

POLITECNICO
MILANO 1863

Gamified Market Application

Data Bases 2

Lecturers: Piero Fraternali

Sara Comai

Students: Davide Baroffio

Nicolò Caleffi

Riccardo Zanelli

Specifications (1/2)

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 (2/2)

Gamified consumer data collection

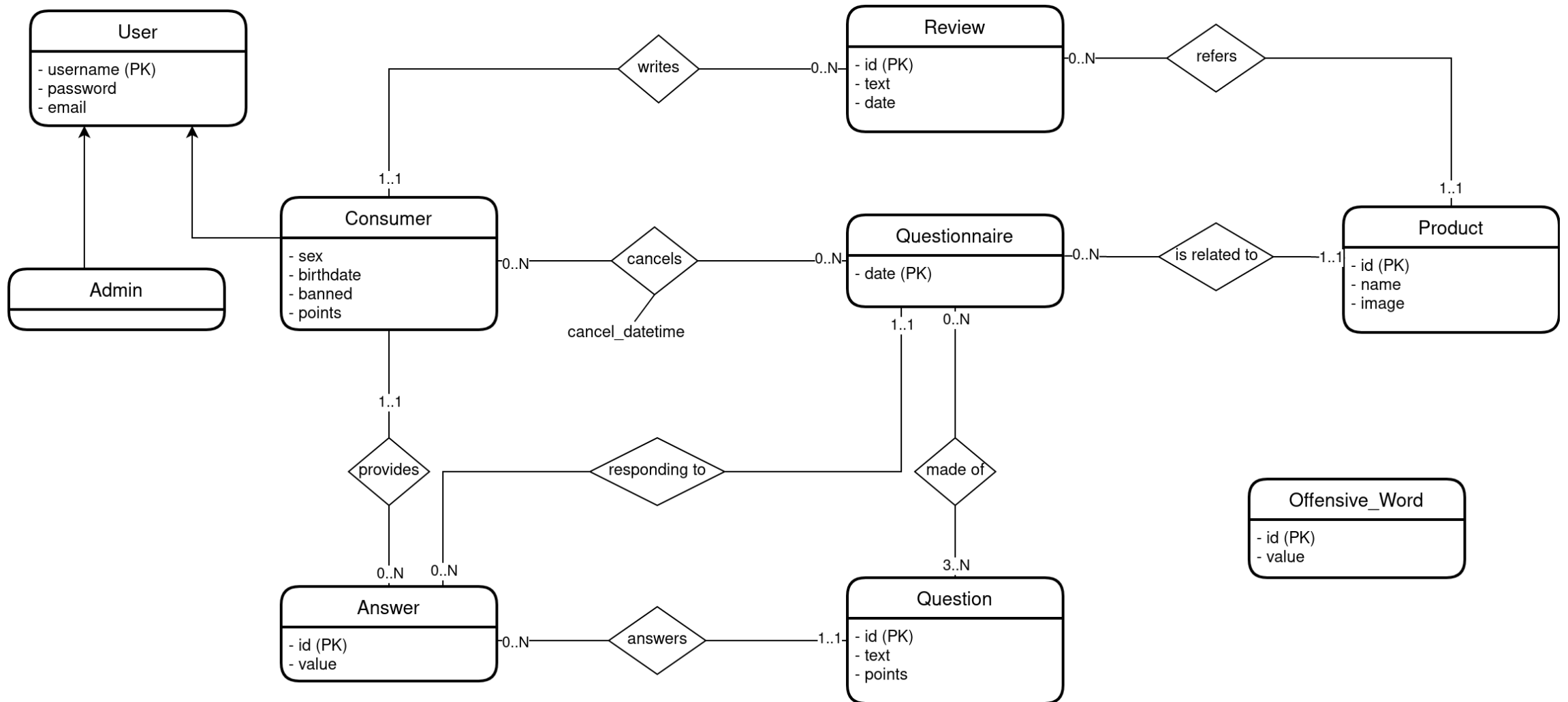
The administrator can access a dedicated application on the same database, which features the following pages :

- 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.

