Data bases 2

JPA exercise: Gamified marketing application

Specifications

Develop an application to deal with gamified consumer data collection.
The implementation requires users authentication (normal and admin users).

Users can access:

- An HOME PAGE which displays the name and image of the PRODUCT OF THE DAY and product reviews by other users.
- A QUESTIONNAIRE PAGE with a questionnaire divided in two section, one with (open) **mandatory** marketing questions and one with statistical **optional** questions (with fixed input).
  Users can submit or cancel a questionnaire and go back to the marketing questions and change the answers.
- 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**)

When a user submits a questionnaire, points are assigned as follows:

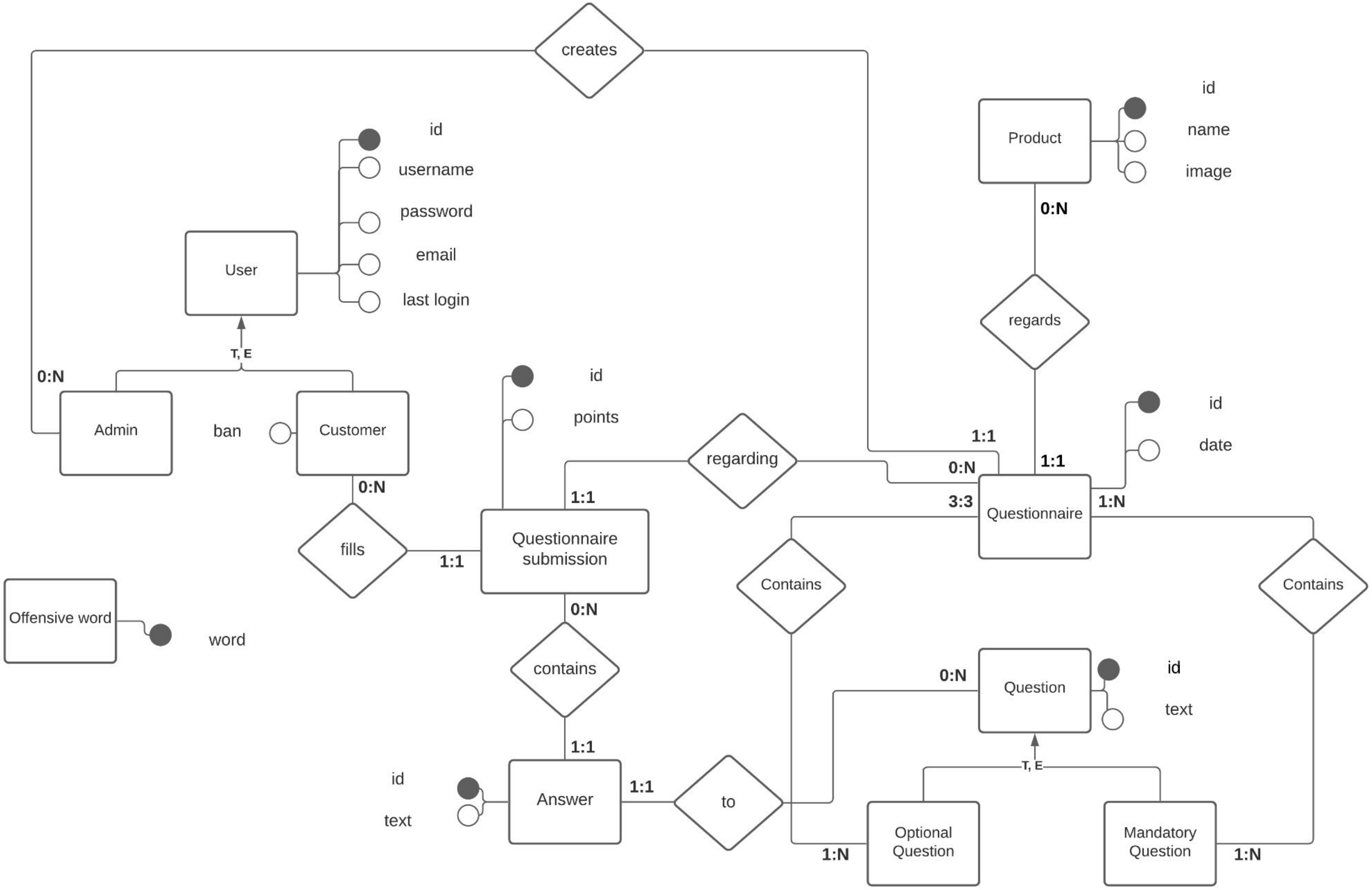
- One point is assigned for every answered question of section 1
- Two points are assigned for every answered optional question of section 2.

