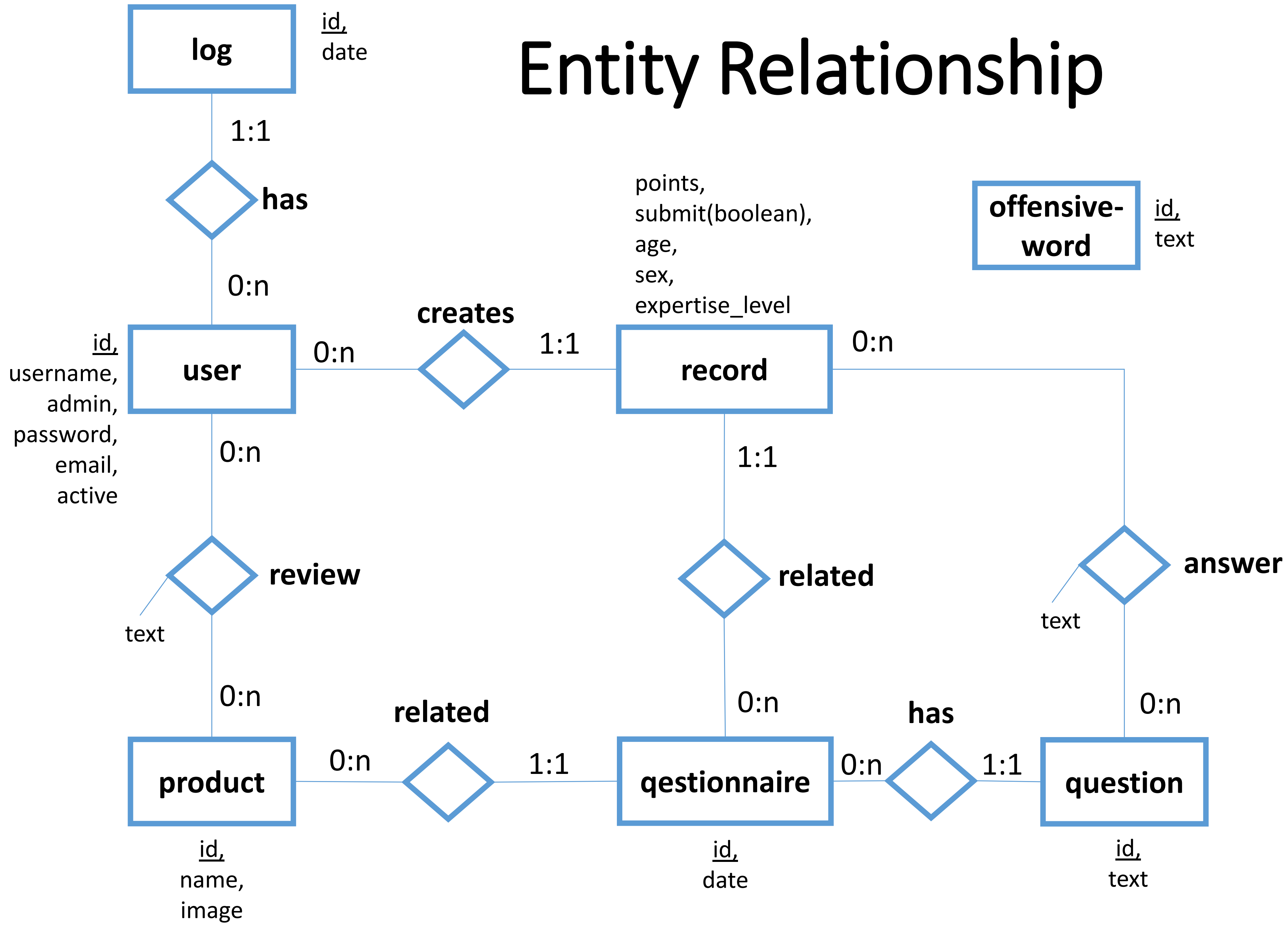
Gamified Marketing Application

Shalby Hazem Hesham Yousef

POLITECNICO
MILANO 1863

Entity Relationship



Relational model

log(id, date, id_user)

user(id, username, password, email, active, admin)

review(id,id_user, id_product, text)

product(id, name, image)

record(id, points, submit, age, sex, expertise_level, id_user, id_questionnaire)

