Gamified Marketing Application

Data Bases 2 Project

Group number: 117
Stefano Longoni - 10537385
Samuele Negrini - 10539341
Giorgio Piazza - 10529035

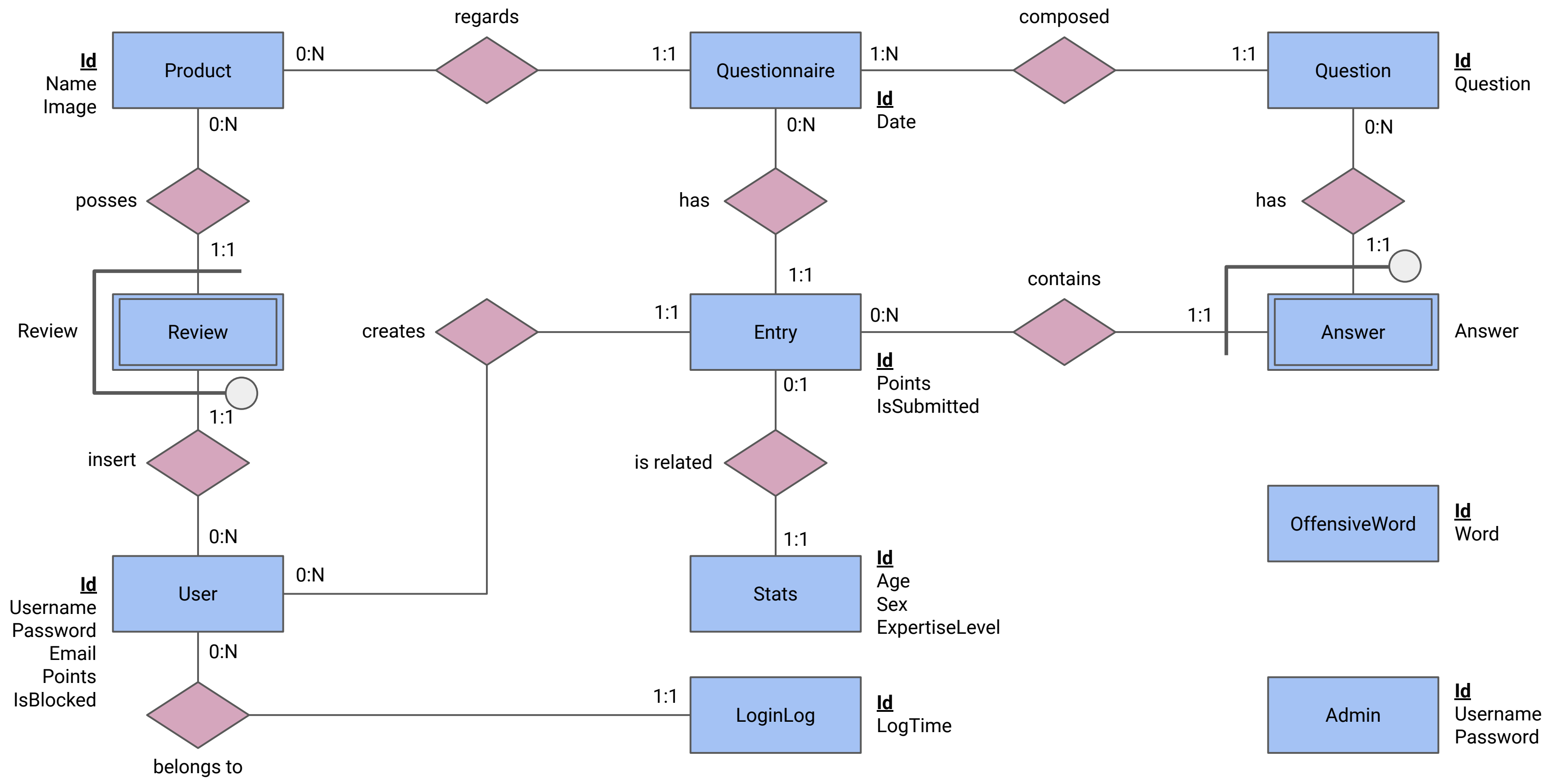
POLITECNICO
MILANO 1863

Introduction

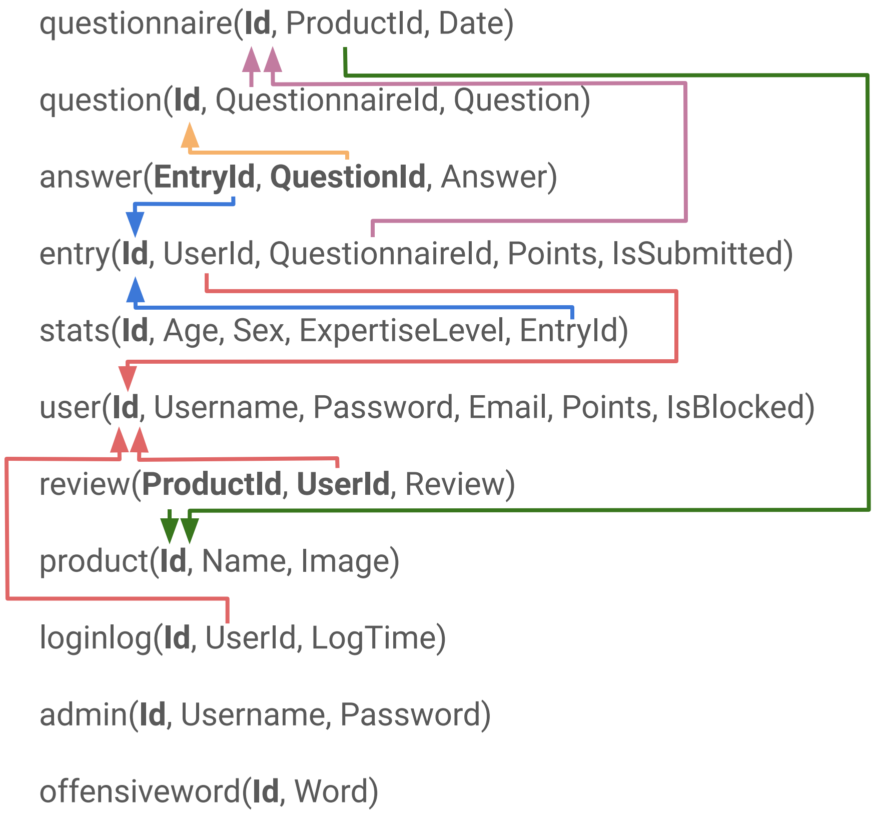
These slides aim to explain the main design choices that have been adopted during the development of the project.

To avoid ending up with a too long presentation, we decided to put our focus on the main relationships of the E/R diagram and on the triggers.

ER Diagram


regards
composed
Id
Name
Image
Product
0:N
1:1
Questionnaire
1:N
1:1
Question
Id
Question
Id
Date
0:N
0:N
0:N
posses
has
has
1:1
1:1
1:1
contains
Review
Review
creates
1:1
Entry
0:N
1:1
Answer
Answer
Id
Points
IsSubmitted
0:1
1:1
insert
is related
0:N
1:1
OffensiveWord
Id
Word
Id
Username
Password
Email
Points
IsBlocked
User
0:N
Stats
Id
Age
Sex
ExpertiseLevel
0:N
1:1
LoginLog
Id
LogTime
Admin
Id
Username
Password
belongs to

Logical model in compact form



questionnaire(**Id**, ProductId, Date)

question(**Id**, QuestionnaireId, Question)

answer(**EntryId**, **QuestionId**, Answer)

entry(**Id**, UserId, QuestionnaireId, Points, IsSubmitted)

stats(**Id**, Age, Sex, ExpertiseLevel, EntryId)

user(**Id**, Username, Password, Email, Points, IsBlocked)

review(**ProductId**, **UserId**, Review)

