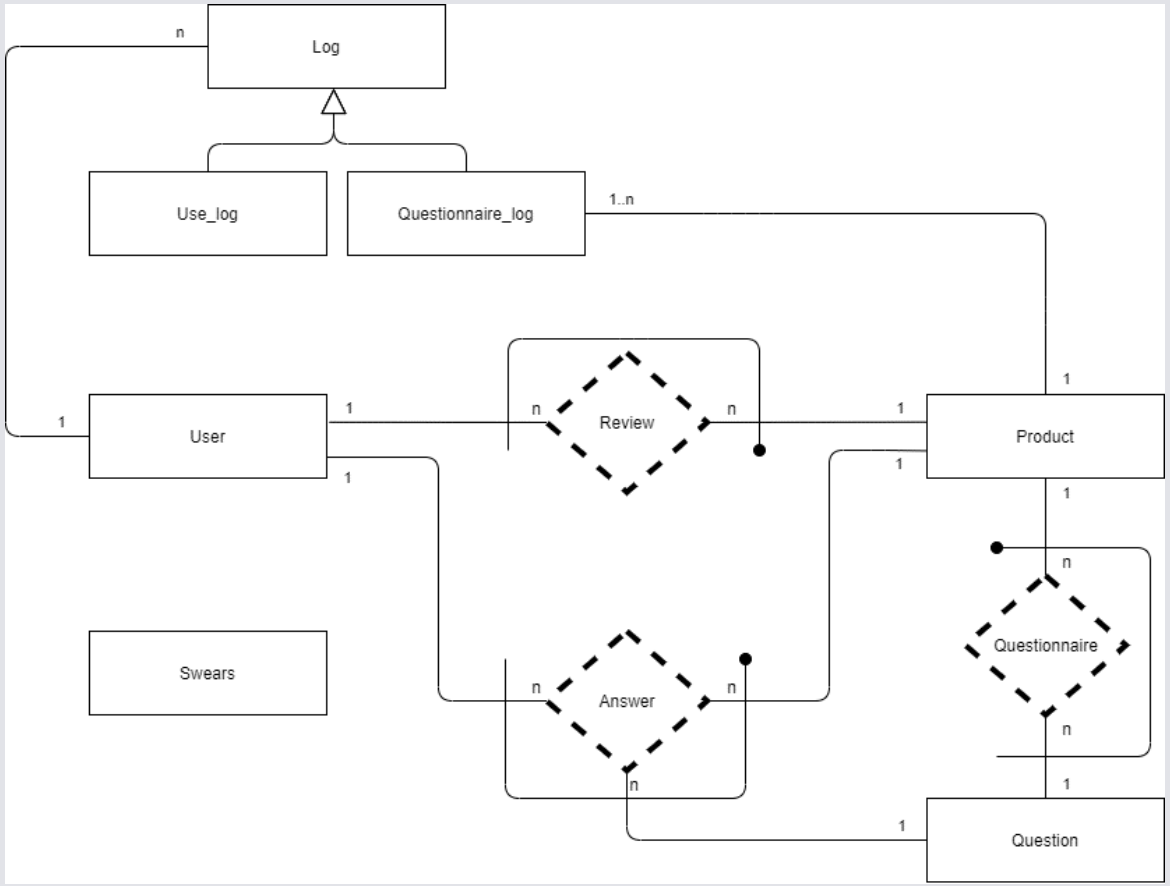
The administrator can access a dedicated application on the same database, which features the following pages :

- 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 includes :
  - List of users who submitted the questionnaire.
  - List of users who cancelled the questionnaire.
  - 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.

Additional specifications

- The marketing section of a questionnaire cannot be null
- A trigger that links the statistical questions to each newly inserted product

Entity Relationship



Relational Model

- Answer (id_product, id_user, id_question, answer_text);
- Product(id_product, name, product_image, date);
- Question(id_question,question_text_points);
- Questionnaire(id_product, id_question);
- Questionnaire_log (id_user, datetime, action, id_product);
- Review(id_user, id_product, review_text, date);
- Swears(swear_text) ;
- User(id_user, username, email, password, authorized, points, admin, active);
- User_log (id_user, datetime, action);

We preferred to add also the relational model to show all the attributes without making the ER diagram heavier.
All the foreign keys are represented using the same name as the one on the owner table.