Administrators can access:

- 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.
- 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.

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.

Entity Relationship



Relational model

```
create table offensive_word (
    word varchar(255) not null,
    constraint offensive_word_word_uindex unique (word)
);
alter table offensive_word
add primary key (word);
```

```
create table product (
    id int auto increment,
    name varchar(255) not null,
    image blob null,
    constraint product_id_uindex unique (id)
);
alter table product
add primary key (id);
```

Relational model

```
create table question (
    id int auto increment,
    text text not null,
    optional tinyint(1) default 0 not null,
    constraint question_id_uindex unique (id)
);
alter table question
add primary key (id);
```

```
create table user (
    id int auto_increment,
    username varchar(255) not null,
    password varchar(255) not null,
    email varchar(255) not null,
    lastlogin datetime null,
    ban tinyint(1) default 0 not null,
    admin tinyint(1) null,
    constraint user_email_uindex unique (email),
    constraint user id uindex unique (id),
    constraint user_username_uindex unique (username)
);
alter table user
add primary key (id);
```

Relational model

```
create table answer (
    id int auto increment,
    text text not null,
    questionnaire_submission_id int null,
    question_id int null,
    constraint answer_id_uindex unique (id),
    constraint answer question id fk foreign key (question_id) references question (id)
        on update cascade on delete cascade,
    constraint answer_questionnaire_submission_id_fk foreign key (questionnaire_submission_id)
        references questionnaire_submission (id)
        on update cascade on delete cascade
);
alter table answer
add primary key (id);
```

Relational model

```sql
create table questionnaire (
    id int auto_increment,
    date date not null,
    user_id int null,
    product_id int not null,
    constraint questionnaire_date_uindex unique (date),
    constraint questionnaire_id_uindex unique (id),
    constraint questionnaire_product_id_fk foreign key (product_id)
        references product (id) on update cascade on delete cascade,
    constraint questionnaire_user_id_fk foreign key (user_id)
        references user (id) on update cascade on delete cascade
);
alter table questionnaire
add primary key (id);
```

Relational model

```
create table questionnaire_submission
(
    id int auto_increment,
    points int default 0 null,
    user_id int null,
    questionnaire_id int null,
    constraint questionnaire_submission_id_uindex
        unique (id),
    constraint questionnaire_submission_questionnaire_id_fk
        foreign key (questionnaire_id) references questionnaire (id)
            on update cascade on delete cascade,
    constraint questionnaire_submission_user_id_fk
        foreign key (user_id) references user (id)
            on update cascade on delete cascade
);
alter table questionnaire_submission
    add primary key (id);


create table questionnaire_to_question
(
    questionnaire_id int not null,
    question_id int not null,
    primary key (questionnaire_id, question_id),
    constraint questionnaire_to_question_question_id_fk
        foreign key (question_id) references question (id)
            on update cascade on delete cascade,
    constraint questionnaire_to_question_questionnaire_id_fk
        foreign key (questionnaire_id) references questionnaire (id)
            on update cascade on delete cascade
);
```

Triggers

```
create trigger add_point
after
insert on answer for each row begin
set @point = 1;
if (
    select optional
    from db2.question
    where db2.question.id = new.question_id
) = true then
set @point = 2;
end if;
update db2.questionnaire_submission
set points = points + @point
where id = NEW.questionnaire_submission_id;
end;
```

```
create trigger check_offensive_words
after
insert on answer for each row begin if exists(
        select *
        from db2.offensive_word ow
        where upper(new.text) like concat('%', upper(ow.word), '%')
    ) then signal sqlstate '40666'
set message_text = 'Bad word detected';
end if;
end;
```

Triggers

```
create trigger remove_point
after delete on answer for each row begin
set @point = 1;
if (
    select optional
    from db2.question
    where db2.question.id = old.question_id
) = true then
set @point = 2;
end if;
update db2.questionnaire_submission
set points = points - @point
where id = old.questionnaire_submission_id;
end;
```

```
create trigger add_statistical_questions
after
insert on questionnaire for each row begin
INSERT INTO questionnaire_to_question
SELECT new.id,
    question.id
FROM question
WHERE optional = true;
end;
```

