DB2 Project

Gamified marketing application

Group Members:

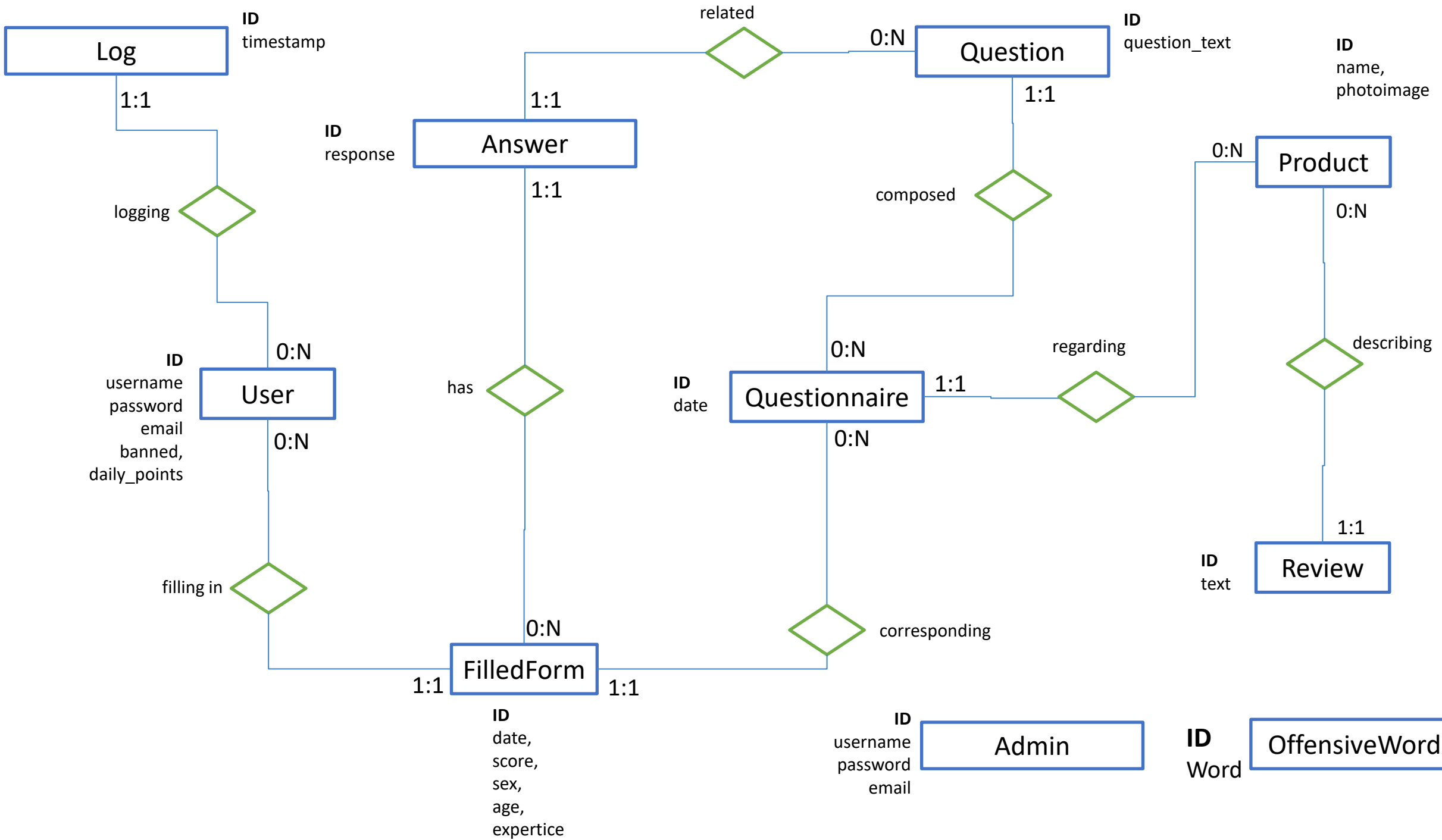
Ludovico Righi

Enrico Gherardi

Arslan Ali

Academic Year 2020/2021

Entity Relationship Diagram


Log
ID
timestamp
1:1
logging
0:N
ID
username
password
email
banned,
daily_points
User
0:N
filling in
1:1
FilledForm
1:1
ID
date,
score,
sex,
age,
expertice
0:N
has
1:1
Answer
ID
response
related
0:N
Question
ID
question_text
1:1
composed
0:N
Questionnaire
ID
date
1:1
regarding
0:N
corresponding
0:N
Product
ID
name,
photoimage
0:N
describing
1:1
Review
ID
text
Admin
ID
username
password
email
OffensiveWords
ID
Word

Motivations

- We have decided to use a one-to-many relationship between product and questionnaire, so that a product can be related to different questionnaires in different days; using a one-to-one would have been good anyway, but more restrictive.

- We have inserted the fields sex, age and expertice into the FilledForm, so that a user can modify his choices everytime that he fills in a form, as required in the specifications. Since the number of these field is fixed to three, we decided to insert them into the FilledForm table, instead of using another one-to-one relationship with FilledForm.

