Data Bases 2

Gamified Marketing Application

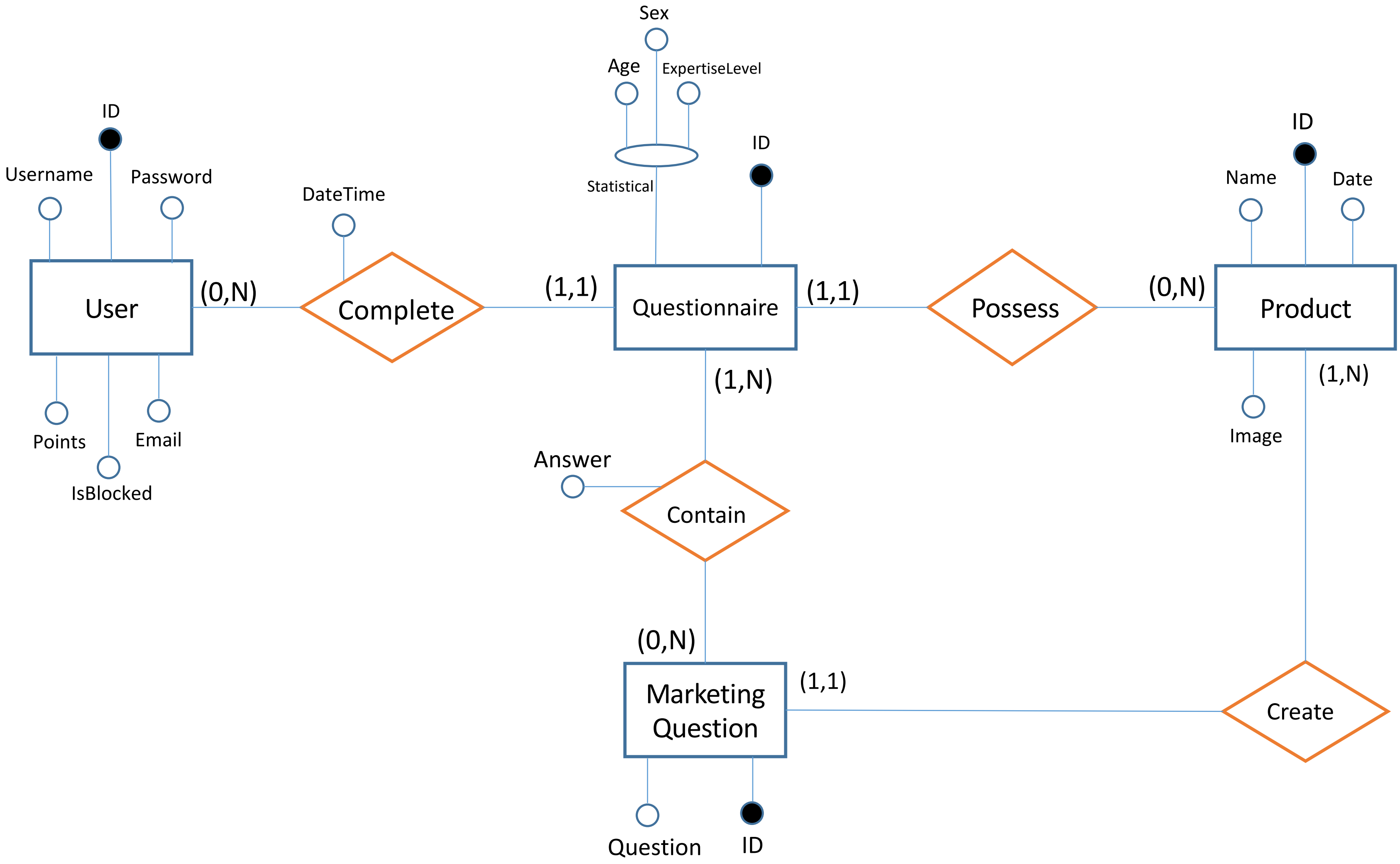
POLITECNICO
MILANO 1863

Zhang Yuedong

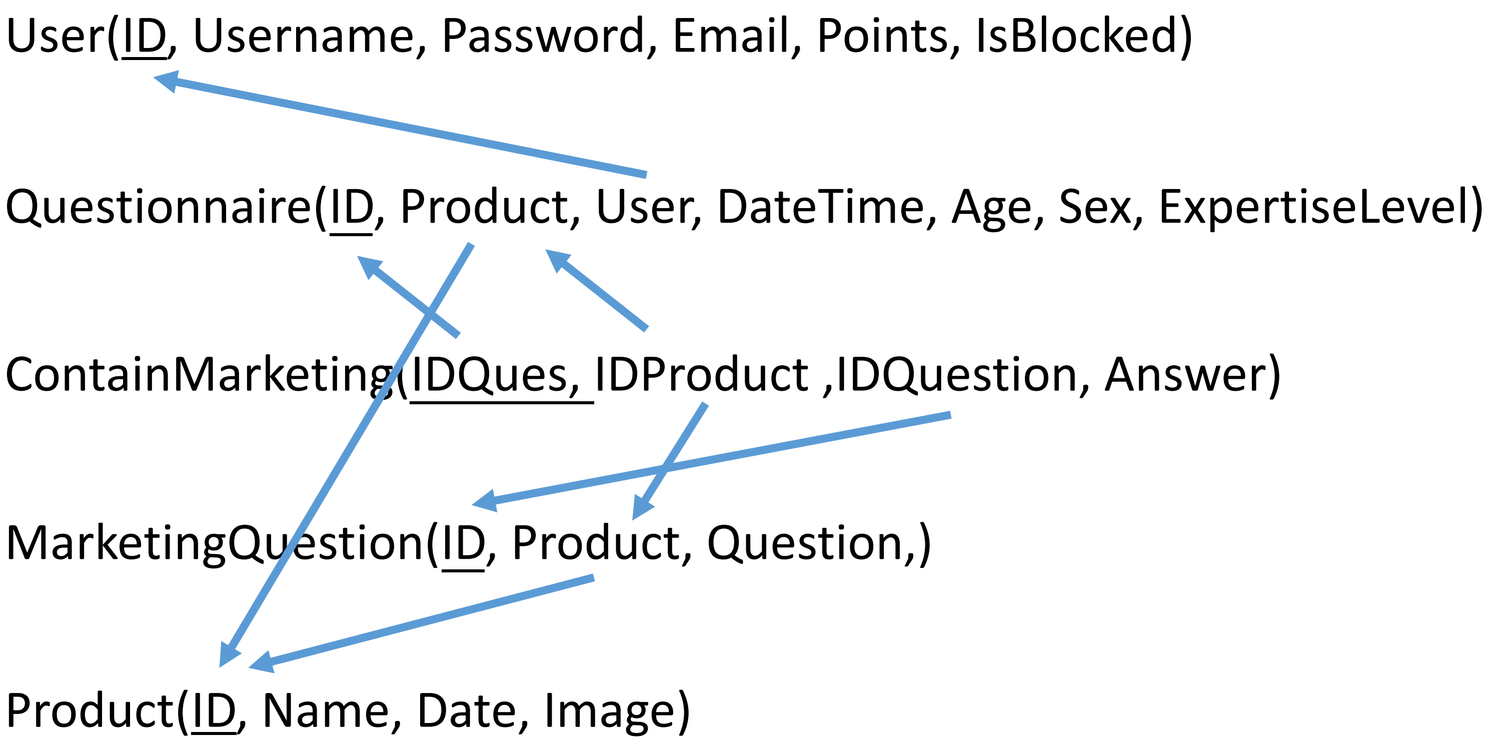
Specifications

- 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:
    - One point is assigned for every answered question of section 1 (remember that the number of questions can vary in different questionnaires).
    - 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).
- 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
        - 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.

Entity Relationship Model – ER Model



Relational model



User(ID, Username, Password, Email, Points, IsBlocked)

Questionnaire(ID, Product, User, DateTime, Age, Sex, ExpertiseLevel)

ContainMarketing(IDQues, IDProduct ,IDQuestion, Answer)

MarketingQuestion(ID, Product, Question,)

Product(ID, Name, Date, Image)

Motivations

To map the relationship between Entity Questionnaire and MarketingQuestion, I used a foreign key "IDProduct" as the constraint in ContainMarketing.