Assumptions

- When a consumer cancels the compilation of a questionnaire multiple times on the same day, only the last login time is stored.
- The reviews are already present in the data base
- The admins are already present in the data base

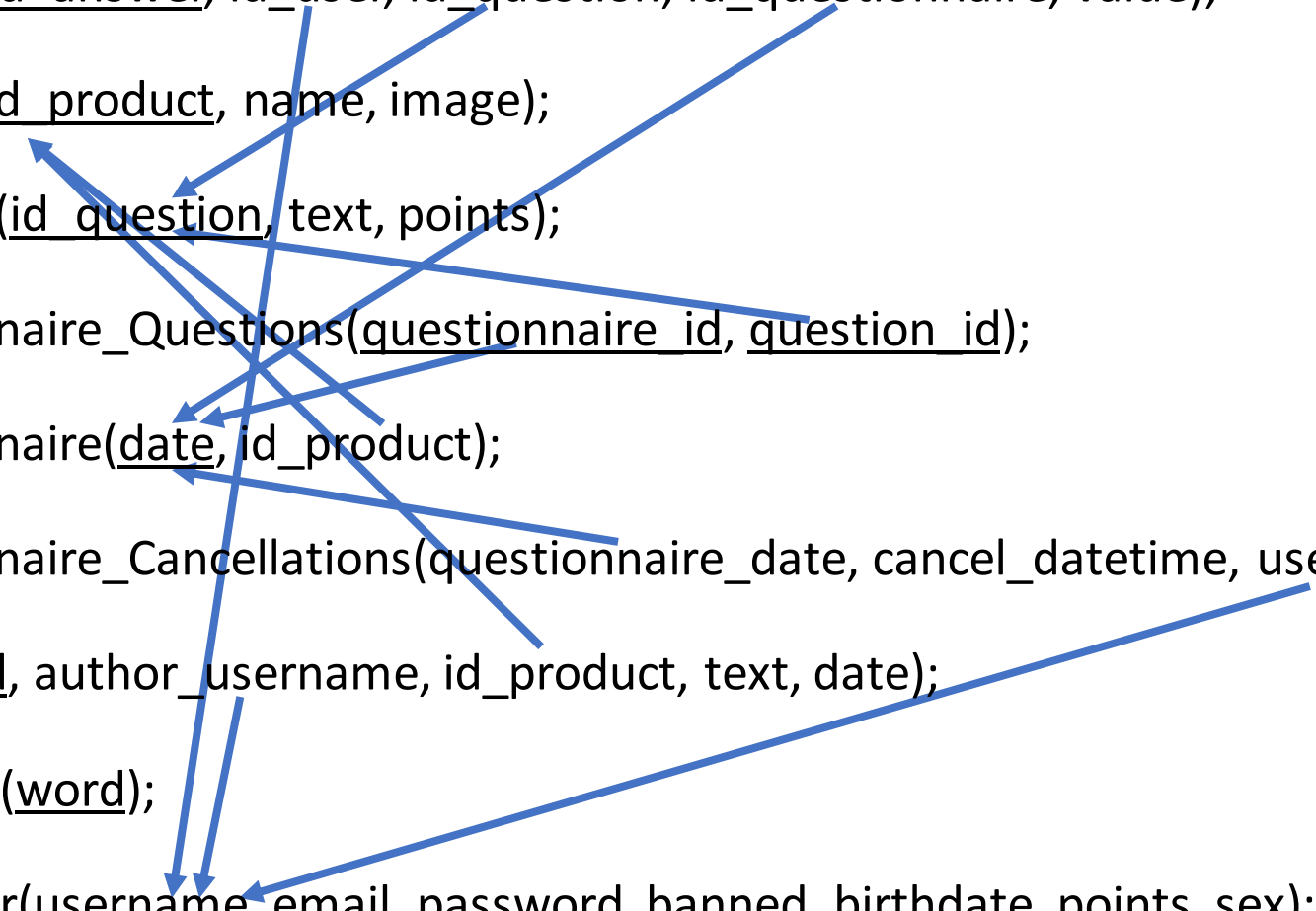
Entity Relationship



Relational Model

- Answer(id_answer, id_user, id_question, id_questionnaire, value);
- Product(id_product, name, image);
- Question(id_question, text, points);
- Questionnaire_Questions(questionnaire_id, question_id);
- Questionnaire(date, id_product);
- Questionnaire_Cancellations(questionnaire_date, cancel_datetime, user_id);
- Review(id, author_username, id_product, text, date);
- BadWord(word);
- Consumer(username, email, password, banned, birthdate, points, sex);
- Admin(username, email, password);

Relational Model

- Answer(id_answer, id_user, id_question, id_questionnaire, value);
 - Product(id_product, name, image);