SQL Data Definition Language

```sql
create table usr(
    id INTEGER UNSIGNED AUTO_INCREMENT,
    username VARCHAR(20) UNIQUE NOT NULL,
    email VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(45) NOT NULL,
    banned TINYINT(1) DEFAULT 0,
    daily_points INTEGER DEFAULT 0,
    PRIMARY KEY (id)
);

create table admn(
    id INTEGER UNSIGNED AUTO_INCREMENT,
    username VARCHAR(20) UNIQUE NOT NULL,
    email VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(45) NOT NULL,
    PRIMARY KEY (id)
);

create table offensive_word(
    id INTEGER UNSIGNED AUTO_INCREMENT,
    word VARCHAR(20) UNIQUE NOT NULL,
    PRIMARY KEY (id)
);
```

```sql
create table log(
    id INTEGER UNSIGNED AUTO_INCREMENT,
    user_id INTEGER UNSIGNED,
    ts timestamp NOT NULL,
    date_log date NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES usr(id) ON DELETE CASCADE
);

create table product (
    id INTEGER UNSIGNED AUTO_INCREMENT,
    prod_name VARCHAR(30) NOT NULL,
    photoimage longblob,
    PRIMARY KEY (id)
);

create table review (
    id INTEGER UNSIGNED AUTO_INCREMENT,
    product_id INTEGER UNSIGNED,
    review_text VARCHAR(400) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (product_id) REFERENCES product(id) ON DELETE CASCADE
);
```

```sql
create table questionnaire (
    id INTEGER UNSIGNED AUTO_INCREMENT,
    date_questionnaire date UNIQUE NOT NULL,
    product_id INTEGER UNSIGNED,
    PRIMARY KEY (id),
    FOREIGN KEY (product_id) REFERENCES product(id) ON DELETE CASCADE
);

create table filled_form (
    id INTEGER UNSIGNED AUTO_INCREMENT,
    user_id INTEGER UNSIGNED,
    questionnaire_id INTEGER UNSIGNED,
    date_form date NOT NULL,
    age INTEGER UNSIGNED DEFAULT NULL,
    sex VARCHAR(10) DEFAULT NULL,
    expertice VARCHAR(150) DEFAULT NULL,
    score INTEGER DEFAULT 0,
    PRIMARY KEY (id),
    FOREIGN KEY (user_id) REFERENCES usr(id) ON DELETE CASCADE,
    FOREIGN KEY (questionnaire_id) REFERENCES questionnaire(id) ON DELETE CASCADE,
    CONSTRAINT only_one_form UNIQUE(user_id, questionnaire_id)
);
```

```sql
create table question (
    id INTEGER UNSIGNED AUTO_INCREMENT,
    question_text  VARCHAR(150) NOT NULL,
    questionnaire_id INTEGER UNSIGNED,
    PRIMARY KEY(id),
    FOREIGN KEY (questionnaire_id) REFERENCES questionnaire(id) ON DELETE CASCADE
);

create table answer (
    id INTEGER UNSIGNED AUTO_INCREMENT,
    question_id INTEGER UNSIGNED,
    form_id INTEGER UNSIGNED,
    response VARCHAR(150) DEFAULT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY (question_id) REFERENCES question(id) ON DELETE CASCADE,
    FOREIGN KEY (form_id) REFERENCES filled_form(id) ON DELETE CASCADE

);
```

Logical Model

offensive_word(ID, word)

admn(ID, username, email, password)

log(ID, user_id, ts)

usr(ID, username, email, password, banned, daily_points)

filled_form(ID, user_id, questionnaire_id, date_form, age, sex, expertice, score)