Relational Model

- Answer (id_product, id_user, id_question, answer_text);
- Product(id_product, name, product_image, date);
- Question(id_question,question_text_points);
- Questionnaire(id_product, id_question);
- Questionnaire_log (id_user, datetime, action, id_product);
- Review(id_user, id_product, review_text, date);
- Swears(swear_text) ;
- User(id_user, username, email, password, authorized, points, admin, active);
- User_log (id_user, datetime, action);

We preferred to add also the relational model to show all the attributes without making the ER diagram heavier.

Rationale

We decided to define a pure-relational database schema without using any ‘Many-to-Many’ relationships. The rational of this decision consists of the will of having the full control on the data base, i.e. having a complete mapping between the data managed by JPA and the data actually stored into the data base.

SQL DDL - Tables

```
create table answer
(
    id_product  int  not null,
    id_user     int  not null,
    id_question int  not null,
    answer_text text not null,
    constraint answer_pk
        unique (id_product, id_user, id_question),
    constraint answer_product_id_product_fk
        foreign key (id_product) references product (id_product)
            on update cascade on delete cascade,
    constraint answer_question_id_question_fk
        foreign key (id_question) references question
(id_question)
            on update cascade on delete cascade,
    constraint answer_user_id_user_fk
        foreign key (id_user) references user (id_user)
            on update cascade

);
```

```
create trigger add_points
    after insert
    on answer
    for each row
begin
        update user
        set user.points = user.points + (select q.points from
question as q where q.id_question = new.id_question)
        where user.id_user = new.id_user;
    end;

create trigger rm_points
    after delete
    on answer
    for each row
begin
        update user
        set user.points = user.points - (select q.points from
question as q where q.id_question = old.id_question)
        where user.id_user = old.id_user;
    end;
```

SQL DDL - Tables

```
create table product
(
    id_product    int auto_increment
        primary key,
    name          varchar(64) not null,
    product_image longblob    not null,
    date          date        not null,
    constraint product_date_uindex
        unique (date)
);
```

```
create trigger statistical_questions
    after insert
    on product
    for each row
begin
    insert into questionnaire
    select NEW.id_product, id_question
    from question where points = 2;
end;
```

SQL DDL - Tables

```
create table question
(
    id_question    int auto_increment
        primary key,
    question_text varchar(255) not null,
    points        int          not null,
    constraint question_question_text_uindex
        unique (question_text)
);
```

```
create table questionnaire
(
    id_product  int not null,
    id_question int not null,
    primary key (id_product, id_question),
    constraint questionnaire_product_id_product_fk
        foreign key (id_product) references product
(id_product)
            on update cascade on delete cascade,
    constraint questionnaire_question_id_question_fk
        foreign key (id_question) references question
(id_question)
            on update cascade on delete cascade
);
```

SQL DDL - Tables

```
create table questionnaire_log
(
    id_user     int         not null,
    datetime    datetime    not null,
    action      varchar(64) not null,
    id_product int          null,
    primary key (id_user, datetime),
    constraint questionnaire_log_product_id_product_fk
        foreign key (id_product) references product
(id_product)
            on update cascade on delete cascade,
    constraint questionnaire_log_user_id_user_fk
        foreign key (id_user) references user (id_user)
            on update cascade
)
```

```
create table user_log
(
    id_user  int        not null,
    datetime datetime    not null,
    action   varchar(64) not null,
    primary key (id_user, datetime),
    constraint user_log_user_id_user_fk
        foreign key (id_user) references user (id_user)
            on update cascade
)
```

comment 'The foreign key ''id_user'' has "on update cascade and on delete no action" because it's needed to be always available to check reliable statistical data. Therefore, also if a user is deleted, the data of the submit questionnaire must be retained. Otherwise, on update, the data has to cascading be updated since all the data is to be retrieved from that user. It's different the case in which a product is deleted from the DB: all the data can be deleted as the product is, probably, not under analysis anymore.';

SQL DDL - Tables

```
create table review
(
   id_user     int      not null,
   id_product  int      not null,
   review_text text     not null,
   date        datetime not null,
   primary key (id_user, id_product),
   constraint review_product_id_product_fk
      foreign key (id_product) references product
(id_product)
         on update cascade on delete cascade,
   constraint review_user_id_user_fk
      foreign key (id_user) references user
(id_user)
         on update cascade
)
   comment 'the primary key is composed by
(id_user, id_product) because we assumed that
every user can review each product at most once.';
```

```
create table swears
(
   swear_text varchar(64) not null
      primary key
);
```