So that can avoid a new insertion to a Questionnaire but answering the Question corresponding to another product.

```
CREATE TABLE User(
   ID INTEGER PRIMARY KEY auto_increment,
   Username varchar(45) not null unique,
   Password varchar(45) not null,
   Email varchar(255) not null,
   Point integer not null DEFAULT 0 CHECK (Point>=0),
   IsBlocked boolean not null DEFAULT FALSE,
   CONSTRAINT mailcheck CHECK (regexp_like(Email,'^(\\S+)\\@(\\S+).(\\S+)$'))
);
```

```
CREATE TABLE Product(
   ID INTEGER PRIMARY KEY auto_increment,
   Name varchar(45) unique not null ,
   Date DATE not null default (CURRENT_DATE),
   Image longblob not null
);
```

```
CREATE TABLE Questionnaire(
   ID integer NOT NULL auto_increment PRIMARY KEY,
   Product integer not null,
   User integer not null,
   Datetime TIMESTAMP not null default CURRENT_TIMESTAMP(),
   Age integer,
   Sex char check (Sex in ('M','F')),
   ExpertiseLevel varchar(8) check (ExpertiseLevel in ('LOW','MEDIUM','HIGH')),
   UNIQUE (ID, Product),
   UNIQUE (Product, User),
   FOREIGN KEY (Product) references Product(ID)
                  ON UPDATE CASCADE
                  ON DELETE CASCADE,
   FOREIGN KEY (User) references User(ID)
                  ON UPDATE CASCADE
                  ON DELETE CASCADE
);
```

```
CREATE TABLE MarketingQuestion(
   ID INTEGER PRIMARY KEY auto_increment,
   Product integer,
   Question varchar(512) NOT NULL ,
   UNIQUE (ID, Product),
   UNIQUE (Question, Product),
   FOREIGN KEY (Product) references Product(ID)
                  ON UPDATE CASCADE
                  ON DELETE CASCADE
);
```

```
CREATE TABLE ContainMarketing(
   IDQues integer,
   IDProduct integer,
   IDQuestion integer,
   Answer varchar(512) not null,
   PRIMARY KEY (IDQues, IDProduct, IDQuestion),
   FOREIGN KEY (IDQues, IDProduct) references Questionnaire(ID, Product)
                  ON UPDATE CASCADE
                  ON DELETE CASCADE,
   FOREIGN KEY (IDProduct, IDQuestion) references MarketingQuestion(Product, ID)
                  ON UPDATE CASCADE
                  ON DELETE CASCADE
);
```

```
CREATE TABLE Administrator(
   ID INTEGER PRIMARY KEY auto_increment,
   Name varchar(45) NOT NULL UNIQUE ,
   Password varchar(45) NOT NULL UNIQUE
);
```

```
CREATE TABLE OffensiveWord(
   ID INTEGER PRIMARY KEY auto_increment,
   OffensiveWord varchar(45) not null UNIQUE
);
```