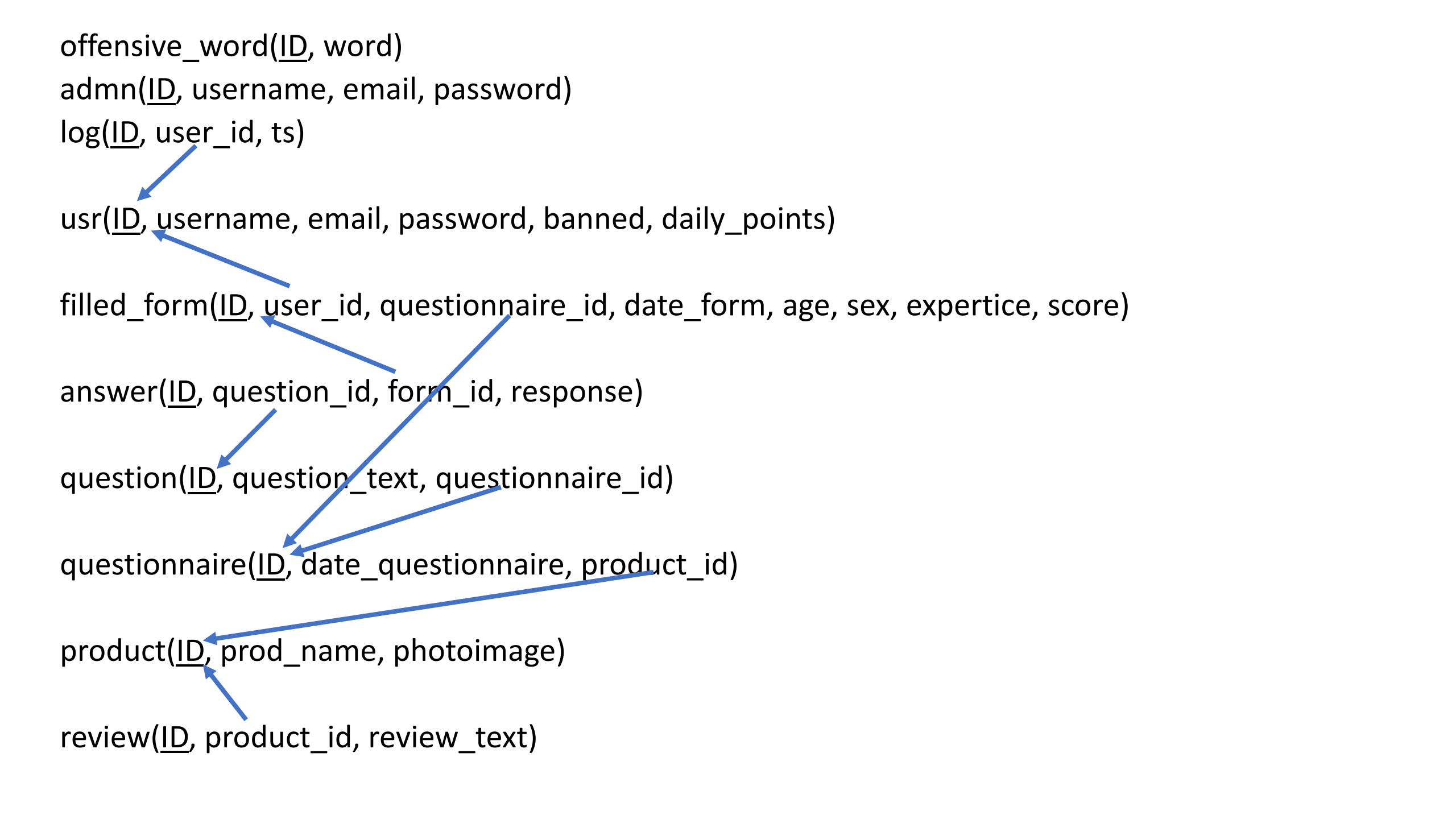
answer(ID, question_id, form_id, response)

question(ID, question_text, questionnaire_id)

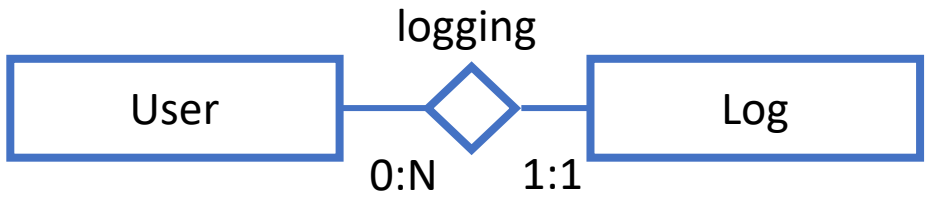
questionnaire(ID, date_questionnaire, product_id)

product(ID, prod_name, photoimage)

review(ID, product_id, review_text)

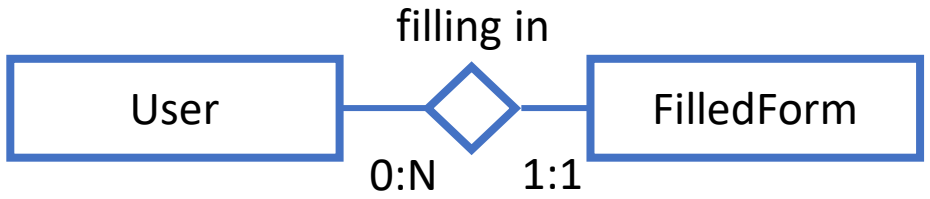


Relationship “logging”