SQL DDL - Tables

```
create table user
(
    id_user     int auto_increment
        primary key,
    username    varchar(64)      not null,
    email       varchar(64)      not null,
    password    varchar(255)     not null,
    authorized bit default b'1' not null,
    points      int default 0    not null,
    admin       bit              not null,
    active      bit default b'1' not null,
    constraint user_email_uindex
        unique (email),
    constraint user_username_uindex
        unique (username)
);
```

SQL DDL - View

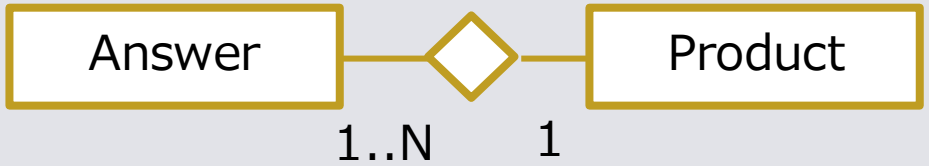
```
create definer = root@localhost view
userquestionnairepoints as
select `aw`.`id_user`                AS `id_user`,
     `aw`.`id_product`               AS `id_product`,
     sum(`gamified_market`.`q`.`points`) AS `points`
from (`gamified_market`.`answer` `aw`
      join `gamified_market`.`questionnairedetails` `q`)
where ((`gamified_market`.`q`.`id_product` =
`aw`.`id_product`) and
     (`aw`.`id_question` =
`gamified_market`.`q`.`id_question`))
group by `aw`.`id_user`, `aw`.`id_product`;
```

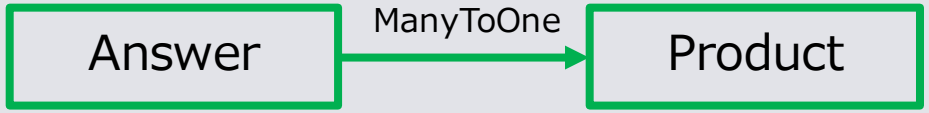
SQL DDL - Routines

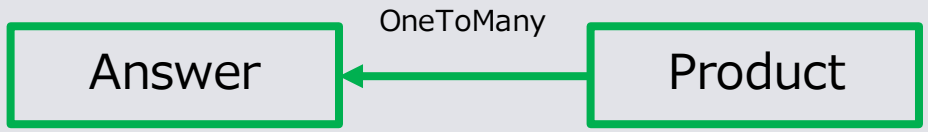
```
create procedure delete_questionnaires_details
(IN product_toRemove int)
begin
   IF current_date > (select date from product where
product.id_product = product_toRemove)
      THEN
         delete from answer
         where id_product = product_toRemove;

         delete from questionnaire
         where id_product = product_toRemove;
   ELSE
      SIGNAL SQLSTATE '42000'
         SET MESSAGE_TEXT = 'You can't delete the
questionnaire's data, since the date is not preceding the
current one';
   END IF;
end;
```

```
create function insert_answer
(in_product int, in_user int, in_question int, in_text text) returns int
BEGIN
   DECLARE swear_num int;
   select count(*)
   into swear_num
   from swears
   where in_text LIKE CONCAT('%', swear_text, '%');

   IF swear_num > 0
      THEN
         RETURN -1;
   ELSE
      insert into answer
      (id_product, id_user, id_question, answer_text)
      values (in_product, in_user, in_question, in_text);
      RETURN 1;
   END IF;
END;
```

Relationships < Answer – Product >







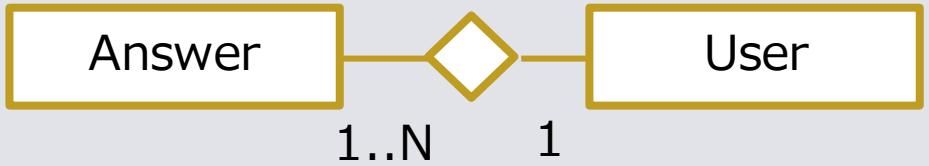
Answer Entity

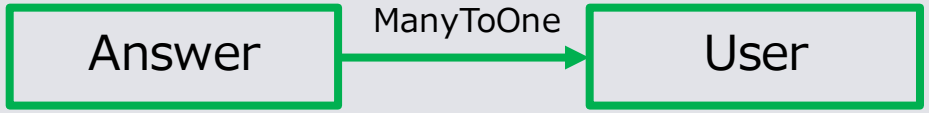
```
private int idProduct;
private Product productByIdProduct;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_product",
                      referencedColumnName = "id_product")
public Product getProductByIdProduct();
```