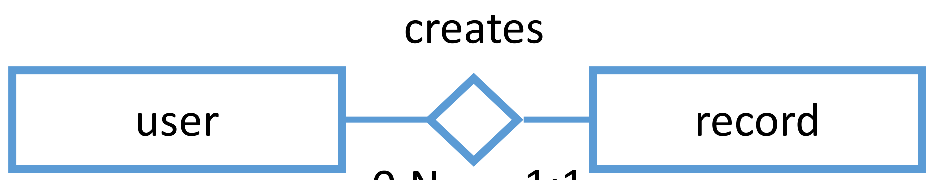
answer(id_record, id_question, text)

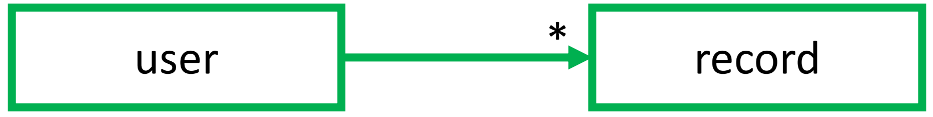
question(id, text, id_questionnaire)

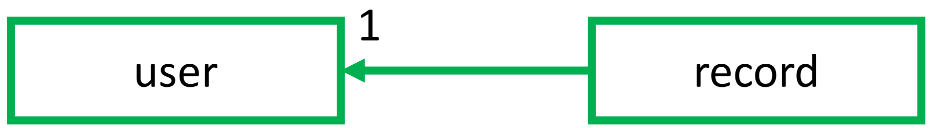
questionnaire(id, date, id_product)

offensive-word(id, text)

Relationship “Creates”