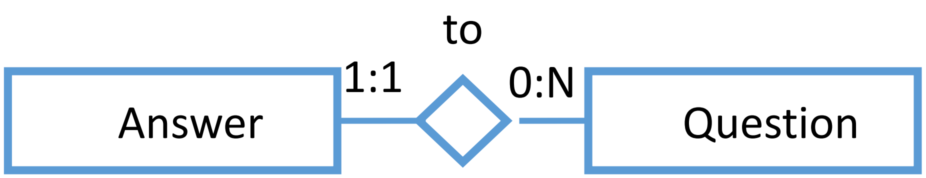
Motivation

In the logical model we have chosen to collapse mandatory and optional questions *in a single entity* **Question**, using a selector to distinguish between them.

This forced us to define a bridge table, not exploiting the limited cardinality of the optional questions.

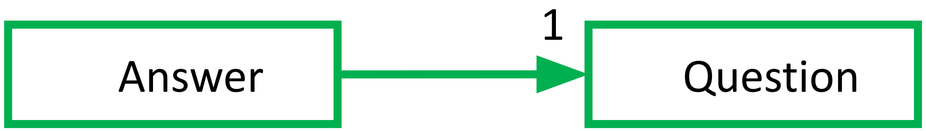
On the other hand, we think this allows for a better flexibility in reusing the mandatory questions, and in extending the number of optional questions.

Relationship “to”



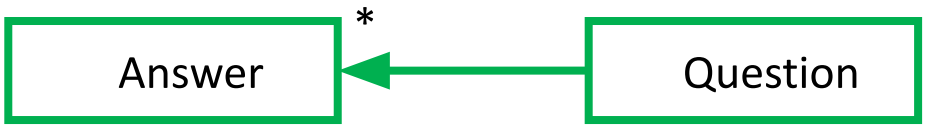
Answer Question
@ManyToOne

- ➔ Eager Fetch
- ➔ Cascade in All cases apart from Delete



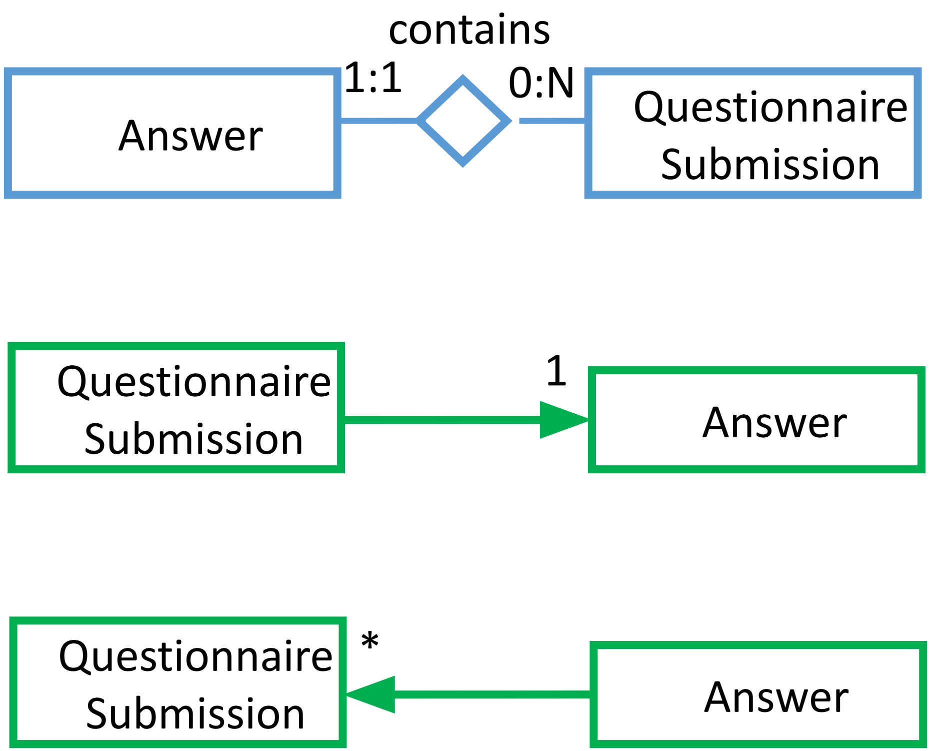
Question Answer
@OneToMany

- ➔ Lazy Fetch
- ➔ Cascade in All cases
- ➔ Orphan Removal



Removing an Answer should not remove the Question, but when a question is removed also all the answer to it should be removed