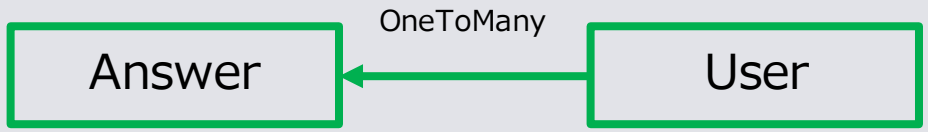
Product Entity

```
private int idProduct;
private Collection<Answer> answersByIdProduct;

@OneToMany(mappedBy = "productByIdProduct")
public Collection<Answer> getAnswersByIdProduct();
```

Relationships < Answer – User >







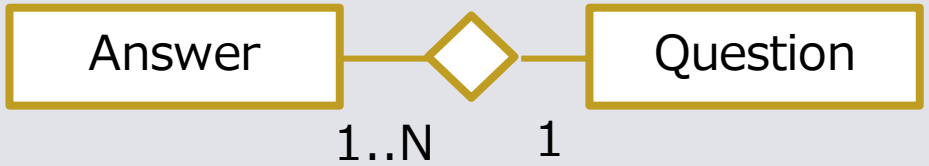
Answer Entity

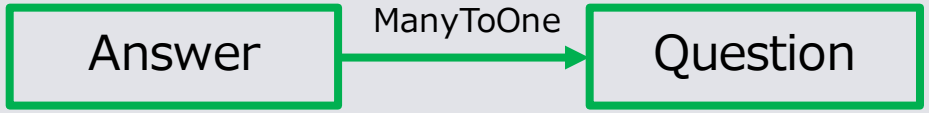
```
private int idUser;
private User userByIdUser;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_user",
                      referencedColumnName = "id_user")
public User getUserByIdUser();
```

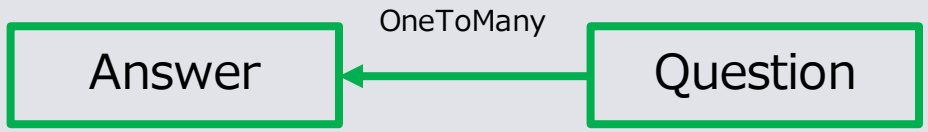
User Entity

```
private Collection<Answer> answersByIdUser;

@OneToMany(mappedBy = “userByIdUser")
public Collection<Answer> getAnswersByIdUser();
```

Relationships < Answer – Question >







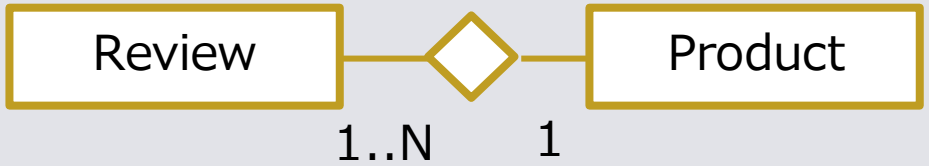
Answer Entity

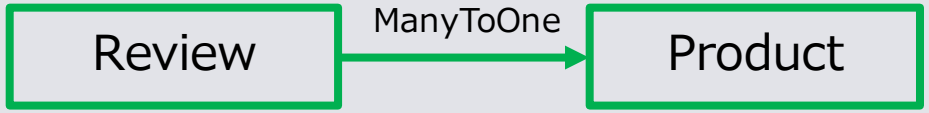
```
private int idQuestion;
private Question questionByIdQuestion;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_question",
                      referencedColumnName = "id_question")
public Question getQuestionByIdQuestion();
```

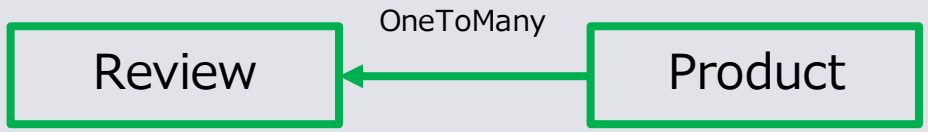
Question Entity

```
private Collection<Answer> answersByIdQuestion;

@OneToMany(mappedBy = "questionByIdQuestion")
    public Collection<Answer> getAnswersByIdQuestion() ;
```