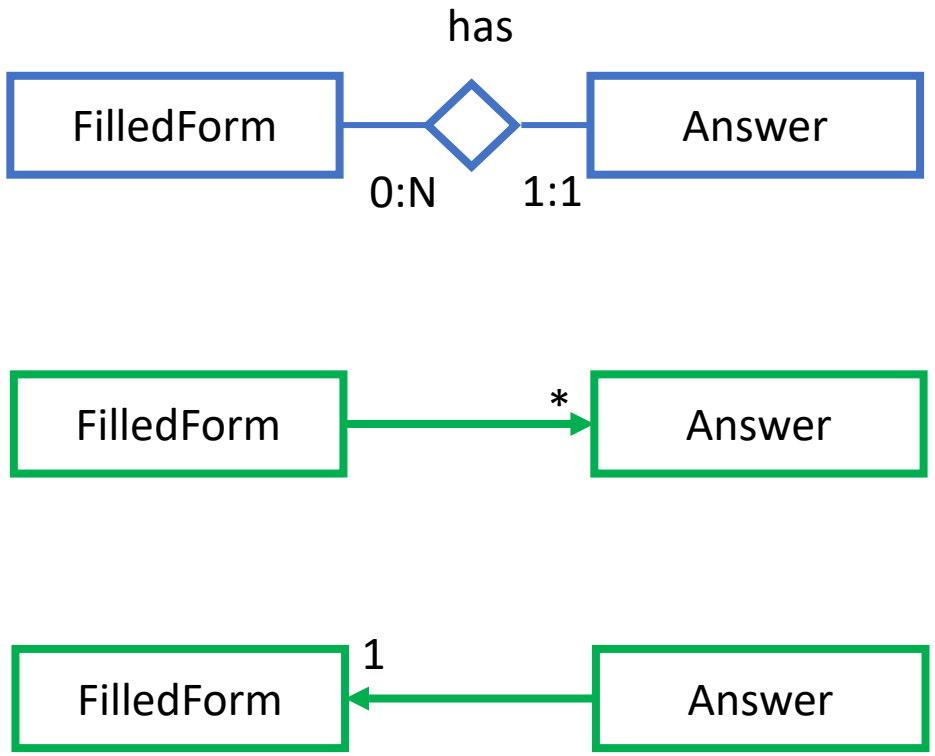
- User → Log: @OneToMany not required, mapped for simplicity
- Log → User: @ManyToOne necessary for retrieving the users who logged in a certain date
  - Owner: entity Log

Relationship “filling in”



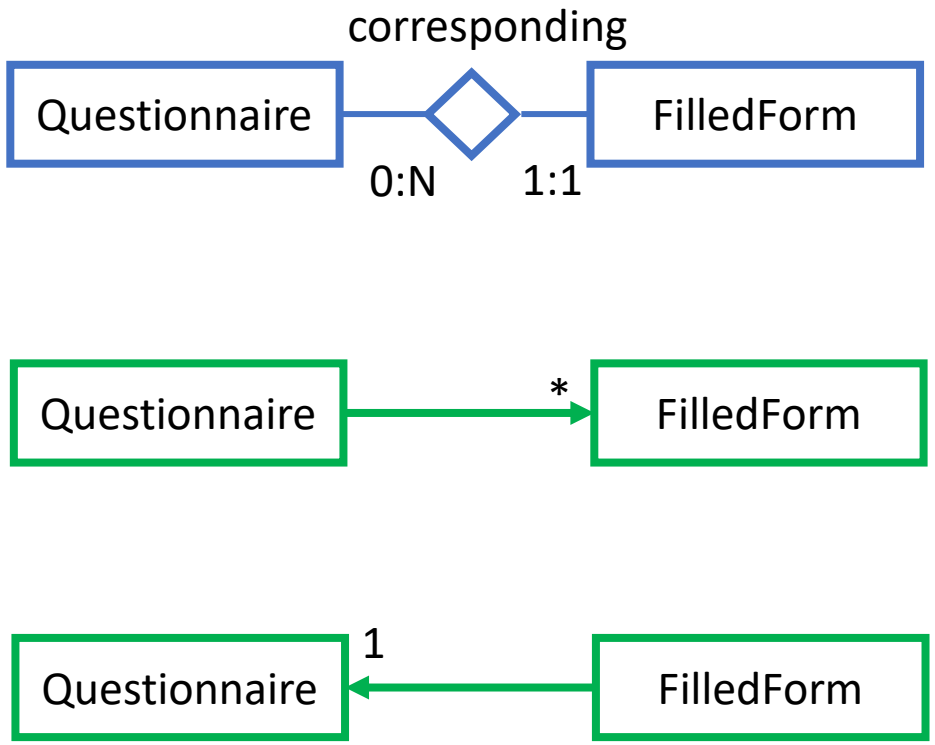
- User → FilledForm: @OneToMany not required, mapped for simplicity
- FilledForm → User: @ManyToOne necessary for retrieving the users who filled in the Form in a certain date
  - Owner: entity FilledForm

Relationship “has”



- FilledForm → Answer: @OneToMany, fetchType.EAGER for retrieving all the answers of a given FilledForm
  - CascadeType.PERSIST, CascadeType.REMOVE
- Answer → FilledForm: @ManyToOne not necessary, mapped for simplicity
  - Owner: entity Answer