Relationship “contains”



Answer Submission

@ManyToOne

➔ Lazy Fetch

➔ Cascade in All cases apart from Delete

Submission Answer

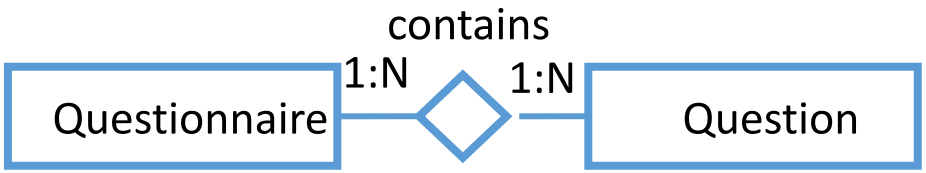
@OneToMany

➔ Lazy Fetch

➔ Cascade in All cases

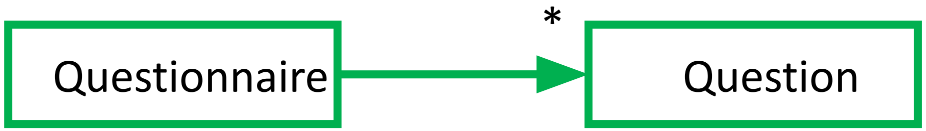
➔ Orphan Removal

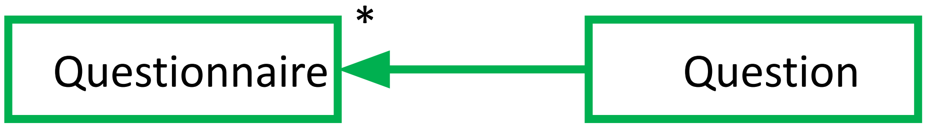
Relationship “contains”



Questionnaire Question
@ManyToMany

- ➔ Lazy Fetch
- ➔ Cascade in All cases apart from Delete



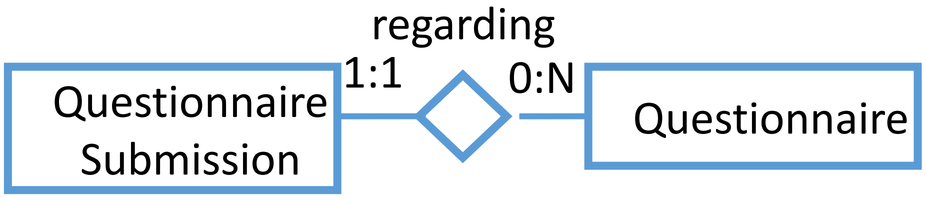


Question Questionnaire
@ManyToMany

- ➔ Lazy Fetch
- ➔ Cascade in All cases apart from Delete

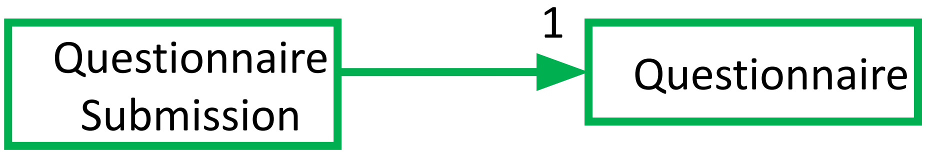
Join table is **Questionnaire_to_Questionnaire**

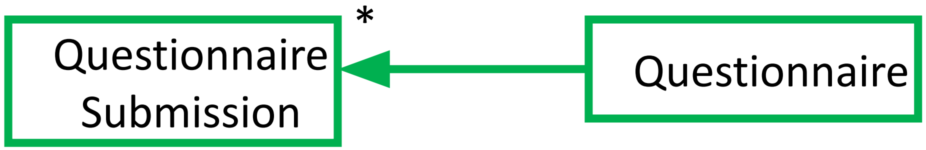
Relationship “regarding”