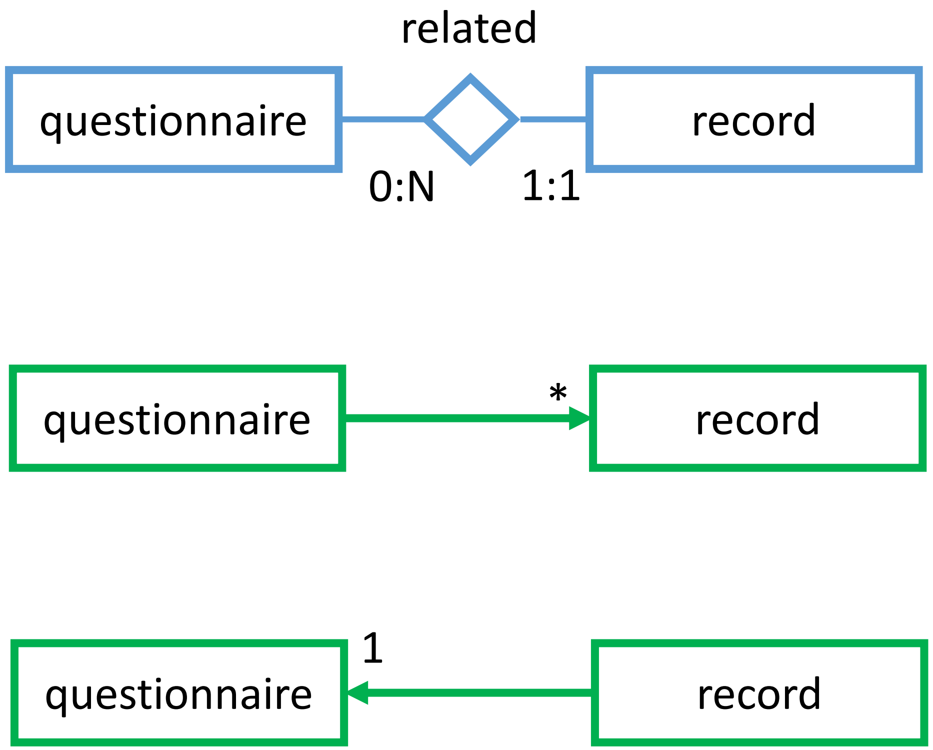






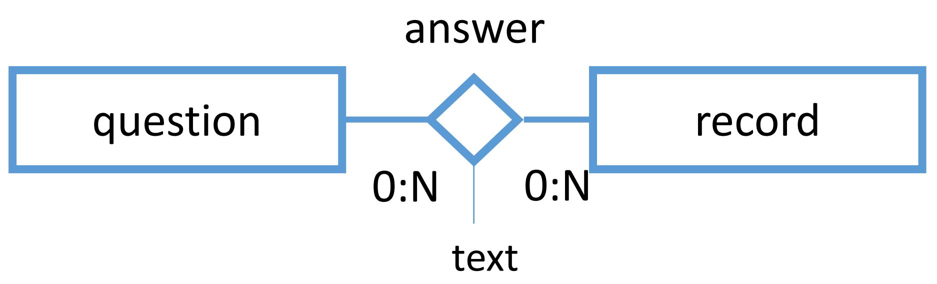
- user → record @OneToMany
  - Cascade : REMOVE, REFRESH, MERGE
- record → user @ManyToOne
  - Not useful for the system but implemented for simplicity and for system evolution

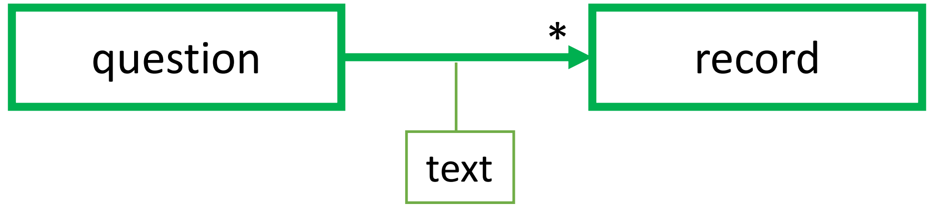
Relationship “Related”

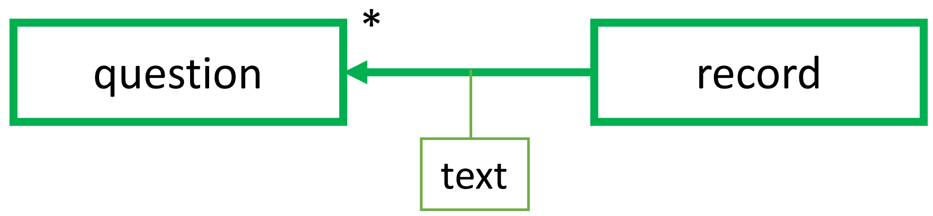


- questionnaire→ record @OneToMany
  - Cascade : REMOVE, REFRESH
- record → questionnaire @ManyToOne
  - Not useful for the system but implemented for simplicity and for system evolution

Relationship “answer”