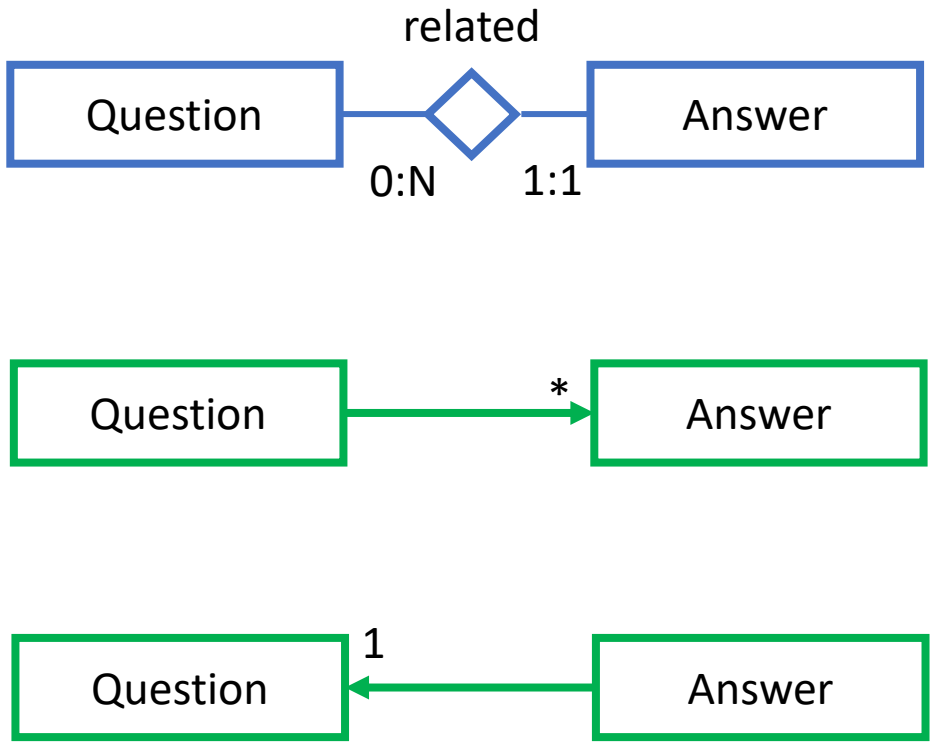
Relationship “corresponding”



- Questionnaire → FilledForm: @OneToMany for removing all the forms related to a specific deleted questionnaire with CascadeType.REMOVE

- FilledForm → Questionnaire: @ManyToOne not necessary, mapped for simplicity
  - Owner: entity FilledForm

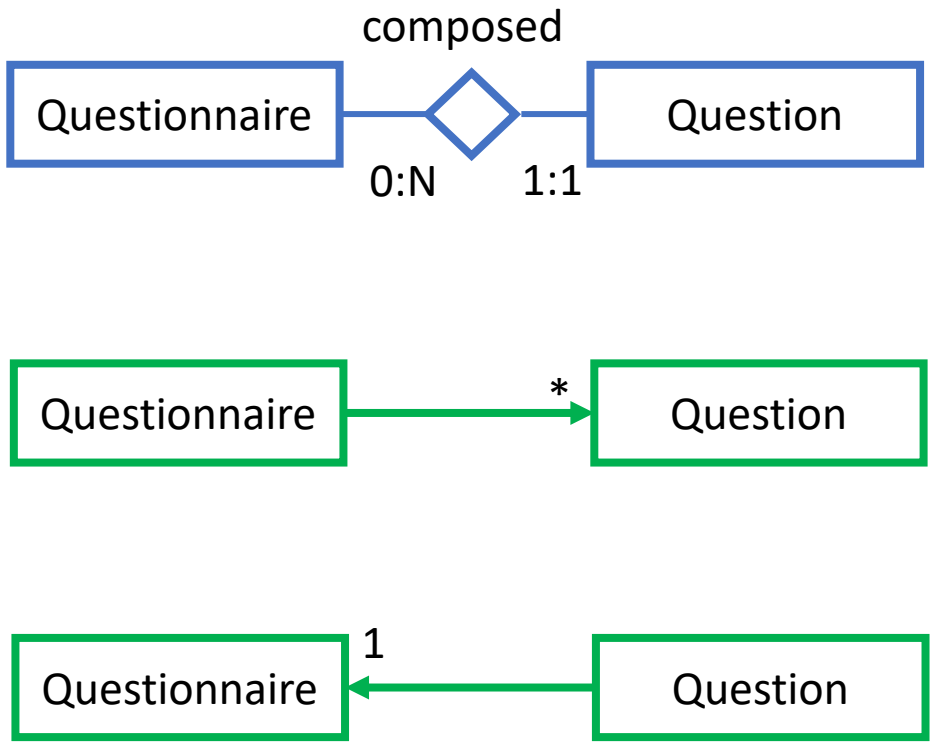
Relationship “related”



- Question → Answer: @OneToMany not necessary, mapped for simplicity. Could be used in future extentions for printing all the answer of a specific question.