 - Question(id_question, text, points);
 - Questionnaire_Questions(questionnaire_id, question_id);
 - Questionnaire(date, id_product);
 - Questionnaire_Cancellations(questionnaire_date, cancel_datetime, user_id);
 - Review(id, author_username, id_product, text, date);
 - BadWord(word);
 - Consumer(username, email, password, banned, birthdate, points, sex);
 - Admin(username, email, password);
- 

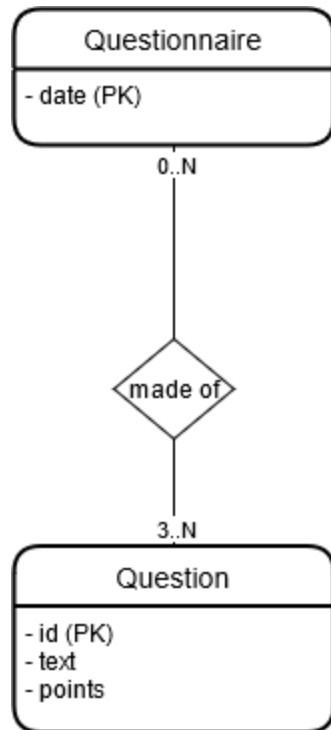
Relationships

Relationship Answer-Question (“answers”)



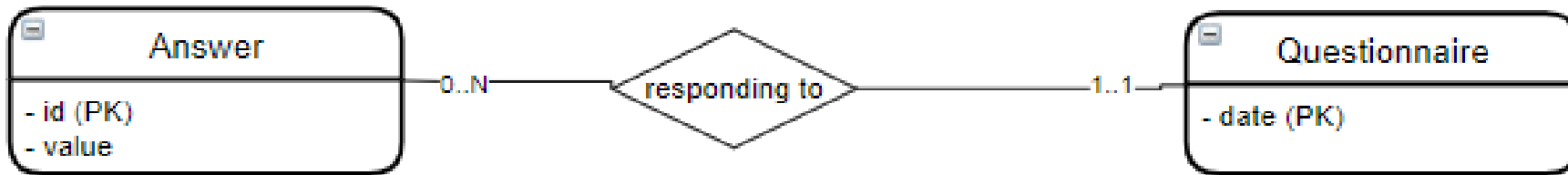
- Answer→Question
@ManyToOne is necessary to retrieve the question to which the answer is responding. Used to link the newly given answer to the question once the questionnaire has been compiled.
- Question→Answer
@OneToMany is necessary to retrieve all the answers associated to the question. Necessary when the admin views the answers each consumer gave to the questions of a questionnaire.