Trigger

```
/*
 The user’s account is blocked,
 so that no questionnaires can be filled in by such account in the future.
 */
CREATE TRIGGER AccountIsBlocked1
BEFORE insert on Questionnaire FOR EACH ROW
BEGIN
   if (SELECT IsBlocked FROM User WHERE User.ID = NEW.User) is True
   then SIGNAL SQLSTATE '45000'  SET MESSAGE_TEXT = 'Your account is blocked';
   end if;
END;
```

```
/*
 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.
*/
CREATE TRIGGER CheckOffensiveWord
BEFORE insert on ContainMarketing FOR EACH ROW
BEGIN
   if TRUE in (SELECT IsBlocked FROM User
      WHERE ID in (SELECT User FROM Questionnaire
      WHERE Questionnaire.ID = NEW.IDQues))
   then
      SIGNAL SQLSTATE '45000'  SET MESSAGE_TEXT = 'Your account is blocked';
   else
      if (SELECT COUNT(*) FROM OffensiveWord WHERE NEW.Answer REGEXP CONCAT('\(',OffensiveWord,'\)')) > 0
      then
         SIGNAL SQLSTATE '45000'  SET MESSAGE_TEXT = 'Your submitted answer contain offensive words';
      end if;
   end if;
END;
```

```
/*
   AddPointsForMarketingSection
   One point is assigned for every answered question of section 1
*/
CREATE TRIGGER AddPointsForMarketingSection
AFTER insert on ContainMarketing FOR EACH ROW
BEGIN
   UPDATE User
      SET Point = Point + 1
      WHERE ID in (SELECT User FROM Questionnaire WHERE Questionnaire.ID = NEW.IDQues);
END;
```

```
/*
 Two points are assigned for every answered optional question of section 2
 */
CREATE TRIGGER AddPointsForStatisticalSection
AFTER insert on Questionnaire FOR EACH ROW
BEGIN
   IF NEW.Sex is not null
   then
      UPDATE User
         SET Point = Point + 2
      WHERE User.ID = NEW.User;
   end if;
   IF NEW.Age is not null
   then
      UPDATE User
         SET Point = Point + 2
      WHERE User.ID = NEW.User;
   end if;
   IF NEW.ExpertiseLevel is not null
   then
      UPDATE User
         SET Point = Point + 2
      WHERE User.ID = NEW.User;
   end if;
END;
```

```
/*
   A CREATION page for inserting the product of the day for the current date or for a posterior date
 */
CREATE TRIGGER OnlyCreateProductForCurrentORPosteriorDate
BEFORE insert on Product FOR EACH ROW
BEGIN
   if NEW.Date < CURRENT_DATE()
   then
      SIGNAL SQLSTATE '45000'  SET MESSAGE_TEXT = 'You can only inserting
               the product for the current date or for a posterior date';
   end if;
END;
```

```
/*
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.
*/
CREATE TRIGGER DeletionQuestionnaireOnlyForPrecedingDate
BEFORE delete on Questionnaire FOR EACH ROW
BEGIN
    if DATE(OLD.Datetime) >= CURRENT_DATE
    then
        SIGNAL SQLSTATE '45000'  SET MESSAGE_TEXT = 'Delete only for the preceding date.';
   end if;
END;
```

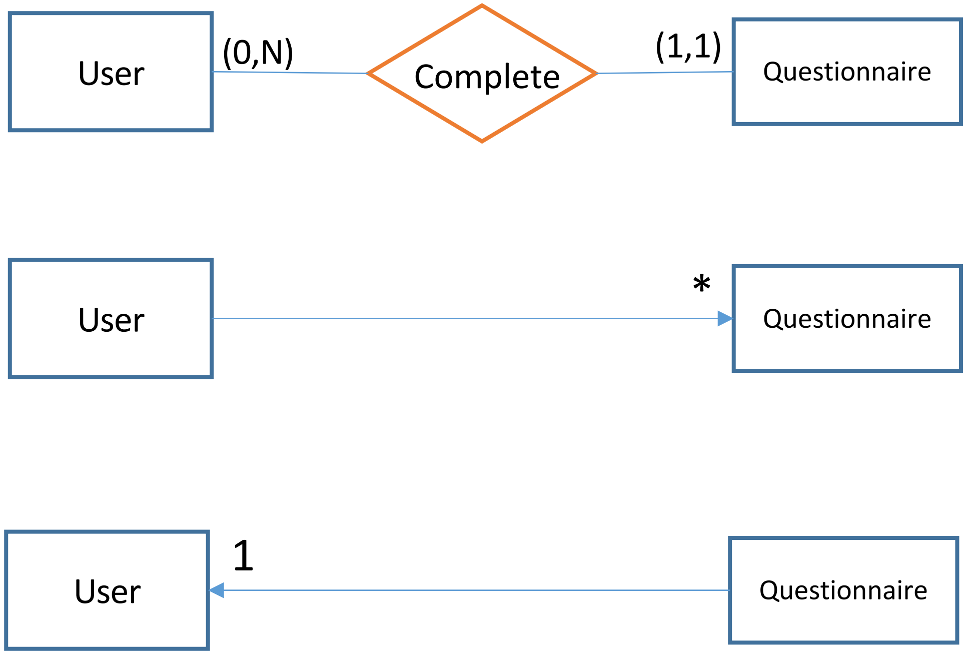
```
/*
A DELETION page for ERASING the questionnaire data and
the related responses and points of all users who filled in the questionnaire.
 */
CREATE TRIGGER ReducePointsAfterDeletionQuestionnaire
AFTER delete on Questionnaire FOR EACH ROW
BEGIN
   if OLD.Sex is not null
   then
      UPDATE User
         SET Point = Point - 2
      WHERE User.ID = OLD.User ;
   end if;

   if OLD.Age is not null
   then
      UPDATE User
         SET Point = Point - 2
      WHERE User.ID = OLD.User;
   end if;

   if OLD.ExpertiseLevel is not null
   then
      UPDATE User
         SET Point = Point - 2
      WHERE User.ID = OLD.User;
   end if;
END;
```

```
CREATE TRIGGER ReducePointsAfterDeletionContainMarketing
AFTER DELETE on ContainMarketing FOR EACH ROW
BEGIN
   UPDATE User
      SET Point = Point - 1
         WHERE User.ID = (SELECT User FROM Questionnaire WHERE Questionnaire.ID = OLD.IDQues);
END;
```

Motivations

I used the Incremental approach to calculate the point.
And other motivations included in the comment above the codes.