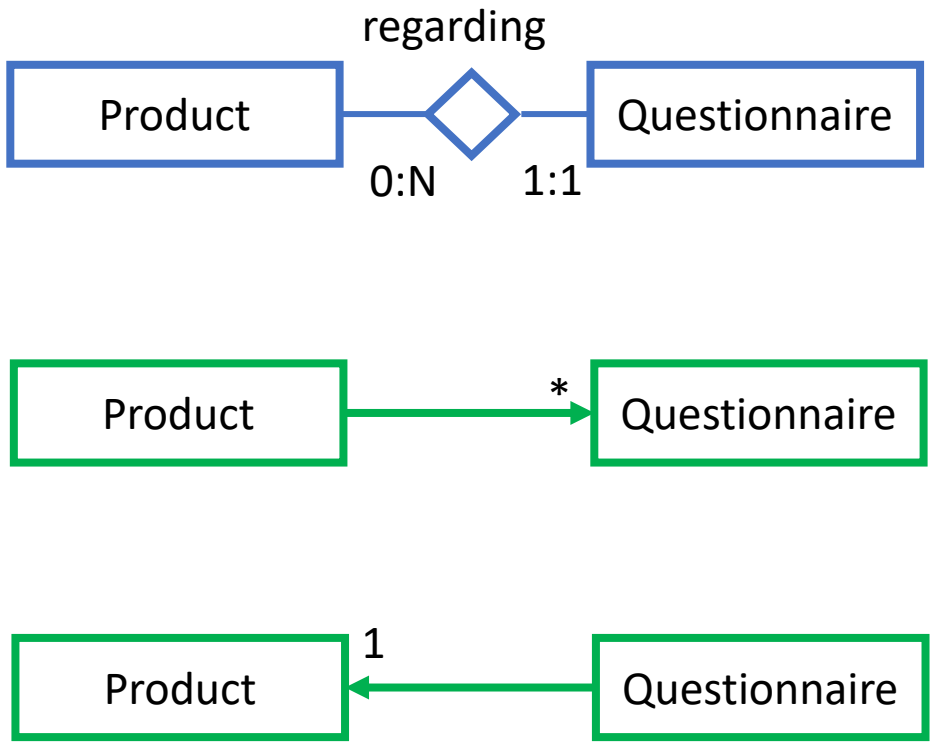
- Answer → Question : @ManyToOne not necessary, mapped for simplicity. Could be used in future extentions for printing the question of a given answer
  - Owner: entity Answer

Relationship “composed”



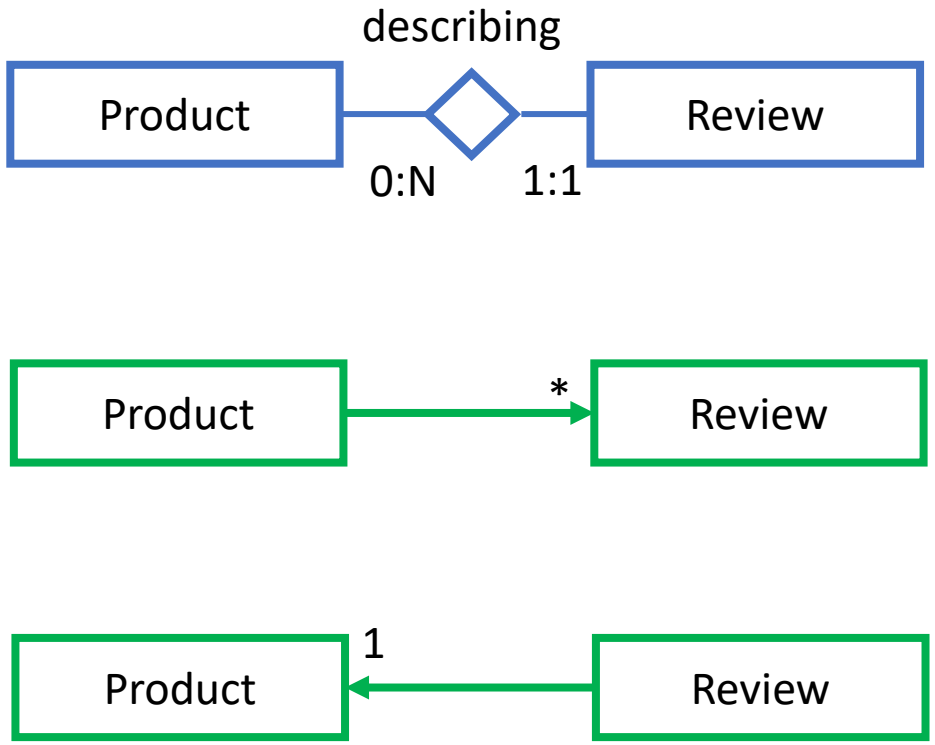
- Questionnaire → Question: @OneToMany, fetchType.EAGER for retrieving all the questions of a specific questionnaire
  - CascadeType.PERSIST, CascadeType.REMOVE
- Question → Questionnaire : @ManyToOne not necessary, mapped for simplicity
  - Owner: entity Question

Relationship “regarding”



- Product → Questionnaire : @OneToMany not necessary, mapped for simplicity
- Questionnaire → Product : @ManyToOne for retrieving the product of the questionnaire of the day
  - Owner: entity Questionnaire

Relationship “describing”



- Product → Review : @OneToMan, fetchType.EAGER for retrieving all the reviews of the product of the day
- Review → Product : @ManyToOne not necessary, mapped for simplicity
  - Owner: entity Review

Entities

- In the following slides are reported the Entities that have been used with the motivations of the NamedQueries and their main methods.
- All the Entities have been mapped exactly as just described in the preceding slides.

Admin

```
@Entity
@Table(name = "admn", schema = "db_easyr")
@NamedQuery(name = "Admin.checkCredentials",
query = "SELECT a FROM Admin a  WHERE a.username = ?1 and a.password = ?2")

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

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

private String username;
private String email;
private String password;
```

Motivations

- **checkCredentials** is used to check if exists an admin with certain username and password.