Relationships < Review – Product >







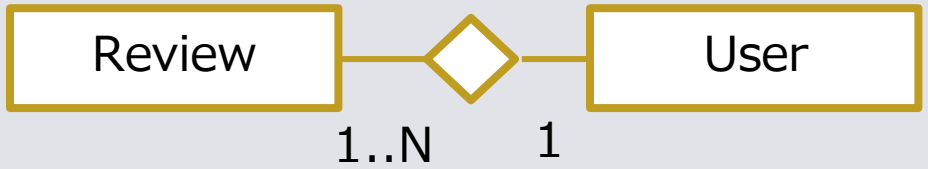
Review Entity

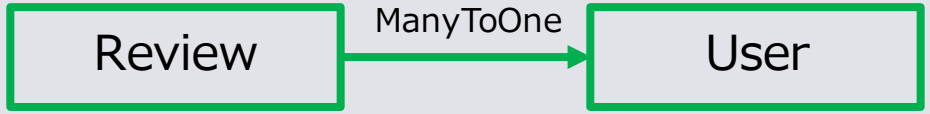
```
private Product productByIdProduct;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_product",
                      referencedColumnName = "id_product")
public Product getProductByIdProduct();
```

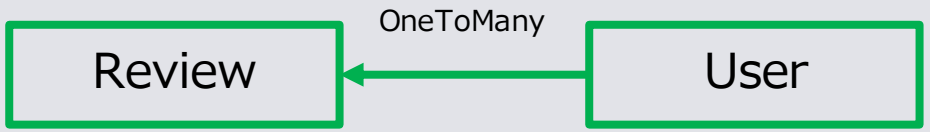
Product Entity

```
private Collection<Review> reviewsByIdProduct;

@OneToMany(mappedBy = "productByIdProduct")
public Collection<Review> getReviewsByIdProduct() ;
```

Relationships < Review – User >







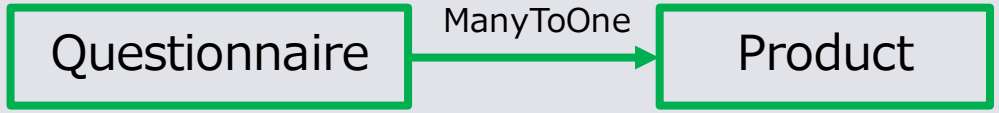
Review Entity

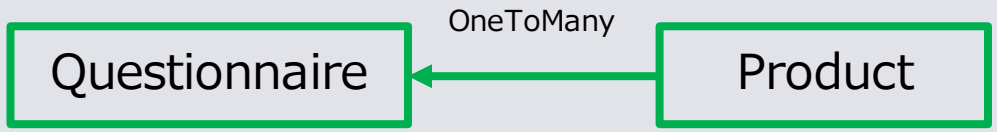
```
private User userByIdUser;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_user",
                      referencedColumnName = "id_user")
 public User getUserByIdUser();
```

User Entity

```
private Collection<Review> reviewsByIdUser;

@OneToMany(mappedBy = "userByIdUser")
public Collection<Review> getReviewsByIdUser();
```

Relationships < Questionnaire – Product >





Questionnaire Entity

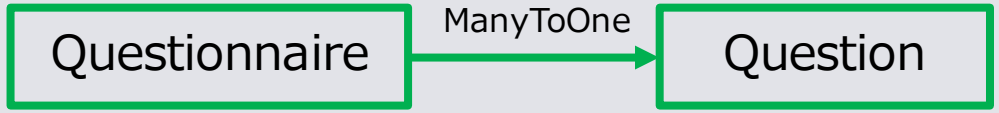
```
private int idProduct;
private Product productByIdProduct;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_product", referencedColumnName =
"id_product")
public Product getProductByIdProduct();
```

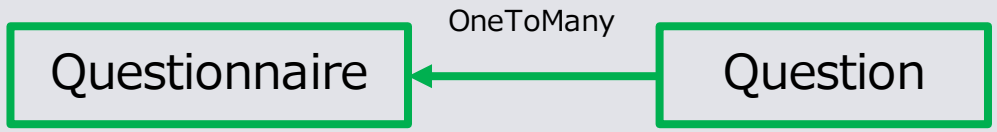
_____

Product Entity

```
 private Collection<Questionnaire>questionnairesByIdProduct;

 @OneToMany(mappedBy = "productByIdProduct")
    public Collection<Questionnaire> getQuestionnairesByIdProduct();
```

Relationships < Questionnaire – Question >





Questionnaire Entity

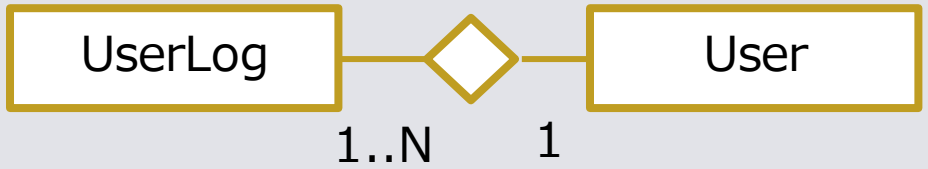
```
private int idQuestion;
private Question questionByIdQuestion;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_question", referencedColumnName =
"id_question")
public Question getQuestionByIdQuestion();
```