Submission Questionnaire
@ManyToOne

- ➔ Eager Fetch
- ➔ Cascade in All cases apart from Delete

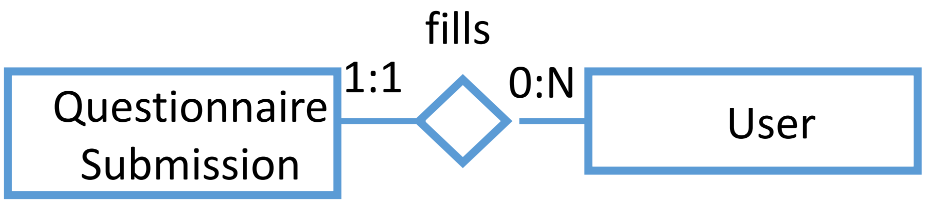




Questionnaire Submission
@OneToMany

- ➔ Lazy Fetch
- ➔ Cascade in All cases apart from Delete
- ➔ Orphan Removal

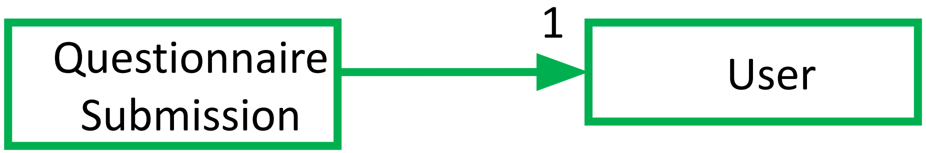
Relationship “fills”



Submission User

@ManyToOne

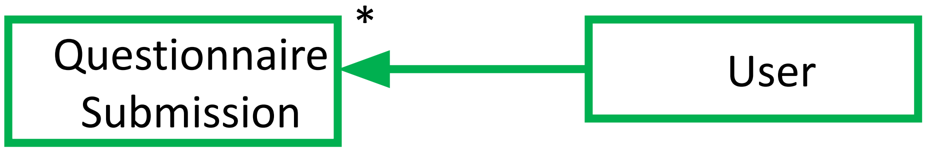
- ➔ Eager Fetch
- ➔ Cascade in All cases apart from Delete



User Submission

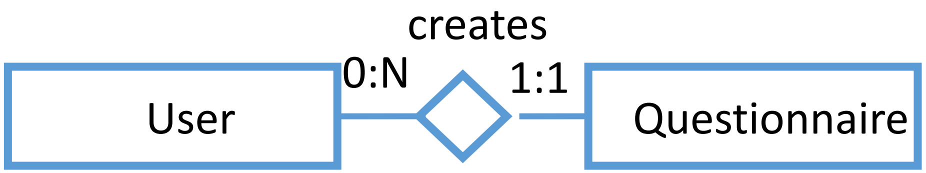
@OneToMany

- ➔ Lazy Fetch
- ➔ Cascade in All cases
- ➔ Orphan Removal



The user submissions are fetched lazily because they are never requested

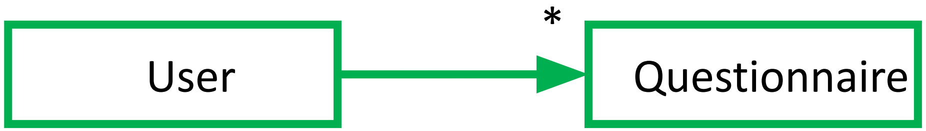
Relationship “creates”



Questionnaire User

@ManyToOne

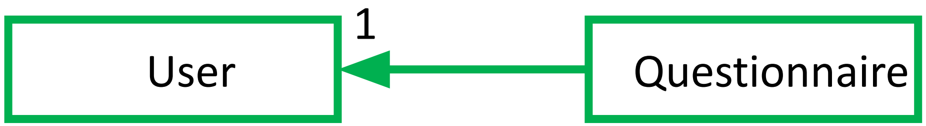
- ➔ Eager Fetch
- ➔ Cascade in All cases apart from Delete



User Questionnaire

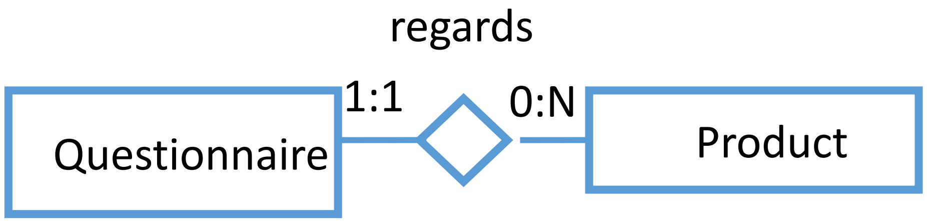
@OneToMany

- ➔ Lazy Fetch
- ➔ Cascade in All cases
- ➔ Orphan Removal



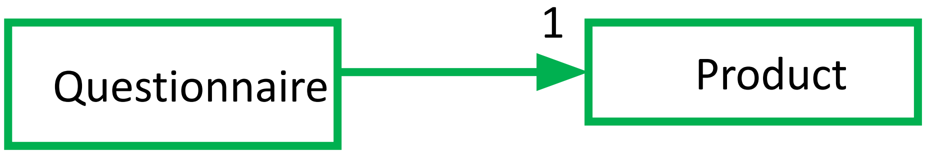
The questionnaire created from the user are fetched lazily because they are never requested