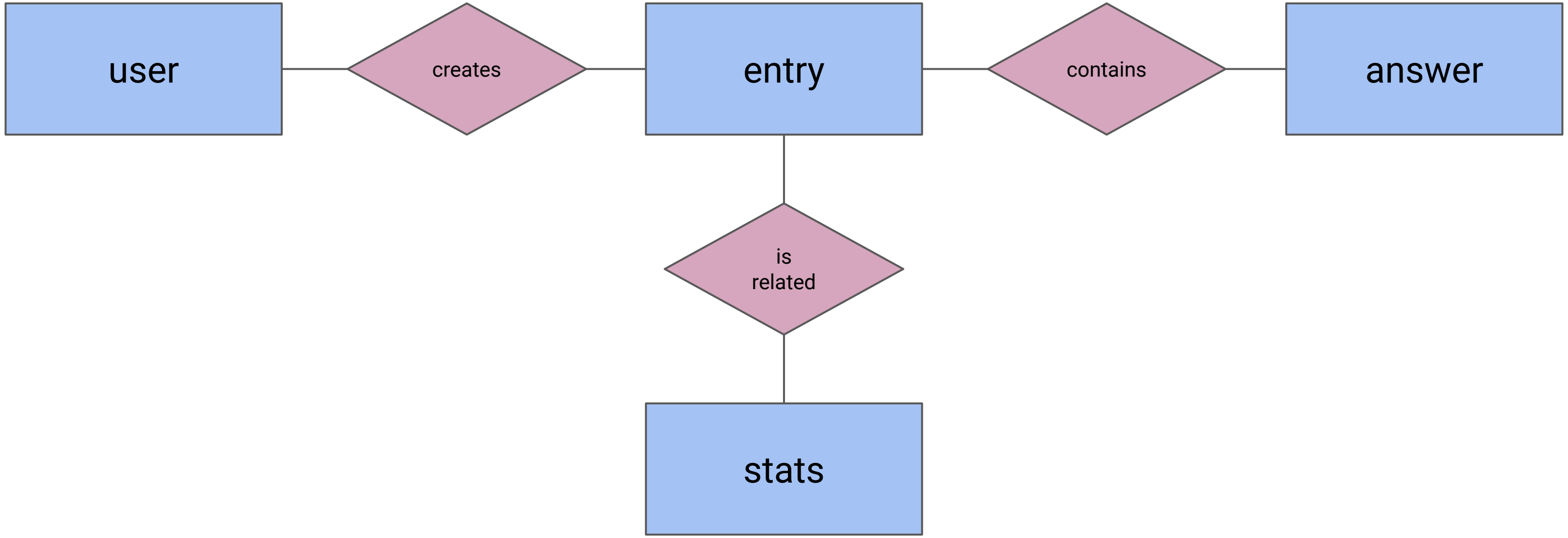
product(**Id**, Name, Image)

loginlog(**Id**, UserId, LogTime)

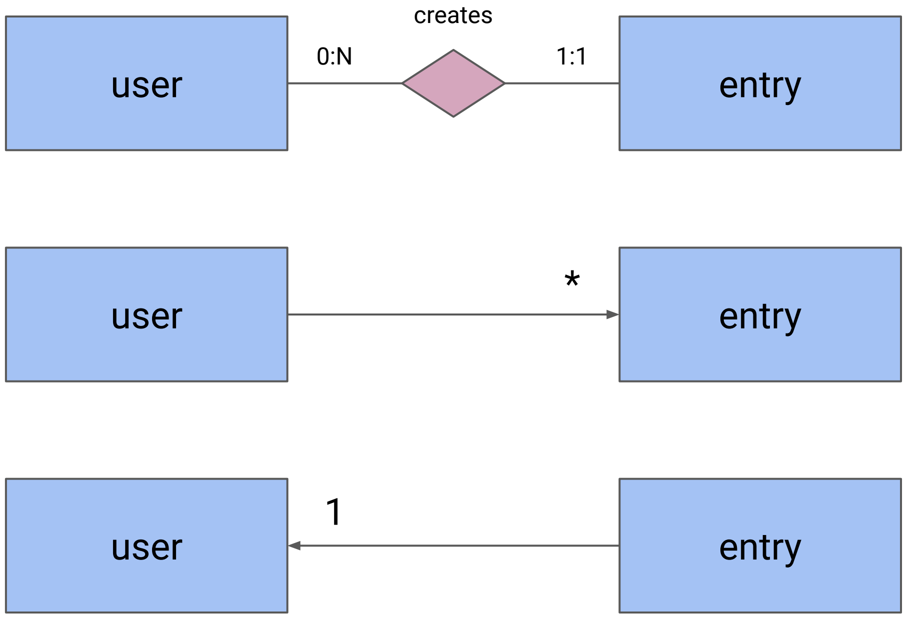
admin(**Id**, Username, Password)

offensiveword(**Id**, Word)