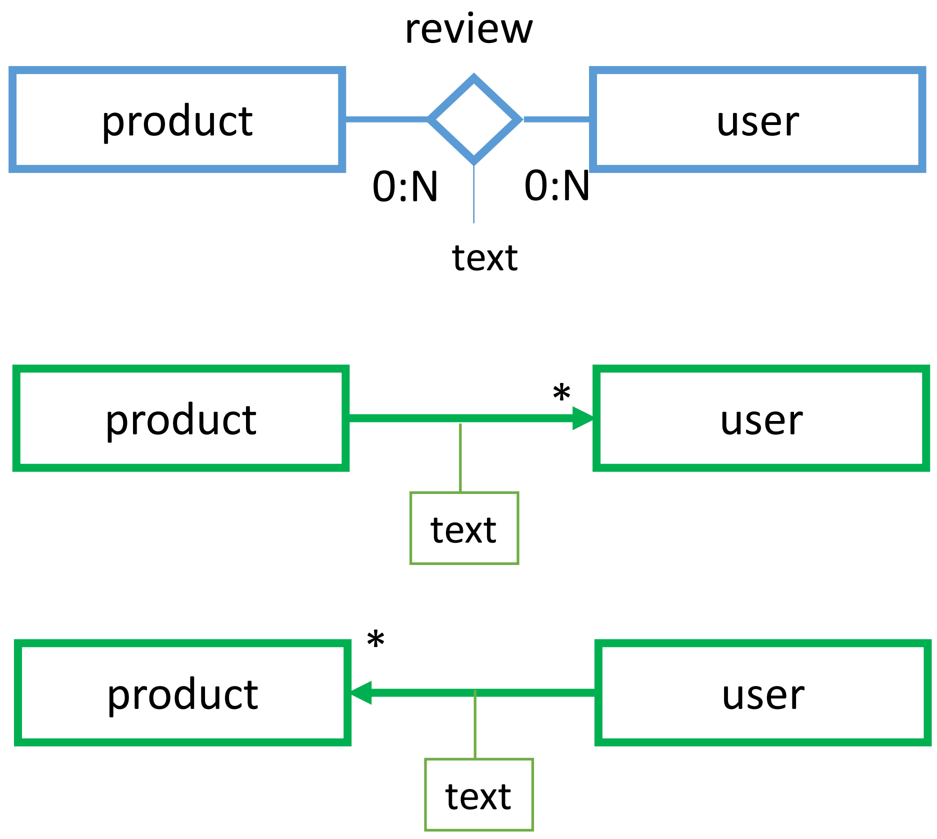






- question→ record @ElementCollection (@ManyToMany)
  - Not useful for the system but implemented for simplicity and for system evolution
- record → question @ElementCollection (@ManyToMany)
  - Necessary for the admin part to show the answers for users record
  - `fetch = FetchType.`***EAGER***

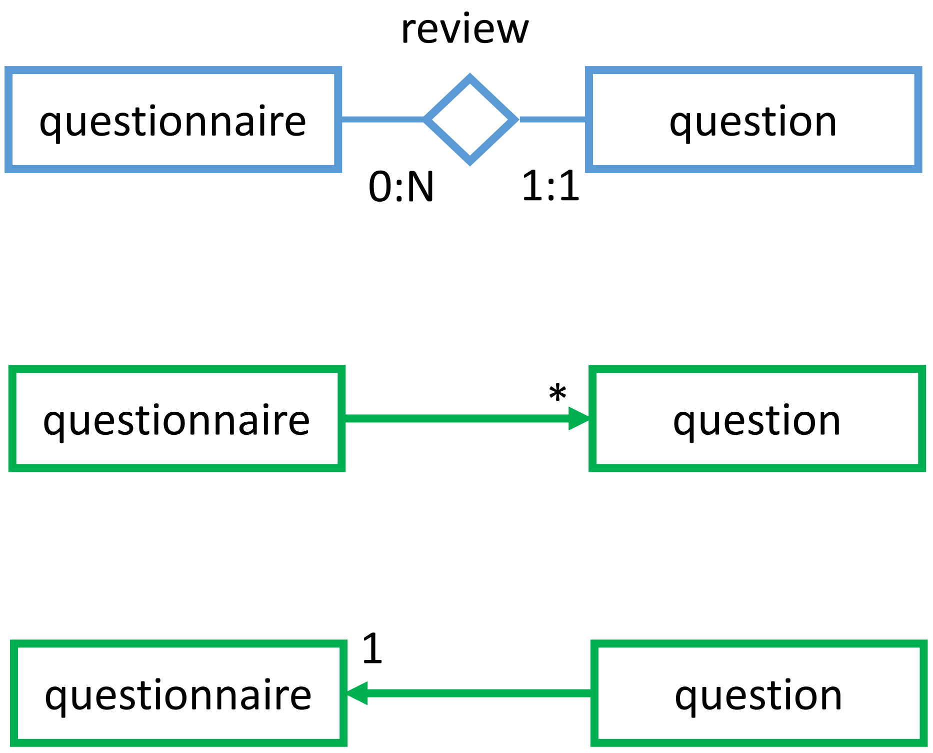
Relationship “review”