Relationship “regards”



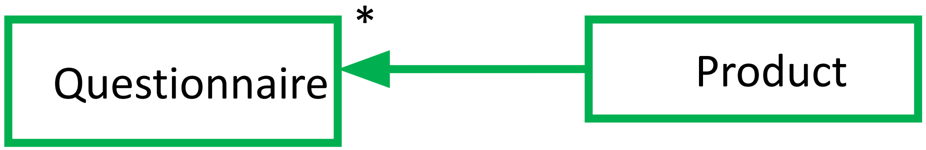
Questionnaire Product
@ManyToOne

- ➔ Eager Fetch
- ➔ Cascade in All cases apart from Delete



Product Questionnaire
@OneToMany

- ➔ Lazy Fetch
- ➔ Cascade in All cases
- ➔ Orphan Removal



Each questionnaire fetches the product eagerly because the product of the day is determined by the questionnaire of the day.

The product of the day should be shown in the homepage.

Entity Answer

```
@Entity
@Table(name = "answer", schema = "db2")
@NamedQueries(
        {
                @ NamedQuery(
                        name = "Product.findReviewsByProduct" ,
                        query = "SELECT a " +
                                "FROM AnswerEntity a " +
                                "WHERE a.questionnaireSubmission.questionnaire.product.id = :productId"
                ),
                @ NamedQuery(
                        name = "Answer.findAnswersByUserAndQuestionnaire" ,
                        query = "SELECT a " +
                                "FROM AnswerEntity a " +
                                "JOIN QuestionnaireSubmissionEntity qs " +
                                "JOIN QuestionnaireEntity q " +
                                "JOIN UserEntity u " +
                                "WHERE u.id = :userId " +
                                "AND q.id = :questionnaireId"
                ),
                @ NamedQuery(
                        name = "Answer.findAnswerByQuestionAndQuestionnaireSubmission" ,
                        query = "SELECT a " +
                                "FROM AnswerEntity a " +
                                "JOIN QuestionnaireSubmissionEntity qs " +
                                "JOIN QuestionEntity q " +
                                "WHERE q.id = :questionId " +
                                "AND qs.id = :questionnaireSubmissionId"
                )
        }
)
public class AnswerEntity implements Serializable {
// attributes & their annotations
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    @Column(nullable = false)
    private String text;

// relationships & their annotations
     @ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
    @JoinColumn(name = "question id")
    private QuestionEntity question;

    @ManyToOne(fetch = FetchType.LAZY, cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH,
CascadeType.DETACH}, optional = false)
    @JoinColumn(name = "QUESTIONNAIRE SUBMISSION ID" , nullable = false)
    private QuestionnaireSubmissionEntity questionnaireSubmission;
…
}
```

Entity Offensive Word

```
@Entity
@Table(name = "offensive_word", schema = "db2")
@NamedQuery(name = "OffensiveWord.findAll", query = "SELECT w FROM OffensiveWordEntity w")
public class OffensiveWordEntity implements Serializable {


// attributes & their annotations

@GeneratedValue(strategy = GenerationType.IDENTITY)
    @Id
    @Column(name = "word", nullable = false)
    private String word;
    ...
}
```

Entity Product

```
@Entity
@Table(name = "product", schema = "db2")
@NamedQueries(
    {
        @NamedQuery(
                name = "Product.findByDate",
                query = "SELECT p " +
                        "FROM ProductEntity p JOIN QuestionnaireEntity q ON p.id = q.product.id " +
                        "WHERE q.date = :date"
        ),
        @NamedQuery(
                name = "Product.findAll",
                query = "SELECT p " +
                        "FROM ProductEntity p"
        ),
    }
)
public class ProductEntity implements Serializable {

// attributes & their annotations

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    @Column(nullable = false)
    private String name;
    @Lob
    @Column
    private byte[] image;

// relationships & their annotations


    @OneToMany(fetch = FetchType.LAZY, mappedBy = "product", cascade = CascadeType.ALL,
orphanRemoval = true)
    private List<QuestionnaireEntity> questionnaire;
    ...
}
```