Entry relationships



Relationship User **“creates”** Entry



user ➔ entry **`@OneToMany`** is necessary to get the entries of the logged user.

entry ➔ user **`@ManyToOne`** is not requested by the specifications but can be useful for future purposes.

User Entity

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "Id", nullable = false)
private int id;

@Column(name = "Username", nullable = false, length = 45)
private String username;

@Column(name = "Password", nullable = false, length = 45)
private String password;

@Column(name = "Email", nullable = false, length = 90)
private String email;

@Column(name = "Points", nullable = false)
private int points = 0;

@Column(name = "IsBlocked", nullable = false)
private Byte isBlocked = 0;

@OneToMany(mappedBy = "user", cascade = {CascadeType.PERSIST}, orphanRemoval = true)
private List<ReviewEntity> reviews = new ArrayList<>();

@OneToMany(mappedBy = "user", cascade = {CascadeType.PERSIST})
private List<LoginlogEntity> loginlogs = new ArrayList<>();

@OneToMany(mappedBy = "user", cascade = {CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REMOVE, CascadeType.REFRESH}, orphanRemoval
= true)
private List<EntryEntity> entries = new ArrayList<>();
```

User Entity - Motivations

Bidirectional one-to-many association User to Entry.

- Entry is defined as the owner entity.
- PERSIST, REMOVE, REFRESH, MERGE are cascaded. Persisting or reloading from a database an already existing user also persists/loads any new entry associated to them.
  Removing a user also removes any entries associated with them.
- orphanRemoval = true causes entries without user to be removed.

Bidirectional one-to-many association User to Loginlogs.

- Loginlogs is defined as the owner entity.
- Only PERSIST is cascaded.
- Other CascadeType are not cascaded because login logs shall not be modified in any way.

Bidirectional one-to-many association User to Review.

- Review is defined as the owner entity.
- PERSIST is cascaded. Persisting or reloading from a database an already existing user also persists/loads any new review associated to them.
- We chose not to cascade REMOVE to keep reviews of also deleted users.
- Orphanremoval = true allows to remove reviews when they are removed from the parent entity.

User Entity - Named Queries

```
// Returns the user with the provided login credentials in order to check the user login
@NamedQuery(name = "UserEntity.checkCredentials", query = "SELECT u FROM UserEntity u WHERE u.username = :username AND
u.password = :password")

// Returns the User from the username
@NamedQuery(name = "UserEntity.findByUsername", query = "SELECT u FROM UserEntity u WHERE u.username = :username")

// Returns the User from the email
@NamedQuery(name = "UserEntity.findByEmail", query = "SELECT u FROM UserEntity u WHERE u.email = :email")

// Returns the leaderboard rows of a specific date ordered by the points. A Data Transfer Object (DTO) is used to
simplify the access to data during templating.
@NamedQuery(name = "UserEntity.getLeaderboardByDate", query = "SELECT NEW
it.polimi.db2.gma.GMAEJB.utils.LeaderboardRow(u.username, e.points) FROM UserEntity u INNER JOIN u.entries e INNER JOIN
e.questionnaire q WHERE q.date = :date AND e.isSubmitted = 1 ORDER BY e.points DESC")