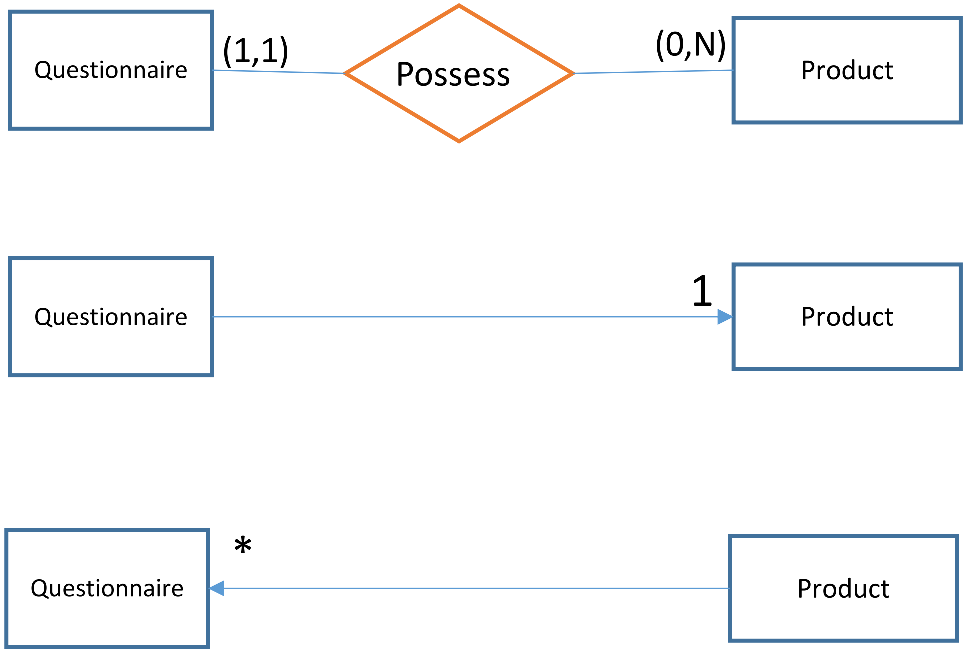
Relationship ’Complete’



User -> Questionnaire
@OneToMany is necessary to get the Questionnaires of a user in Admin Inspection Page

Questionnaire -> User
@ManyToOne is necessary to get the User Info of a Questionnaire in Admin Deletion Page and User Home Page

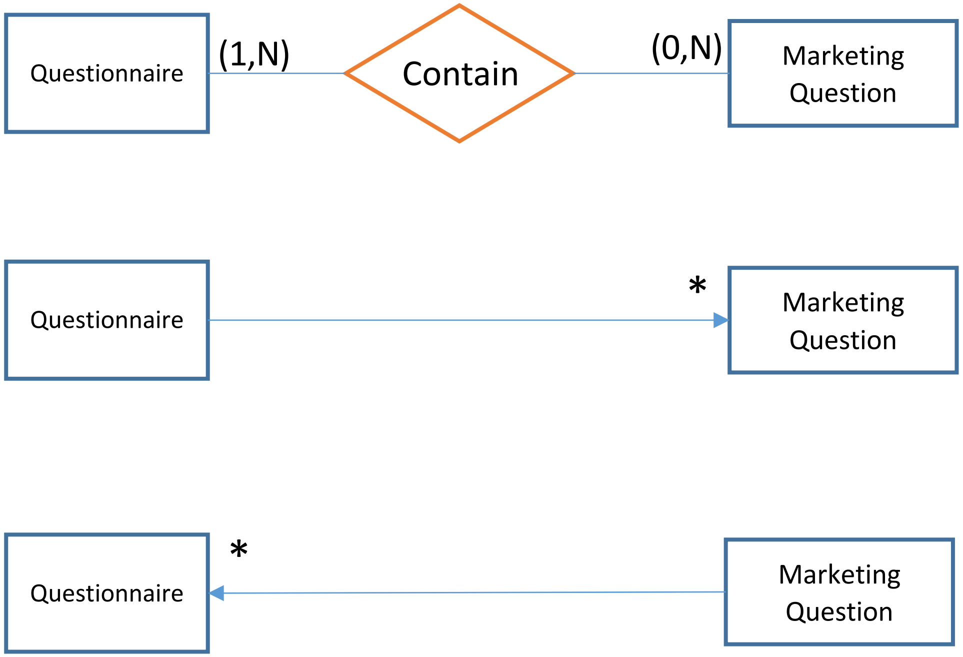
Relationship ‘Possess’