Relationship Question-Questionnaire (“made of”)



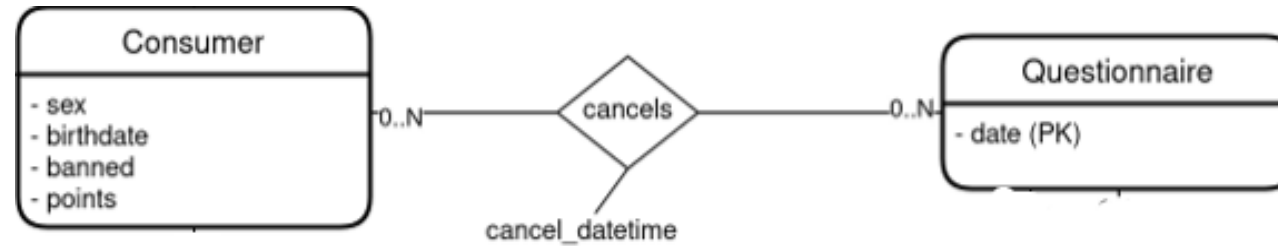
- Questionnaire → Question
@ManyToMany is necessary to retrieve the set of questions related to the questionnaire. Questions' "formulations" are unique in the database and can be associated with multiple questionnaires. Therefore, if multiple questionnaires use the same question, they are all associated with the same row of the Question table.
- Question → Questionnaire
@ManyToMany is necessarily bidirectional. Notice that the cardinality is 3..N: 3 is the minimum number of questions that have to appear in each questionnaire.

Relationship Questionnaire-Answer



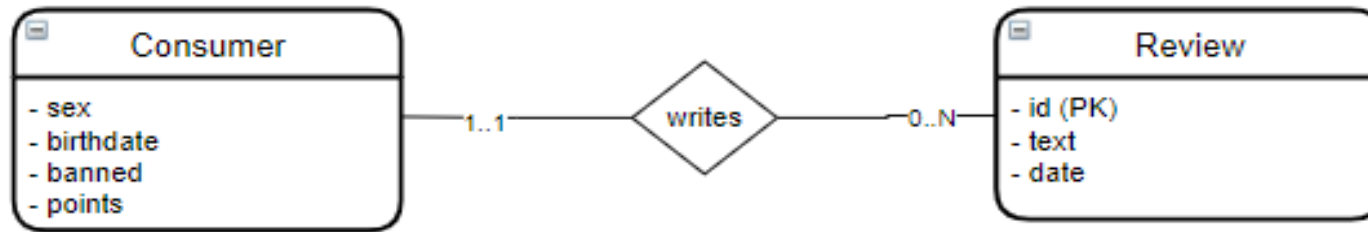
- Questionnaire→Answer
@OneToMany is necessary to retrieve all answers associated to a questionnaire in the admin inspection page.
- Answer→Questionnaire
@ManyToOne is usable to view questionnaire associated with the answer. It is not strictly requested by the specifications but is required to access the questionnaires answered by a given consumer.

Relationship Questionnaire-Consumer



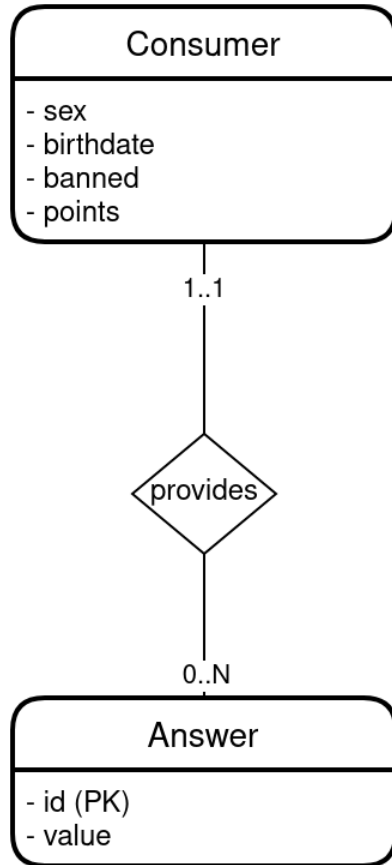
- Questionnaire→Consumer
@ManyToMany is necessary to maintain a list of users who deleted the questionnaires
- Consumer→Questionnaire
@ManyToMany not requested by specifications