// Retrieves the users information of the users who completed the provided questionnaire. The users are filtered based on
the submit status of the entry. A DTO is used to simplify the access to data during templating.
@NamedQuery(name = "UserEntity.getEntriesUserInfo", query = "SELECT NEW it.polimi.db2.gma.GMAEJB.utils.UserInfo(u.id,
u.username) FROM UserEntity u INNER JOIN u.entries e WHERE e.questionnaire.id = :id AND e.isSubmitted = :submitted ORDER
BY u.id")
```

Entry Entity

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "Id", nullable = false)
private int id;

@Column(name = "Points", nullable = false)
private int points = 0;

@Column(name = "IsSubmitted", nullable = false)
private Byte isSubmitted = 0;

@ManyToOne
@JoinColumn(name = "UserId", nullable = false)
private UserEntity user;

@ManyToOne
@JoinColumn(name = "QuestionnaireId")
private QuestionnaireEntity questionnaire;

@OneToOne(mappedBy = "entry", cascade = {CascadeType.PERSIST, CascadeType.REMOVE,
        CascadeType.REFRESH, CascadeType.MERGE}, orphanRemoval = true)
private StatsEntity stats;

@OneToMany(mappedBy = "entry", fetch = FetchType.EAGER, cascade = {CascadeType.PERSIST, CascadeType.REMOVE,
        CascadeType.REFRESH, CascadeType.MERGE}, orphanRemoval = true)
private List<AnswerEntity> answers = new ArrayList<>();
```

Entry Entity - Motivations

Bi-directional many-to-one association to User (Entry is the owner entity).

Bi-directional many-to-one association to Questionnaire (Entry is the owner entity).

Bi-directional one-to-one association Entry to Stats.

- Stats is defined as the owner entity.
- PERSIST, REFRESH, MERGE, REMOVE are cascaded to the dependent entity Stats. In particular, REMOVE allows to delete all the stats associated with the entry when the latter is deleted.
- orphanRemoval = true causes stats without entry to be removed (private parent-child relationship: there are no other relations referring to stats).

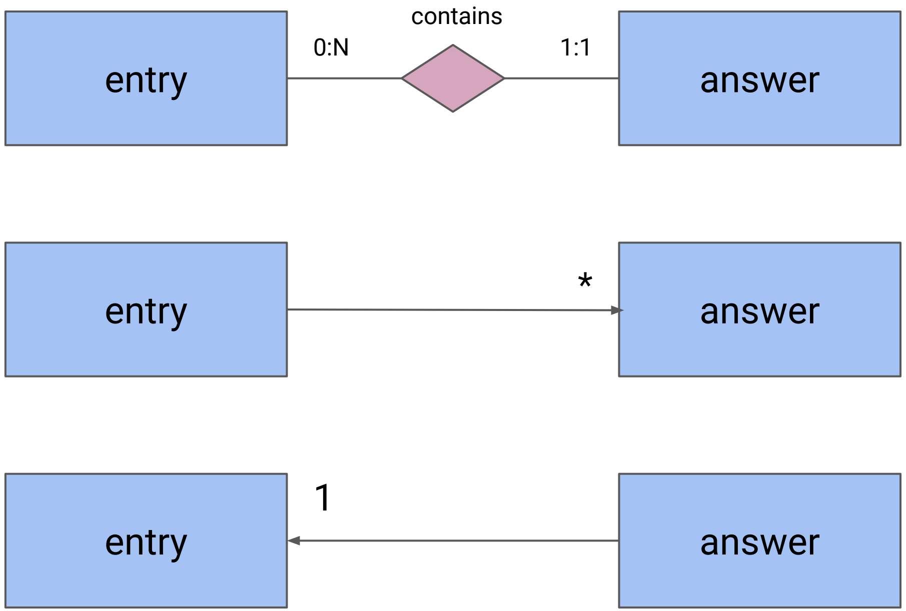
Bidirectional one-to-many association Entry to Answer.

- Answer is defined as the owner entity.
- PERSIST, REFRESH, MERGE, REMOVE are cascaded to the entity Answer. In particular, REMOVE allows to delete all the answers associated with the entry when the latter is deleted.
- Orphanremoval = true allows to remove answers when they are removed from the parent entity.
- FetchType is set to EAGER in order to retrieve immediately the answers linked to the entry when the inspection page is loaded.