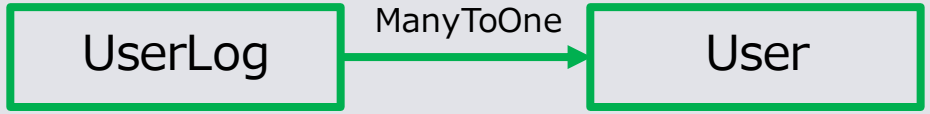
_____

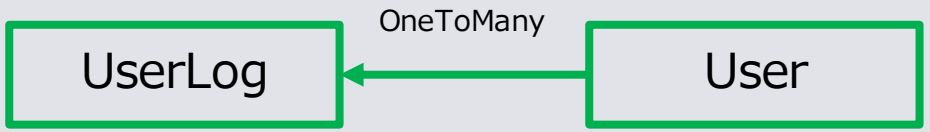
Question Entity

```
 private Collection<Question>questionnairesByIdQuestion;

 @OneToMany(mappedBy = « questionByIdQuestion")
    public Collection<Questionnaire> getQuestionnairesByIdQuestion();
```

Relationships < User – user_log >







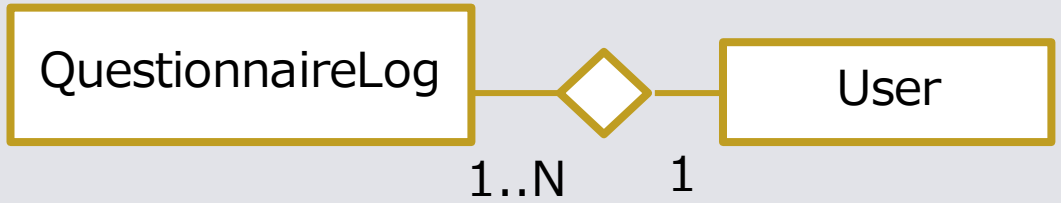
UserLog Entity

```
private User userByIdUser;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_user",
                      referencedColumnName = "id_user")
 public User getUserByIdUser();
```

User Entity

```
private Collection<Userlog> userlogsByIdUser;

@OneToMany(mappedBy = "userByIdUser")
public Collection<Userlog> getUserlogsByIdUser();
```

Relationships < User – questionnaire_log>



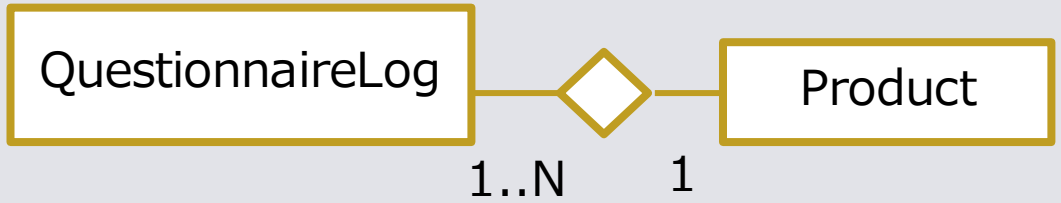
QuestionnaireLog Entity

```
private User userByIdUser;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_user",
                      referencedColumnName = "id_user")
 public User getUserByIdUser();
```

User Entity

```
private Collection<QuestionnaireLog> questionnaireLogsByIdUser;

@OneToMany(mappedBy = "userByIdUser")
 public Collection<QuestionnaireLog> getQuestionnaireLogsByIdUser();
```

Relationships < Product – questionnaire_log>



QuestionnaireLog Entity

```
private User userByIdUser;

@ManyToOne
@PrimaryKeyJoinColumn(name = "id_product",
                      referencedColumnName = "id_product")
 public Product getProductByIdProduct();
```

_____

Product Entity

```
private Collection<QuestionnaireLog> questionnaireLogsByIdProduct;

@OneToMany(mappedBy = "productByIdProduct")
public Collection<QuestionnaireLog> getQuestionnaireLogsByIdProduct();
```

Entities

From now on all the entities will be depicted.

Notice that we did not write the aforementioned methods (those for laying out the relationships), but only the “class” attributes and their mapping on the database.