Questionnaire -> Product
@OneToMany is necessary to get the Product Info of a Product in Admin Deletion Page

Product -> Questionnaire
@ManyToOne is not necessary but implemented for simplicity

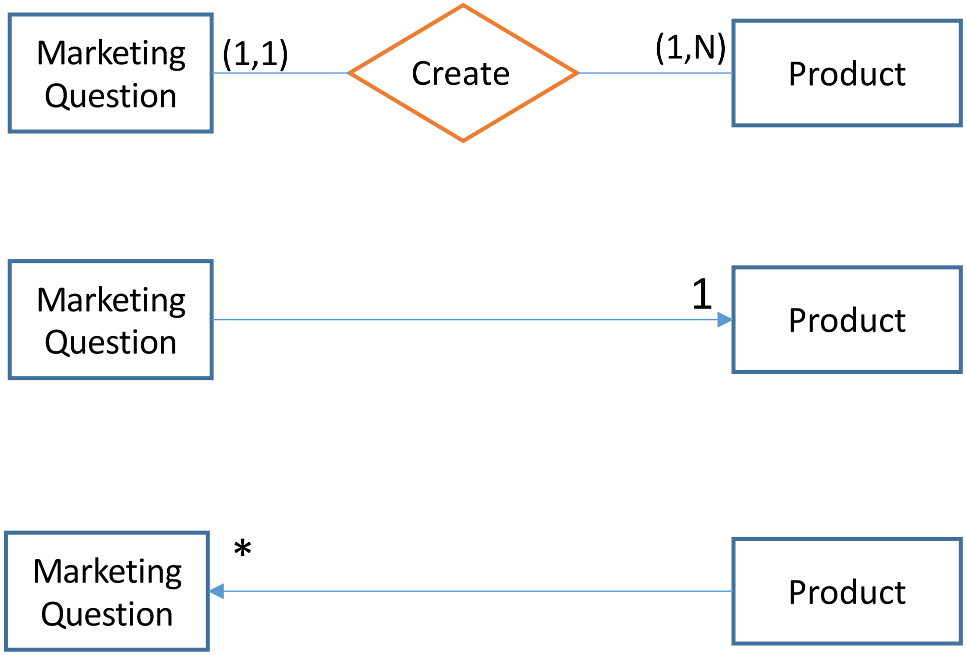
Relationship ‘Contain’



Questionnaire -> MarketingQuestion
@ManyToMany is necessary to get/set the MarketingQuestions/Answer Map of a Questionnaire When submitting the Questionnaire with MarketingQuestions' answers or show the Questionnaire's detail.

MarketingQuestion -> Questionnaire
@ManyToMany is necessary to get the Questionnaire/Answer Map in the User Home Page.

Relationship ‘Create’



MarketingQuestion -> Product
@ManyToOne is necessary when register a new MarketingQuestion to indicate the product for this Question.

Product -> Marketing Question
@OnyToMany is necessary to display the Question of a Product when a user is prepared to answer them.