Entry Entity - Named Queries

```
// Returns the answers of an entry. A DTO is used to simplify the access to data during templating. The ORDER BY is used
to display the questions in the order they were originally inserted.
@NamedQuery(name = "EntryEntity.getQuestionsAnswers", query = "SELECT NEW
it.polimi.db2.gma.GMAEJB.utils.QuestionAnswer(q.question, a.answer) FROM EntryEntity e INNER JOIN e.answers a INNER JOIN
a.question q WHERE e.questionnaire.id = :qid AND e.user.id = :uid ORDER BY q.id"),

// Returns the answered stats of an entry. A DTO is used to simplify the access to data during templating.
@NamedQuery(name = "EntryEntity.getStatsAnswers", query = "SELECT NEW it.polimi.db2.gma.GMAEJB.utils.StatsAnswers(s.age,
s.sex, s.expertiseLevel) FROM EntryEntity e INNER JOIN e.stats s WHERE e.questionnaire.id = :qid AND e.user.id = :uid"),

// Returns the entry by means of his composite primary key (userId + questionnaireId)
@NamedQuery(name = "EntryEntity.findByUserAndQuestionnaire", query = "SELECT e FROM EntryEntity e WHERE e.user.id =
:userId AND e.questionnaire.id = :questionnaireId")
```

Relationship Entry “**contains**” Answer



entry ➔ answer **`@OneToMany`** is necessary to get the answers of the entry.

answer ➔ entry **`@ManyToOne`** is not requested by the specifications but can be useful for future purposes.

Answer Entity

```
// Class AnswerEntityPK
@Embeddable

private int entryId;

private int questionId;


// Class AnswerEntity

@EmbeddedId
private AnswerEntityPK answerEntityPK;

@Column(name = "Answer", nullable = false, length = 45)
private String answer;

@ManyToOne
@JoinColumn(name = "EntryId")
@MapsId("entryId")
private EntryEntity entry;

@ManyToOne
@JoinColumn(name = "QuestionId")
@MapsId("questionId")
private QuestionEntity question;
```

Answer Entity - Motivations

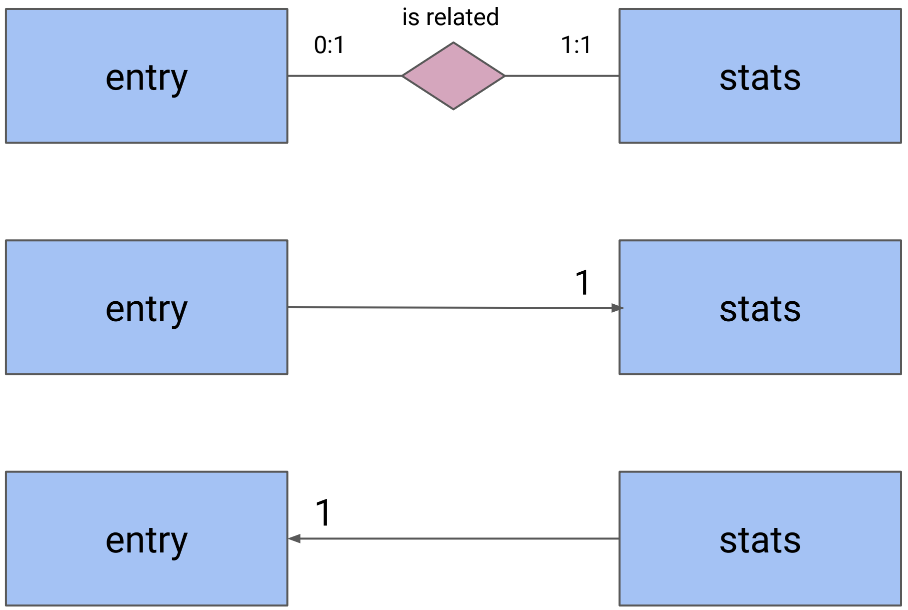
Bi-directional many-to-one association to Entry (Answer is the owner entity).

Bi-directional many-to-one association to Question (Answer is the owner entity).

The Answer entity has a composite primary key (userId + questionnaireId) which is implemented into JPA with the @Embeddable and @EmbeddedId annotations.

@MapsId annotates the relationship attribute to indicate that it is mapping the ID attribute as well. Note that physical mapping annotations (e.g., @Column) should not be specified on the attribute entryId nor questionId since @MapsId is indicating that the relationship attribute is where the mapping occurs.