Relationship Consumer-Review



- **Consumer→Review**
@OneToMany not requested by the specifications. A future implementation might expect to want to view reviews of logged user.
- **Review→Consumer**
@ManyToOne is necessary to display the username of the user who wrote the review in the consumer home page

Consumer-Answer



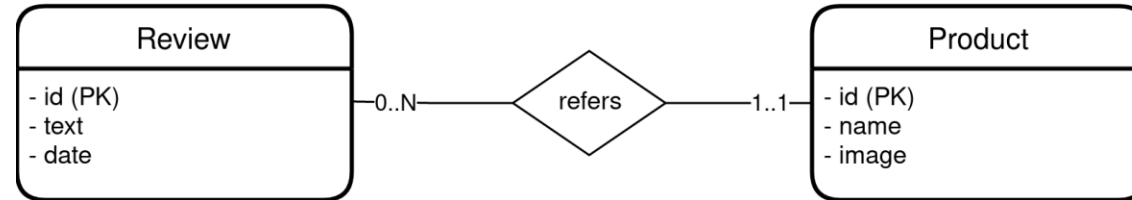
- Consumer → Answer

@OneToMany is not strictly required by the specifications.

- Answer → Consumer

@ManyToOne is used to fetch the author of the answers displayed in the recap page of each questionnaire. On top of that it is necessary to retrieve the customers who submitted a reply to a given questionnaire.

Product-Review



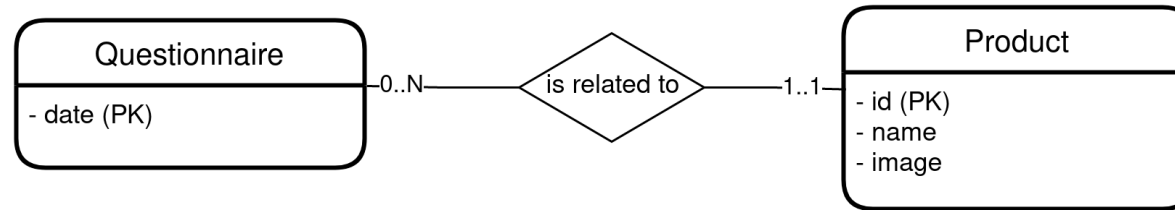
- Review → Product

@ManyToOne is not strictly required by the specifications.

- Product → Review

@OneToMany is used to fetch all the reviews relative to the product-of-the-day so they can be displayed in the consumer home page.

Product-Questionnaire



- Questionnaire → Product

@ManyToOne is used to fetch the product-of-the-day's information, such as the product's name and image, to be displayed in the homepage of the customers. Given that the same product can be elected as product of the day on different dates, therefore it can be associated with multiple questionnaires.

- Product → Questionnaire

@OneToMany is not strictly required by the specifications, but it helps increasing the precision of the mapping to jpa.

JPA Entities

Entity Questionnaire

```
@Entity
@Table(name = "QUESTIONNAIRE", schema = "gmadb")

@NamedQueries({
    // ...
})

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

    // =====
    // Attributes
    // =====

    @Id
    @Temporal(TemporalType.DATE)
    private Date date;

    // =====
    // Relations
    // =====

    @OneToMany(mappedBy="questionnaire", cascade=
        CascadeType.REMOVE)
    private Set<Answer> answers;

    @ManyToOne
    @JoinColumn(name = "PRODUCT_ID")
    private Product product;

    @ManyToMany(cascade = CascadeType.PERSIST)
    @JoinTable (name = "QUESTIONNAIRE_QUESTION",
        joinColumns=@JoinColumn(name =
            "QUESTIONNAIRE_ID"),
        inverseJoinColumns=@JoinColumn(name =
            "QUESTION_ID"))
    private List<Question> questions;
```

```
// =====
// Collections
// =====

@ElementCollection(fetch = FetchType.LAZY)
@CollectionTable(name = "QUESTIONNAIRE_CANCELLATIONS",
    joinColumns = @JoinColumn(name =
        "QUESTIONNAIRE_DATE"))
@MapKeyJoinColumn(name = "USER_ID")
@Column(name = "CANCEL_DATETIME")
private Map<Consumer, Timestamp> cancellations;

// ...
```