Entity Question

```
@Entity
@Table(name = "question", schema = "db2")
@NamedQueries(
        {
                @NamedQuery(
                        name = "Question.findByQuestionnaire",
                        query = "SELECT q " +
                                "FROM QuestionEntity q, QuestionnaireEntity qr " +
                                "WHERE q.optional = :optional AND qr.id = :questionnaireId " +
                                "AND q IN (qr.questions) "
                )
        }
)

public class QuestionEntity implements Serializable {

// attributes & their annotations

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    @Column(nullable = false)
    private String text;
    @Column(nullable = false)
    private Boolean optional;


// relationships & their annotations

@ManyToMany(fetch = FetchType.LAZY, cascade = {CascadeType.PERSIST, CascadeType.MERGE,
CascadeType.REFRESH, CascadeType.DETACH})
    @JoinTable(name = "questionnaire to question", schema = "db2",
            joinColumns = @JoinColumn(name = "question id"),
            inverseJoinColumns = @JoinColumn(name = "questionnaire id"))
    private List<QuestionnaireEntity> questionnaires = new ArrayList<>();

    @OneToMany(mappedBy = "question", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<AnswerEntity> answers;
...
}
```

Entity Questionnaire

```
@Entity
@Table(name = "questionnaire", schema = "db2")
@NamedQueries(
        {
                @NamedQuery(
                        name = "Questionnaire.findByDate",
                        query = "SELECT q " +
                                "FROM QuestionnaireEntity q "+
                                "WHERE q.date = :date"
                ),
                @NamedQuery(
                        name = "Questionnaire.findAll",
                        query = "SELECT q " +
                                "FROM QuestionnaireEntity q"
                ),
                @NamedQuery(
                        name = "Questionnaire.findSubmitters",
                        query = "SELECT s.user " +
                                "FROM QuestionnaireSubmissionEntity s "+
                                "WHERE s.points > 0 AND s.questionnaire.id = :id  "
                ),
                @NamedQuery(
                        name = "Questionnaire.findCancellation",
                        query = "SELECT s.user " +
                                "FROM QuestionnaireSubmissionEntity s "+
                                "WHERE s.points = 0 AND s.questionnaire.id = :id  "
                )
        }
)
public class QuestionnaireEntity implements Serializable {
// attributes & their annotations
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;

    @Temporal(TemporalType.DATE)
    @Column(name = "date")
    private Date date;
// relationships & their annotations
    @ManyToMany(fetch = FetchType.LAZY, cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH})
    @JoinTable(name = "questionnaire to question", schema = "db2",
            joinColumns = @JoinColumn(name = "questionnaire id"),
            inverseJoinColumns = @JoinColumn(name = "question id"))
    private List<QuestionEntity> questions = new ArrayList<>();

    @OneToMany(mappedBy = "questionnaire", cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH},
orphanRemoval = true)
    private List<QuestionnaireSubmissionEntity> questionnaireSubmissions = new ArrayList<>();

    @ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH}, optional = false)
    @JoinColumn(name = "USER_ID", nullable = false)
    private UserEntity user;

    @ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH}, optional = false)
    @JoinColumn(name = "product id", nullable = false)
    private ProductEntity product;
    ...
}
```