User

```
@Entity
@Table(name = "usr", schema = "db_easyr")
@NamedQueries({
@NamedQuery(name = "User.checkCredentials", query
= "SELECT u FROM User u  WHERE u.username = ?1 and
u.password = ?2"),
@NamedQuery(name = "User.getLeaderboard", query =
"SELECT u FROM User u WHERE u.points>0 ORDER BY
u.points DESC"),
@NamedQuery(name =
"User.hasDoneDailyQuestionnaire", query = "SELECT
u FROM User u, FilledForm f WHERE f.user = u AND
f.date = CURRENT_DATE AND u.id = ?1"),
@NamedQuery(name =
"User.hasDoneQuestionnaireByDate", query = "SELECT
u FROM User u, FilledForm f WHERE f.user = u AND
f.date = ?1"),
```

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

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

private String username;
private String email;
private String password;
private Boolean banned;

@Column(name = "daily_points")
private Integer points;

@OneToMany(mappedBy = "user")
private List<FilledForm> forms = new
ArrayList<FilledForm>();

@OneToMany(mappedBy = "user")
private List<Log> logs;
```

Motivations

- **checkCredentials** is used to check if exists a user with certain username and password.
- **getLeaderboard** is used to build the learderboard retrieving all the users orderd by descending points that have done the questionnaire of the day.
- **hasDoneDailyQuestionnaire**: is used for checking if a specific user has done the questionnaire of the day.
- **hasDoneQuestionnaireByDate**: is used for retrieving all the users that have done the questionnaire in a specific date.

Log

```
@Entity
@Table(name = "log", schema = "db_easyr")
@NamedQuery(name = "Log.hasOpenedQuestionnaireByDate", query = "SELECT DISTINCT l.user FROM Log l WHERE
l.date = ?1 AND l.user.banned=false")
public class Log implements Serializable {

private static final long serialVersionUID = 1L;

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

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

@Temporal(TemporalType.DATE)
@Column(name = "date_log")
private Date date;

@Temporal(TemporalType.TIMESTAMP)
@Column(name = "ts")
private Date timestamp;
```

Motivations

- **hasOpenedQuestionnaireByDate** is used for retrieving all the users (that have not been banned) that have at least opened the questionnaire in a specific date, so that then we can infer who canceled the questionnaire in the Admin inspection page.

Product

```
@Entity
@Table(name = "product", schema = "db_easyr")
@NamedQuery(name = "Product.getProdOfToday", query = "SELECT p FROM Questionnaire q JOIN q.product p WHERE
q.date=CURRENT_DATE")
public class Product implements Serializable {

private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer id;

@OneToMany(mappedBy = "product")
private List<Questionnaire> questionnaires = new ArrayList<Questionnaire>();

@OneToMany(mappedBy = "product", fetch = FetchType.EAGER)
private List<Review> reviews;

@Basic(fetch = FetchType.EAGER)
@Lob
private byte[] photoimage;

@Column(name = "prod_name")
private String name;
```

Motivations

- **getProdOfToday** is used for retrieving the product of the day. Notice that, as already stated, the product does not have the attribute date, because we wanted to permit more questionnaires related to the same product in different dates. For this reason, we have to join the product table with the questionnaire table.
- Notice also that the fecthType with the Reviews is EAGER because we want to display all the Reviews, if any, in the homepage.

Review

```java
@Entity
@Table(name = "review", schema = "db_easyr")
public class Review implements Serializable {

private static final long serialVersionUID = 1L;

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

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

@Column (name="review_text")
private String text;
```

Questionnaire

```
@Entity
@Table(name = "questionnaire", schema =
"db_easyr")

@NamedQueries({
@NamedQuery(name =
"Questionnaire.getQuestOfToday", query = "SELECT
q FROM Questionnaire q WHERE
q.date=CURRENT_DATE"),
@NamedQuery(name =
"Questionnaire.getQuestByDate", query = "SELECT q
FROM Questionnaire q WHERE q.date= ?1") })

public class Questionnaire implements
Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy =
GenerationType.IDENTITY)
private Integer id;
```

```
@OneToMany(mappedBy = "questionnaire",
cascade= CascadeType.REMOVE)
private List<FilledForm> forms;

@Temporal(TemporalType.DATE)
@Column(name = "date_questionnaire")
private Date date;

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

@OneToMany(mappedBy = "questionnaire", fetch
= FetchType.EAGER, cascade = {
CascadeType.PERSIST, CascadeType.REMOVE })
private List<Question> questions = new
ArrayList<Question>();

public void addQuestion(Question
question) {
getQuestions().add(question);
question.setQuestionnaire(this);
}
```

Motivations

- **getQuestOfToday** is used for retrieving the questionnaire of the day.
- **getQuestByDate** is used for retrieving a specific questionnaire of a certain date, so that then the Admin can delete it.

- Notice also that the fecthType with the Questions is **EAGER** because we want to display all the Questions in the questionnairepage.
- We have used a **PERSIST** and **REMOVE** cascading with Questions so that when a Questionnaire is either created or removed the same happens for all the related Questions.
- For the same reason, we have also used **REMOVE** cascading with FilledForm.
- The method **addQuestion ()** is used for keeping updated both sides of the relationship between Questionnaire and Questions.

Question

```java
@Entity
@Table(name = "question", schema = "db_easyr")
public class Question implements Serializable {
private static final long serialVersionUID = 1L;