Entity User

```
@Entity
@Table(name = "User", schema = "db_gamified_marketing_application")
@NamedQuery(name = "User.checkCredentials", query = "SELECT r FROM User r  WHERE r.username = ?1 and r.password = ?2")
@NamedQuery(name = "User.findAllUsersDescByPoints", query = "SELECT u FROM User u order by u.point DESC ")


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

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

   private String username;

   private String password;

   private String email;

   private Integer point;

   private Boolean IsBlocked;

   @OneToMany(fetch = FetchType.EAGER, mappedBy = "User")
   @OrderBy("Datetime ASC")
   List<Questionnaire> questionnaires;
```

Motivations

- Named query User.checkCredentials for credential verification
- Named query User.findAllUsersDescByPoints for find all users

-
  ```
  @OneToMany(fetch = FetchType.EAGER, mappedBy = "User")
  @OrderBy("Datetime ASC")
  List<Questionnaire> questionnaires;
  ```
  To navigate the questionnaires immediately, so I used the EAGER as the FetchType, and asc order the result by the Datetime.

Entity Questionnaire

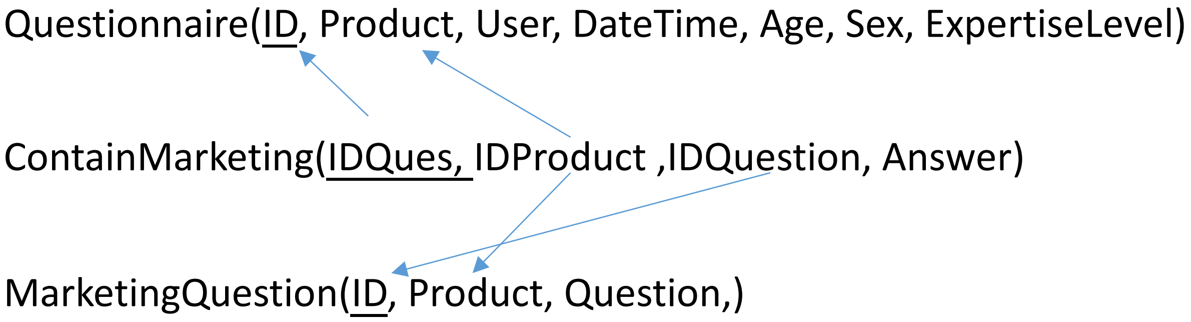
```
@Entity
@Table(name = "Questionnaire", schema = "db_gamified_marketing_application")
@NamedQuery(name = "Questionnaire.findAllQuestionnaireOrderByDatetime", query = "SELECT q FROM Questionnaire q order by q.datetime ASC ")

public class Questionnaire implements Serializable {

    private static final long serialVersionUID = 1L;

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

    private Timestamp datetime;

    private Integer age;

    private String sex;

    private String expertiseLevel;

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

    @ManyToOne
    @JoinColumn(name = "User")
    private User user;

    @ElementCollection(fetch = FetchType.EAGER)
    @CollectionTable(name = "ContainMarketing",joinColumns = @JoinColumn(name = "IDQues"))
    @MapKeyJoinColumn(name = "IDQuestion")
    @MapKeyJoinColumn(name = "IDProduct",referencedColumnName = "Product")
    @Column(name = "Answer")
    private Map<MarketingQuestion, String> questionAnswerMap;
```

Motivations

- NamedQuery Questionnaire.findAllQuestionnaireOrderByDatetime for find all Quetionnaire and order the result by the Datetime

-
  ```
  @ElementCollection(fetch = FetchType.EAGER)
  @CollectionTable(name = "ContainMarketing",joinColumns = @JoinColumn(name = "IDQues"))
  @MapKeyJoinColumn(name = "IDQuestion")
  @MapKeyJoinColumn(name = "IDProduct",referencedColumnName = "Product")
  @Column(name = "Answer")
  private Map<MarketingQuestion, String> questionAnswerMap;
  ```

  This is the relationship map from Qustionnaire to Question/Answer



Questionnaire(ID, Product, User, DateTime, Age, Sex, ExpertiseLevel)

ContainMarketing(IDQues, IDProduct ,IDQuestion, Answer)

MarketingQuestion(ID, Product, Question,)

Entity Product

```
@Entity
@Table(name = "Product", schema = "db_gamified_marketing_application")
@NamedQuery(name = "Product.findProductsByDate", query = "SELECT p FROM Product p where p.date = :date")
@NamedQuery(name = "Product.findProductByName", query = "SELECT p FROM Product p where p.name = :name")

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

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

   private String name;

   private Date date;

   @Lob
   private byte[] image;

   @OneToMany(fetch = FetchType.LAZY, mappedBy = "Product")
   private List<Questionnaire> questionnaires;

   @OneToMany(fetch = FetchType.EAGER, cascade = {CascadeType.PERSIST,CascadeType.REFRESH}, mappedBy = "Product")
   private List<MarketingQuestion> marketingQuestionsList;
```

Motivations

- NamedQuery Product.findProductsByDate To find Product by Product date
- NamedQuery Product.findProductByName To find Product by Product name

- @OneToMany(fetch = FetchType.LAZY, mappedBy = "Product")
  private List<Questionnaire> questionnaires;
  This is a map to the Questionnaires, As mentioned before, it's not necessary, but implemented for simplicity.