- user→ product @ElementCollection (@ManyToMany)
  - Not useful for the system but implemented for simplicity and for system evolution

- product → user @ElementCollection (@ManyToMany)
  - Necessary for the HOME page to show all the reviews associated to a product
  - `fetch = FetchType.`***EAGER***

Relationship “has”



- questionnaire→ question @OneToMany
  - Cascade : REMOVE, PERSIST
- question→ questionnaire @ManyToOne
  - Not useful for the system but implemented for simplicity and for system evolution

Entity User

```
@Entity
@Table(name = "user", schema = "gamified_marketing_db")
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 boolean active;
private boolean admin;
@OneToMany(mappedBy = "user")
private List<Log> log_list;
@OneToMany(mappedBy = "user", cascade = CascadeType.REMOVE, fetch = FetchType.LAZY)
private List<Record> records;
@ElementCollection
@CollectionTable(name = "review", schema = "gamified_marketing_db", joinColumns = @JoinColumn(name = "id_user"))
@MapKeyJoinColumn(name = "id_product")
@Column(name = "text")
private Map<Product,String> reviews;
```

Named Queries (USER)

```
@NamedQueries({

@NamedQuery(name = "User.checkCredentials", query = "SELECT u
FROM User u  WHERE u.username = ?1 and u.password = ?2"),

@NamedQuery(name = "User.count", query = "SELECT count(u)
FROM User u  WHERE u.username = ?1"),

@NamedQuery(name = "User.getLeaderBoard", query = "SELECT NEW
it.polimi.db2.utils.LeaderBoard(u.username, r.points) FROM
User u, Record r  WHERE u = r.user AND r.questionnaire =
(SELECT q FROM Questionnaire q where q.date = current_date )
GROUP BY u.id ORDER BY r.points DESC")

})
```

Entity Record

```
@Entity
@Table(name = "record", schema = "gamified_marketing_db")
public class Record {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
private int points;
private int age;
private boolean submit;
private UserSex sex;
private UserExpertiseLevel expertise_level;
@ManyToOne
@JoinColumn(name = "id_user")
private User user;
@ManyToOne
@JoinColumn(name = "id_questionnaire")
private Questionnaire questionnaire;
@ElementCollection
@CollectionTable(name = "answer", schema = "gamified_marketing_db", joinColumns = @JoinColumn(name = "id_record"))
@MapKeyJoinColumn(name = "id_question")
@Column(name = "text")
private Map<Question, String> answers;
```

Named Queries (RECORD)

```
@NamedQuery(name = "Record.getRecordPerUserAndQuestionnaire",
query = "SELECT r FROM Record r  WHERE r.user = ?1 and
r.questionnaire = ?2")
```

Entity Questionnaire

```
@Entity
@Table(name = "questionnaire", schema = "gamified_marketing_db")
public class Questionnaire {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
@Temporal(TemporalType.DATE)
private Date date;
@ManyToOne
@JoinColumn(name = "id_product")
private Product product;

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

@OneToMany(mappedBy = "questionnaire", cascade = { CascadeType.REMOVE })
private List<Record> records;
```

Named Queries (Questionnaire)

```
@NamedQueries({

@NamedQuery(name = "Questionnaire.getQuestionnaireByDate",
query = "SELECT q FROM Questionnaire q where q.date = ?1"),

@NamedQuery(name = "Questionnaire.eraseQuestionnaireByDate",
query = "Delete FROM Questionnaire q where q.date = ?1")

})
```