Entity Question

```
@Entity
@Table(name="QUESTION", schema = "gmadb")

@NamedQuery(
    // ...
)

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

    // =====
    // Attributes
    // =====

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

    @Column(nullable = false)
    private String text;

    @Column(nullable = false)
    private int points;

    // =====
    // Relations
    // =====

    @OneToMany(mappedBy = "question")
    private Set<Answer> answers;

    @ManyToMany(mappedBy = "questions")
    private Set<Questionnaire> questionnaires;

    // ...
}
```

Entity Answer

```
@Entity
@Table(name = "ANSWER", schema = "gmadb")
@NamedQueries({
    // ...
})
public class Answer implements Serializable{
    private static final long serialVersionUID = 1L;

    // =====
    // Attributes
    // =====

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

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

    // =====
    // Relations
    // =====

    @ManyToOne
    @JoinColumn(name = "USER_ID")
    private Consumer consumer;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "QUESTION_ID")
    private Question question;

    @ManyToOne
    @JoinColumn(name = "QUESTIONNAIRE_ID")
    private Questionnaire questionnaire;

    // ...
}
```

Entity User

```
@MappedSuperclass
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    protected String username;

    @Column(nullable = false)
    protected String password;

    @Column(name = "EMAIL", nullable = false)
    protected String email;
}
```

Entity Consumer

```
@Entity
@NamedQueries({
    @NamedQuery(name="Consumer.getListOrderByPoints",
        query="SELECT c "
            + "FROM Consumer c "
            + "ORDER BY c.points DESC")
})
@Table(name = "CONSUMER", schema = "gmadb")
public class Consumer extends User {
    private static final long serialVersionUID = 1L;

    @Enumerated(EnumType.ORDINAL)
    private Sex sex;

    @Column(nullable = true)
    private Date birthdate;
```

```
    @Column(nullable = false)
    private boolean banned;

    @Column(nullable = false)
    private int points;

    @OneToMany(mappedBy = "author")
    private Set<Review> reviews;

    @OneToMany(mappedBy = "consumer")
    private Set<Answer> answers;

    // ...
}
```

Entity Admin

```
@Entity
@Table(name = "ADMIN", schema = "gmadb")
public class Admin extends User {
    private static final long serialVersionUID = 1L;
}
```

Entity Product

```
@Entity
@NamedQueries({
    @NamedQuery(name="Product.findAll",
        query="SELECT p FROM Product p"),
    @NamedQuery(name="Product.findByName",
        query="SELECT p "
            + "FROM Product p "
            + "WHERE p.name = :nameValue"),
})

@Table(name="PRODUCT", schema = "gmadb")
public class Product implements Serializable {
    private static final long serialVersionUID = 1L;
```

```
    @Column(nullable = false)
    @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;

    @Column(nullable = false, length = 255)
    private String name;

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

    @OneToMany(mappedBy="product")
    private Set<Review> reviews;

    @OneToMany(mappedBy="product")
    private Set<Questionnaire> questionnaires;
```


Entity Review

```
@Entity
@Table(name = "REVIEW", schema = "gmadb")
@NamedQuery(name = "findReviewForProduct", query = "SELECT r
    FROM Review r WHERE r.product.id = :productId")
public class Review {
    @Id
    private long id;

    @Column(nullable = false)
    @Temporal(TemporalType.DATE)
    private Date date;

    @Column(nullable = false)
    private String text;
```

```
@ManyToOne
@JoinColumn(nullable = false)
private Consumer author;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(nullable = false)
private Product product;

@ManyToOne
@JoinColumn(nullable = false)
private Consumer author;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(nullable = false)
private Product product;
```