Relationship Entry **“is related”** Stats



entry ➔ stats **@OneToOne** is necessary to contain the stats of users that fill the questionnaire.

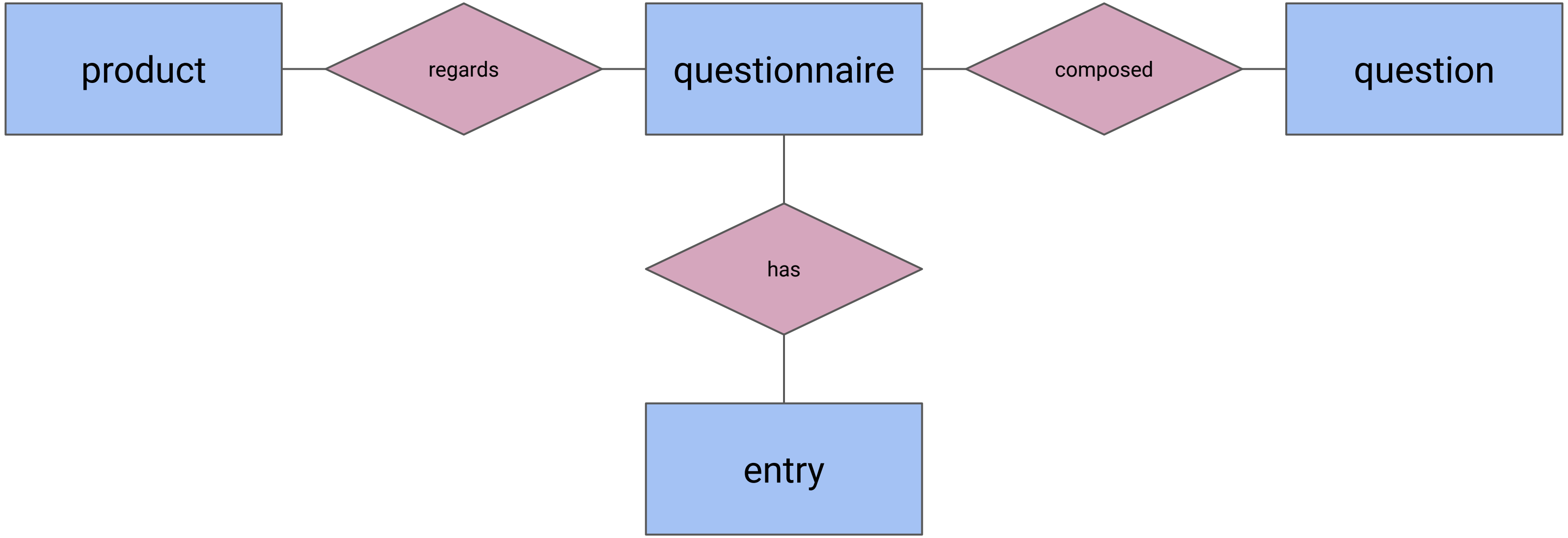
stats ➔ entry **@OneToOne** is not requested by the specifications but can be useful for future purposes.