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

@Column(name = "question_text")
private String text;

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

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

FilledForm

```
@Entity
@Table(name = "filled_form", schema = "db_easyr")
@NamedQuery(name =
"FilledForm.retrieveAnswersByDate", query =
"SELECT f FROM FilledForm f  WHERE f.date = ?1")

public class FilledForm implements Serializable {

private static final long serialVersionUID = 1L;

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

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

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

```
@OneToMany(mappedBy = "form", cascade = {
CascadeType.PERSIST, CascadeType.REMOVE },
fetch = FetchType.EAGER)
private List<Answer> answers = new
ArrayList<Answer>();

@Temporal(TemporalType.DATE)
@Column(name = "date_form")
private Date date;

private Integer age;
private String sex;
private String expertice;
private Integer score;

public void addAnswer(Answer answer) {
getAnswers().add(answer);
answer.setForm(this);
}
```

Motivations

- **retrieveAnswersByDate** is used for retrieving all the filled forms of a specific questionnaire given its date.
- Notice also that the fecthType with the Answer is **EAGER** because we want to display to the Admin all the Answers in the inspection page.
- We have used a **PERSIST** and **REMOVE** cascading with Answer so that when a FilledForm is either created or removed the same happens for all the related Answers.
-
- The method **addAnswer()** is used for keeping updated both sides of the relationship between FilledForm and Answer.

Answer

```
@Entity
@Table(name = "answer", schema = "db_easyr")
 public class Answer implements Serializable {
private static final long serialVersionUID = 1L;

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

@ManyToOne
@JoinColumn(name = "question_id")
private Question question;

@ManyToOne
@JoinColumn(name="form_id")
private FilledForm form;

private String response;
```

Components (1/2)

- **Client (User)**
  - CheckLogin
  - GoToHomePage
  - GoToLeaderboardPage
  - GoToQuestionnairePage
  - GoToRegistrationPage
  - GoToStatisticsPage
  - GoToThanksPage
  - Logout
  - UserRegistration
  - index.html
  - Home.html
  - LeaderboardPage.html
  - QuestionnairePage.html
  - RegistrationPage.html
  - StatisticsPage.html
  - ThanksPage.html

- **Client (Admin)**
  - CheckLogin
  - CreateQuestionnaire
  - DeleteQuestionnaire
  - GetStatistics
  - GoToCreationPage
  - GoToDeletionPage
  - GoToHomePage
  - GoToInspectionPage
  - Logout
  - PrintQuestions
  - index.html
  - CreationPage.html
  - DeletionPage.html
  - Home.html
  - InspectionPage.html

Components: Business Tier (2/2)

- **@Stateless UserService**
  - checkCredentials(String usrn, String pwd)
  - checkIfBanned(Integer userId)
  - hasDoneDailyQuestionnaire(Integer userId)
  - hasDoneQuestionnaireByDate(Date date)
  - saveLog(User user)
  - registerUser(String usrn, String email, String pwd)
  - getLeaderboard(Integer userId)
  - hasOpenedQuestionnaireByDate(Date date)
- **@Stateless QuestionnaireService**
  - getQuestionnaireOfToday()
  - saveQuestionnaire(Date date, Product product, List<Question> questions)
  - deleteQuestionnaire(Date date)
- **@Stateless FilledFormService**
  - saveFilledForm(User user, Questionnaire questionnaire, List<Answer> answers, Integer age, String sex, String expertice)
  - retrieveByDate(Date date)
  - deleteFilledForm(Integer formId)
- **@Stateless AnswerService**
  - saveAnswers(List<String> answers, List<Question> questions)
- **@Stateless ProductService**
  - getProductOfToday()
  - createProduct(byte[] photoimage, String name)
- **@Stateless AdminService**
  - checkCredentials(String usrn, String pwd)
- **@Stateless QuestionService**
  - saveQuestions(List<String> questions)

Stateless EJBs because all the method calls are independent of the session state

Example of business method (1/3)

```
public FilledForm saveFilledForm(User user, Questionnaire questionnaire, List<Answer> answers,
Integer age, String sex, String expertice) throws SQLException {

FilledForm form = null;
form = new FilledForm(user, questionnaire, age, sex, expertice);

for (Answer a : answers) {

  form.addAnswer(a);

}

questionnaire.addFilledForm(form);
user.getForms().add(form);
em.persist(form);

return form; }
```

Example of business method (2/3)

```
public void deleteQuestionnaire(Date date) throws Exception {

Questionnaire questionnaire = null;

try {
  questionnaire = em.createNamedQuery("Questionnaire.getQuestByDate",
  Questionnaire.class).setParameter(1, date).getSingleResult();
} catch (Exception e) {

  throw new Exception("Error searching the questionnaire");
}

em.remove(questionnaire);

}
```

Example of business method (3/3)

```
public Product getProductOfToday() throws Exception {
List<Product> prod = null;
try {
prod = em.createNamedQuery("Product.getProdOfToday", Product.class).getResultList();

} catch (Exception e) {
throw new Exception("Could not find the product of the day");
}

if (prod.isEmpty()) {
return null;

} else {
return prod.get(0);
}
}
```