Entity QuestionnaireSubmission

```
@Entity
@NamedQueries(
        {
                @NamedQuery(
                        name = "QuestionnaireSubmission.getAllUserAnswer",
                        query = "SELECT a " +
                                "FROM QuestionnaireSubmissionEntity s JOIN s.answers a " +
                                "WHERE s.user.id = :userId AND s.questionnaire.id = :questionnaireId"
                ),
                @NamedQuery(
                        name = "QuestionnaireSubmission.findByUserAndQuestionnaire",
                        query = "SELECT qs " +
                                "FROM QuestionnaireSubmissionEntity qs " +
                                "JOIN UserEntity u ON qs.user.id = u.id " +
                                "JOIN QuestionnaireEntity q on qs.questionnaire.id = q.id " +
                                "WHERE u.id = :userId " +
                                "AND q.id = :questionnaireId"
                ),
                @NamedQuery(
                        name = "QuestionnaireSubmission.findLeaderboardByDate",
                        query = "SELECT qs.user.username, qs.points " +
                                "FROM QuestionnaireSubmissionEntity qs " +
                                "JOIN QuestionnaireEntity q ON qs.questionnaire.id = q.id " +
                                "WHERE q.date = :date AND qs.points <> 0 " +
                                "ORDER BY qs.points DESC"
                )
        }
)

@Table(name = "questionnaire submission", schema = "db2")
public class QuestionnaireSubmissionEntity implements Serializable {
// attributes & their annotations
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    @Basic
    @Column(name = "points")
    private Integer points;

// relationships & their annotations
    @ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH}, optional = false)
    @JoinColumn(name = "USER_ID", nullable = false)
    private UserEntity user;

    @ManyToOne(cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH, CascadeType.DETACH}, optional = false)
    @JoinColumn(name = "QUESTIONNAIRE ID", nullable = false)
    private QuestionnaireEntity questionnaire;

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "questionnaireSubmission", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<AnswerEntity> answers = new ArrayList<>();
...
}
```

Entity User

```
@Entity
@NamedQueries(
        {
                @NamedQuery(
                        name = "User.findByUsername",
                        query = "SELECT u " +
                                "FROM UserEntity u " +
                                "WHERE u.username = :username"
                ),
                @NamedQuery(
                        name = "User.findByEmail",
                        query = "SELECT u " +
                                "FROM UserEntity u " +
                                "WHERE u.email = :email"
                )
        }
)
@Table(name = "user", schema = "db2")
public class UserEntity implements Serializable {

// attributes & their annotations
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    @Column(nullable = false, unique = true)
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(name = "email", nullable = false, unique = true)
    private String email;
    private Date lastLogin;
    @Column(nullable = false)
    private Boolean ban;
    private Boolean admin;

// relationships & their annotations
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<QuestionnaireEntity> questionnaires = new ArrayList<>();
    @OneToMany(fetch = FetchType.LAZY, mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<QuestionnaireSubmissionEntity> questionnaireSubmissions = new ArrayList<>();
    ...
}
```

Components

- Client tier
  - admin-panel.html
  - cancellation.html
  - congratulations.html
  - creation-page.html
  - deletion-page.html
  - edit-questionnaire.html
  - inspection-page.html
  - leaderboard.html
  - login.html
  - marketing-questions.html
  - registration.html
  - registration-success.html
  - statistical-questions.html
  - welcome.html

Components

- Web tier
  - AdminPanelController
  - CancellationController
  - CongratulationsController
  - DeletionPageController
  - EditQuestionnaireController
  - HomePageController
  - InspectionPageController
  - LeaderboardController
  - LoginPageController
  - MarketingQuestionsController
  - RegistrationController
  - RegistrationSuccessController
  - StatisticalQuestionsController

Components

- Business Tier
  - UserService
    - Stateless
    - findUser(userId)
    - createUser(username, password, email)
    - findUserByName(username)
    - findUserByEmail(email)
    - checkCredential(username, password)
    - banUser(userId)
  - LeaderboardService
    - Stateless
    - findCurrentLeaderboard()
    - findLeaderboardByDate(Date)

Components

- Business Tier
  - ProductService
    - Stateless
    - getQuestionnaire(questionnaireId)
    - updateQuestion(questionId, newText)
    - removeQuestion(questionId)
    - createQuestionnaire(userId, productId, date)
    - deleteQuestionnaire(questionnaireId)
    - addMarketingQuestion(questionnaireId, questionText)
    - getAllQuestionnaires()
    - getAllSubmitters(questionnaireId)
    - getAllAnswersByUser(userId, questionnaireId)
    - getAllCancellations(questionnaireId)

Components

- Business Tier
  - ProductService
    - Stateless
    - findCurrentProduct()
    - findByDate(date)
    - findProductReviews(product)
    - findAllProducts()

Components

- Business Tier
  - SubmissionService
    - Stateless
    - findCurrentQuestionnaire()
    - findQuestionnaire(date)
    - findMarketingQuestions(questionnaireId)
    - findStatisticalQuestions(questionnaireId)
    - createQuestionnaireSubmission(userId, questionnaireId)
    - findQuestionnaireSubmission(userId, questionnaireId)
    - submitAnswers(questionnareSubmssionId, answers)
    - checkOffensiveWords(answers)
    - cancelQuestionnaireSubmission(userId, questionnaireId)