Property-access has been chosen for the annotations, thus, even though we represented only the getters, the setters have been implemented as well. Exception made for the “UserQuestionnairePoints” which, since it’s @ReadOnly, does not support the property-access.

Entity - Answer

```
@Entity
@Table(name = "answer")
@IdClass(AnswerPK.class)
@NamedQueries({
      @NamedQuery(name = "Answer.getAllAnswers", query =
"SELECT a FROM Answer a")
})
public class Answer {
   public Answer() {
   }

   private int idProduct;
   private int idUser;
   private int idQuestion;
   private String answerText;
```

```
@Id
@Column(name = "id_product", nullable = false)
public int getIdProduct();

@Column(name = "id_user", nullable = false)
@Id
public int getIdUser() ;

@Column(name = "id_question", nullable = false)
@Id
public int getIdQuestion() ;

@Column(name = "answer_text", nullable = false, length = -1)
public String getAnswerText() ;
```

Entity - Product

```
@Entity @Table(name = "product")
@NamedQueries({
    @NamedQuery(name = "Product.findAllProducts", query = "select p from Product p"),
    @NamedQuery(name = "Product.getPastProduct", query = "SELECT p from Product p WHERE p.date < current_date"),
    @NamedQuery(name = "Product.getProductOfTheDay", query = "SELECT p from Product p WHERE p.date = current_date"),
    @NamedQuery(name = "Product.getProductByIdProduct", query = "SELECT p FROM Product p where p.idProduct = :idProduct")
})
public class Product {
  private int idProduct;
  private String name;
  private byte[] productImage;
  private Date date;
```

```
@Id   @Column(name = "id_product", nullable = false)
@GeneratedValue(strategy=GenerationType.IDENTITY)
public int getIdProduct() ;

@Column(name = "name", nullable = false, length = 64)
public String getName() ;

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

/* This method has to be used when you want to display the image in thymeleaf
 * @return Base64 encoded image */
public String imageString();

@Column(name = "date", nullable = false)
public Date getDate() ;
```

Entity - Question

```
@Entity
@Table(name = "question")
@NamedQueries(
      {@NamedQuery(name = "Question.findAllQuestions",
query = "select q from Question q"),
       @NamedQuery(name =
"Question.findQuestionIdByText", query = "select
q.idQuestion from Question q where q.questionText = ?1")}
)
public class Question {
    private int idQuestion;
    private String questionText;
    private int points;
```

```
@Id
@Column(name = "id_question", nullable = false)
@GeneratedValue(strategy=GenerationType.IDENTITY)
    public int getIdQuestion();

@Column(name = "question_text", nullable = false,
length = 255)
public String getQuestionText() ;

@Column(name = "points", nullable = false)
public int getPoints() ;
```

Entity - Questionnaire

```
@Entity
@Table(name = "questionnaire")
@IdClass(QuestionnairePK.class)
@NamedQueries({
      @NamedQuery(name = "Questionnaire.getQuestions", query
= "SELECT q from Questionnaire q WHERE q.idProduct = ?1"),
      @NamedQuery(name =
"Questionnaire.getAllQuestionnaires", query = "SELECT q FROM
Questionnaire q")
})
public class Questionnaire {
   private int idProduct;
   private int idQuestion;
```

```
@Id
@Column(name = "id_product", nullable = false)
public int getIdProduct();

@Id
@Column(name = "id_question", nullable = false)
public int getIdQuestion();
```

Entity - QuestionnaireLog

```
@Entity
@Table(name = "questionnaire_log")
@IdClass(QuestionnaireLogPK.class)
@NamedQuery(name = "QuestionnaireLog.retrieveProductLog",
      query = "SELECT ql FROM QuestionnaireLog as ql WHERE
ql.idProduct = ?1")
public class QuestionnaireLog {

   private int idUser;
   private Timestamp datetime;
   private String action;
   private Integer idProduct;
```

```
@Id
@Column(name = "id_user", nullable = false)
public int getIdUser();

@Id
@Column(name = "datetime", nullable = false)
public Timestamp getDatetime() ;

@Column(name = "action", nullable = false, length = 64)
public String getAction() ;

@Column(name = "id_product")
public Integer getIdProduct();
```