- @OneToMany(fetch = FetchType.*EAGER*, cascade = {CascadeType.*PERSIST*,CascadeType.*REFRESH*}, mappedBy = "Product")
  private List<MarketingQuestion> marketingQuestionsList;
  It needs the cascade the PERSIST operation when submitting the Marketing Questions and cascades the REFRESH operation to get the MarketingQuestion on User Home Page.

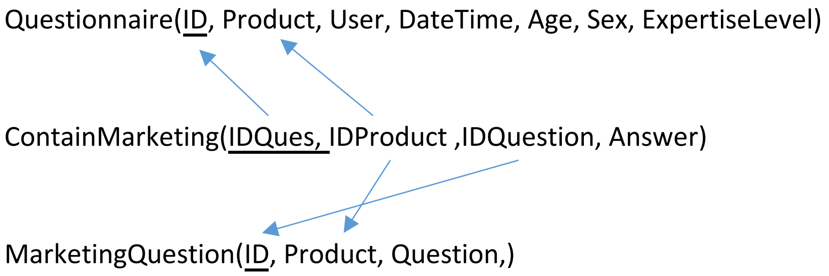
Enetity MarketingQuestion

```
@Entity
@Table(name = "MarketingQuestion", schema = "db_gamified_marketing_application")

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

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

   private String question;

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

   @ElementCollection
   @CollectionTable(name = "ContainMarketing",joinColumns = @JoinColumn(name = "IDQuestion"),foreignKey = @ForeignKey(name = "IDProduct"))
   @MapKeyJoinColumn(name = "IDQues")
   @MapKeyJoinColumn(name = "IDProduct",referencedColumnName = "Product")
   @Column(name = "Answer")
   private Map<Questionnaire, String> questionnaireAnswerMap;
```

Motivations

```
@ElementCollection
@CollectionTable(name = "ContainMarketing",joinColumns = @JoinColumn(name = "IDQuestion"),foreignKey = @ForeignKey(name = "IDProduct"))
@MapKeyJoinColumn(name = "IDQues")
@MapKeyJoinColumn(name = "IDProduct",referencedColumnName = "Product")
@Column(name = "Answer")
private Map<Questionnaire, String> questionnaireAnswerMap;
```

This is the map from MarketingQuestion to Questionnaire/Answer. It can map the Questions' Answers by a Questionnaire.
This little like the opposite map from Questionnaire to Question/answer as above mentioned.



Questionnaire(ID, Product, User, DateTime, Age, Sex, ExpertiseLevel)

ContainMarketing(IDQues, IDProduct ,IDQuestion, Answer)

MarketingQuestion(ID, Product, Question,)

Components

- Client components
  - AdminCreateProductPage
  - AdminDeletionPage
  - AdminHomePage
  - AdminInspectionPage
  - AdminLogin
  - CancelQuestionnaire
  - CheckLogin
  - GoToHomePage
  - GoToLeaderboardPage
  - GoToQuestionnaireMarketingSection
  - GoToQuestionnaireStatisticalSection
  - Logout
  - Register
  - SubmitQuestionnaire

- Business Components
  - UserService(@Stateless)
    - registerNewClient
    - checkCredentials
    - checkAdminCredentials
    - findUsersAnsweredQuestionsOnADay
    - findUsersCancelledQuestionnaireOnADay
    - toBlockAccount
  - QuestionnaireService(@Stateless)
    - submitAQuestionnaire
    - cancelAQuestionnaire
    - getAllQuestionnaire
    - deleteQuestionnaire
  - ProductService(@Stateless)
    - createAProduct
    - findProductByName
    - findProductsByDate

Motivations

All the Bussiness components are stateless, because all client requests are served independently and update the database, no main memory conversational state need to be maintained.