Entity OffensiveWord

```
@Entity
@NamedQuery(name="OffensiveWord.findAll",
query="SELECT o FROM OffensiveWord o")
@Table(name = "OFFENSIVE_WORD", schema = "gmadb")
public class OffensiveWord {

    @Id
    @Column(name = "WORD", nullable = false)
    private String word;
```

Design Rationale: CascadeType

In general, except for Questionnaire→Answer and Questionnaire→Questions, the CascadeType has not been specified in order to have more control over the management of related entities in the persistence context. In fact, most of the cascade operations (when necessary) are handled by the code depending on the situation.

- Questionnaire→Questions

`@ManyToMany(cascade = CascadeType.PERSIST)`

Questions are always persisted following the creation of the corresponding Questionnaire, without duplicating the already existing questions.

- Questionnaire→Answer

`@OneToMany(cascade = CascadeType.REMOVE)`

In the case in which a questionnaire is deleted, it is explicitly required that the associated answers are deleted from the DB. A trigger is designated to subtract the points to each corresponding user once an answer has been removed.

Business tier components 1/4

- @Stateless CatalogService
 - public Set<Product> getAllProducts()
 - public void saveNewProduct(String productName, byte[] productImg) throws ProductNameAlreadyUsedException
- @Stateless ConsumerHomeService
 - public Product getProductOfTheDay()
 - public Review getOneReview(long productId)
 - public Consumer getConsumer(String username)
- @Stateless LeaderboardService
 - public List<Consumer> getLeaderboard()
 - public List<Consumer> getFullLeaderboard()

Business tier components 2/4

- @Stateless LoginService
 - public User requestLogin(String username, String password, Boolean adminLogin) throws UserNotFound, WrongPassword
 - public User requestAdminLogin(String username, String password) throws UserNotFound, WrongPassword
 - public User requestConsumerLogin(String username, String password) throws UserNotFound, WrongPassword
- @Stateless QuestionnaireCompilerService
 - public Questionnaire findTodaysQuestionnaire()
 - public void persistAnswers(Set<Answer> answers) throws OffensiveWordException
 - public void banUser(String username)
 - public boolean couldSubmit(Consumer c)
 - public void cancelQuestionnaire(Consumer c, LocalDateTime d, Questionnaire q) throws Exception
- @Sateful QuestionnaireCreationService
 - public List<String> getQuestions()
 - public void setQuestions(List<String> questions)
 - public void setDate(Date date)
 - public void setProduct(long productId) throws ProductNotFoundException
 - public Date getDate()
 - public void createQuestionnaire() throws DateNotAvailableException, QuestionnaireCreationException
 - public void remove()

Business tier components 3/4

- @Stateless QuestionnaireHistoryService
 - public List<Questionnaire> getPage(int page, int pageSize)
 - public List<Questionnaire> getPlannedPage(int page, int pageSize)
 - public void removeQuestionnaire(Date questionnaireDate)
- @Stateless RecapService
 - public Questionnaire getInformation(String string) throws ParseException
- @Stateless RegistrationService
 - public void registerConsumer(Consumer c) throws ConsumerAlreadyRegisteredException

Business tier components 4/4

The majority of the beans are Stateless EJBs because all the method calls are independent of the session state. In some cases, the session state could be directly saved in the session object because the few information kept is not too large in size. While when creating a questionnaire, a stateful EJB was chosen so to take advantage of its passivation.

A singleton was used to keep track of the number of the existing questionnaire so that their paging (in the admin home) wouldn't require to query the database, in order to check for the availability of more questionnaires.

Thank you for your
attention