Entity – Review & Swears

```
@Entity
@Table(name = "review")
@IdClass(ReviewPK.class)
public class Review {
    private int idUser;
    private int idProduct;
    private String reviewText;
    private Timestamp date;
```

```
@Id
@Column(name = "id_user", nullable = false)
public int getIdUser() ;

@Id
@Column(name = "id_product", nullable = false)
public int getIdProduct() ;

@Column(name = "review_text", nullable = false, length = -1)
public String getReviewText() ;

@Column(name = "date", nullable = false)
public Timestamp getDate();
```

```
@Entity
@Table(name = "swears")
public class Swears {
    private String swearText;
@Id
@Column(name = "swear_text", nullable = false, length = 64)
public String getSwearText() ;
```

Entity - User

```
@Entity
@Table(name = "user")
@NamedQuery(name = "User.checkCredentials", query =
"SELECT r FROM User r  WHERE r.email = ?1 and
r.password = ?2")
public class User {
    private int idUser;
    private String username;
    private String email;
    private String password;
    private boolean authorized;
    private int points;
    private boolean admin;
    private boolean active;
```

```
@Id
@Column(name = "id_user", nullable = false)
@GeneratedValue(strategy=GenerationType.IDENTITY)
public int getIdUser() ;

@Column(name = "username", nullable = false, length = 64)
public String getUsername() ;

@Column(name = "email", nullable = false, length = 64)
public String getEmail() ;

@Column(name = "password", nullable = false, length = 255)
public String getPassword() ;

@Column(name = "authorized", nullable = false)
public boolean isAuthorized() ;

@Column(name = "points", nullable = false)
public int getPoints();

@Column(name = "admin", nullable = false)
public boolean isAdmin() ;

@Column(name = "active", nullable = false)
public boolean isActive() ;
```

Entity – Userlog

```
@Entity
@Table(name = "user_log")
@IdClass(UserlogPK.class)
public class Userlog {
    private int idUser;
    private Timestamp datetime;
    private String action;
    private User userByIdUser;
```

```
@Id
@Column(name = "id_user", nullable = false)
public int getIdUser() ;

@Id
@Column(name = "datetime", nullable = false)
public Timestamp getDatetime() ;


@Column(name = "action", nullable = false, length = 64)
public String getAction() ;
```

Entity – UserQuestionnairePoints

```
@ReadOnly
@Entity
@Table(name = "userquestionnairepoints")
@IdClass(UserQuestionnairePointsPK.class)
@NamedQuery(name = "UserQuestionnairePoints", query = "SELECT r FROM
UserQuestionnairePoints r WHERE r.idProduct = ?1 ORDER BY r.userPoints DESC")
public class UserQuestionnairePoints {
    @Id
    @Column(name = "id_product", nullable = false)
    private int idProduct;
    @Id
    @Column(name = "id_user", nullable = false)
    private int idUser;
    @Column(name = "points")
    private int userPoints;
```

Client components

Client components:

- AddNewProduct
- AdminHomePage
- InspectQuestionnaire
- LeaderBoard
- LogOut
- QuestionnaireServletMarketing
- QuestionnaireServletStatistical
- QuestionnaireSummary
- RemoveQuestionnaire
- ReviewQuestionnaires
- SignIn
- SignUp
- UserHomePage

Server Components

- Logger @Stateless
- *Void logAction(idUser, UserAction, Integer);*
- *List<Questionnaires> retrieveProductLog(idProduct);*

- ProductService @Stateless
- *Product getProductOfTheDay();*
- *List<Review> getReviews(idProduct);*
- *List<Product> getAllProducts();*
- *Review addReview(IdProduct, IdUser, reviewTxt, date);*
- *Map<User, Integer> getLeaderBoard();*
- *Product addProduct(productName, productImage, productDate);*
- *getPastProducts();*

Server Components – cont'd

- QuestionnaireService @Stateless
- *List<Questionnaire> retrieveQuestionnaire(IdProduct);*
- *List<Question> retrieveQuestions(IdProduct);*
- *List<Answer> retrieveAllQuestionnaires();*

- QuestionService @Stateless
- *Question addQuestion(questionText, questionPoints);*
- *Int findQuestionByText(String qText);*
- *List<Question> getAllQuestions();*

Server Components – cont'd

- UserQuestionnaire @Stateful @SessionScoped
- *Void insertSingleAnswer(Answer asw);*
- *UserAction validateUserQuestionnaire();*
- *Void cancelQuestionnaire(User);*
- *Boolean alreadyFullfilled(User);*
- *List<Questionnaire> getCurrentSectionQuestions();*

- UserService @Stateless
- *User checkCredentials(email, password);*
- *User registerNewUser(username, email, password);*
- *Void banUser(idUser);*

Thank you for your attention