Entity Question

```java
@Entity
@Table(name = "question", schema = "gamified_marketing_db")
public class Question implements Serializable {

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

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

@ElementCollection
@CollectionTable(name = "answer", schema = "gamified_marketing_db", joinColumns =
@JoinColumn(name = "id_question"))
@MapKeyJoinColumn(name = "id_record")
@Column(name = "text")
private Map<Record, String> answers;
```

Entity Product

```
@Entity
@Table(name = "product", schema = "gamified_marketing_db")
public class Product implements Serializable {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
private String name;
@Basic(fetch = FetchType.EAGER)
@Lob
private byte[] image;

@ElementCollection
@CollectionTable(name = "review", schema = "gamified_marketing_db", joinColumns =
@JoinColumn(name = "id_product"))
@MapKeyJoinColumn(name = "id_user")
@Column(name = "text")
private Map<User, String> reviews;

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

Named Queries (Product)

```
@NamedQueries({
@NamedQuery(name = "Product.getProductByDate", query =
"SELECT p FROM Product p, Questionnaire q where q.date = ?1
and p= q.product"),
@NamedQuery(name = "Product.getAllProducts", query = "SELECT
p FROM Product p")
})
```

TRIGGER (update user points)

```
CREATE TRIGGER `UPDATE_POINTS_1`
AFTER INSERT ON `answer`
FOR EACH ROW
BEGIN
UPDATE RECORD SET points = points + 1 WHERE new.id_record = id;
END

CREATE TRIGGER `UPDATE_POINTS_2`
BEFORE INSERT ON `record`
FOR EACH ROW
BEGIN
SET new.points = new.points + 2* (new.age>0) + 2* (NOT
isnull(new.expertise_level)) + 2* (NOT isnull(new.sex));
END

//two triggers are used: the first one assign the point to each
//question answers, while the second one is used to compute the
//points for the optional field (SEX, EXPERTISE LEVEL, AGE)
```

TRIGGER(DETECT OFFENSIVE WORD)

```
CREATE TRIGGER `DETECT_OFFENSIVE_WORD`
BEFORE INSERT ON `answer`
FOR EACH ROW
BEGIN
IF((select count(*) FROM offensive_word o where new.text like
concat('%',o.text,'%'))>0)
THEN
SIGNAL sqlstate '45001' set message_text = "No way ! You cannot use this
language";
END IF;
END

//If any user answers contains one of the forbidden words, an
//exception is launched, and  the transaction is aborted. The EJB
//application capture the exception and manage it by banning the
//user
```

Business Components

```
@Stateless UserService

public User checkCredentials(String usrn, String pwd) throws CredentialsException,
NonUniqueResultException

public User registerUser(String username, String password, String email, boolean active,
boolean admin) throws CredentialsException

public void banUser(User user)
```

```
@StateLess LeaderBoardService

public List<LeaderBoard> getLeaderBoardOfTheDay()
```

```
@StateLess ProductService

public Product getProductOfTheDay()

public Product getProductbyDate(Date date)

public List<Product> getTheProdcutList()

public Product getProductById(int id)
```

Business Components

```
@StateLess QuestionnaireService

public Questionnaire getQuestionnaireOfTheDay()

public Questionnaire getQuestionnaireByDate(Date date)

public void addQuestionnaire(Date date, Product product, String[] questions)

public void eraseQuestionnaire(Date date) throws DeletionAfterCurrentDateException



@StateLess RecordService

public boolean isThereAnyRecord(User user, Questionnaire questionnaire)

public void addNewRecord(String[] answers, int age, UserSex sex, UserExpertiseLevel
user_expertise_level, User user, Questionnaire questionnaire, List<Question> questions)
throws NotValidEntryException, NonActiveUserException, OffensiveLanguageException

public void cancelQuestionnaire(User user, Questionnaire questionnaire)
```