Stats Entity

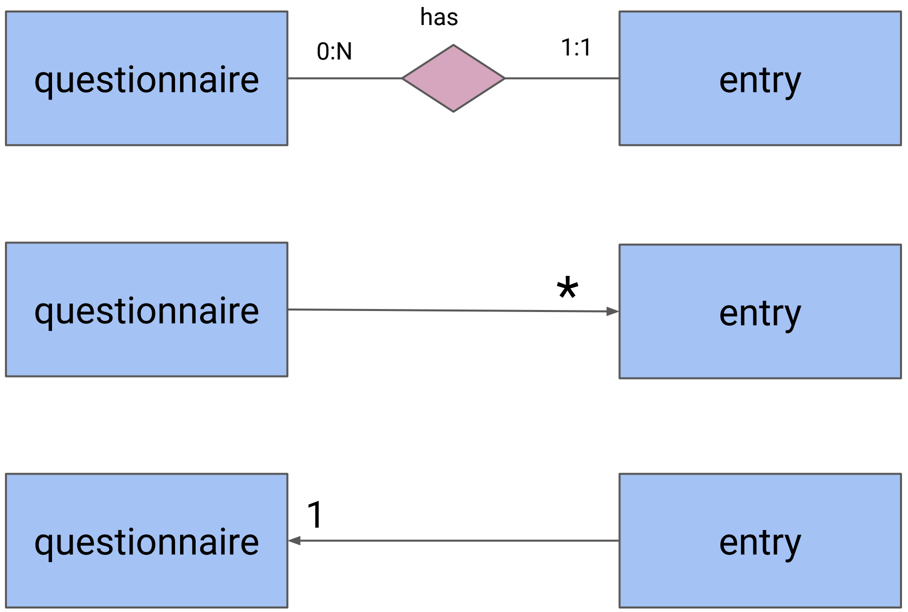
```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "Id", nullable = false)
private int id;

@Column(name = "Age", nullable = true)
private Integer age;

@Column(name = "Sex", nullable = true)
@Enumerated(EnumType.STRING)
private Sex sex;

@Column(name = "ExpertiseLevel", nullable = true)
@Enumerated(EnumType.STRING)
private ExpertiseLevel expertiseLevel;

@OneToOne
@JoinColumn(name = "EntryId")
private EntryEntity entry;
```

Bi-directional one-to-one association to Entry (Stats is the owner entity).

Questionnaire relationships



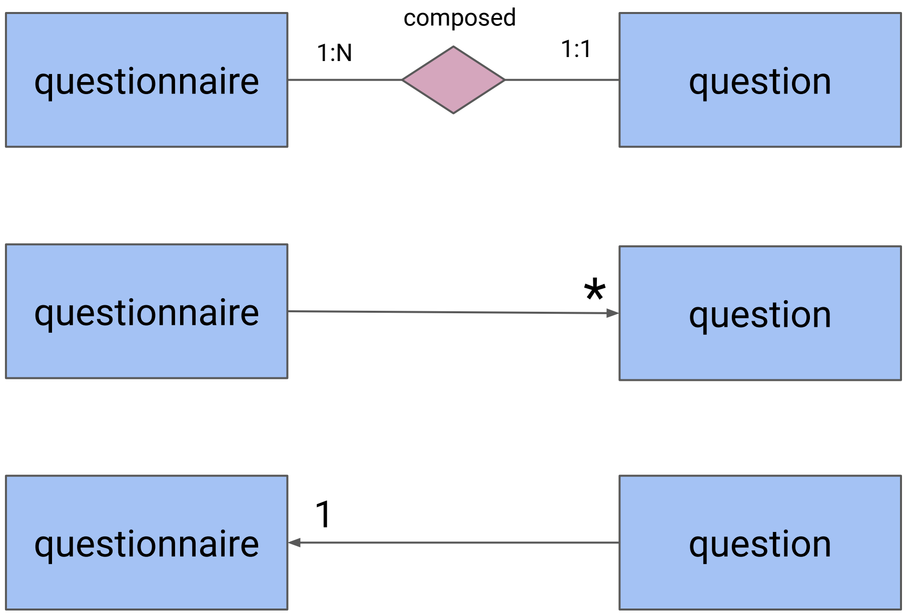
Relationship Questionnaire “**has**” Entry



questionnaire ➔ entry **`@OneToMany`**
is necessary to list all the entries inserted by users

entry ➔ questionnaire **`@ManyToOne`**
is not requested by the specifications, but could be useful in the future

Relationship Questionnaire “**composed**” Question



questionnaire ➔ question **`@OneToMany`** is necessary to list the questions a user needs to answer

question ➔ questionnaire **`@ManyToOne`**
is not requested by the specifications, but could be useful in the future

Questionnaire Entity

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "Id", nullable = false)
private int id;

@Column(name = "Date", nullable = false)
private Date date;

@ManyToOne
@JoinColumn(name = "ProductId")
private ProductEntity product;

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

@OneToMany(mappedBy = "questionnaire", cascade = {CascadeType.PERSIST, CascadeType.REMOVE,
       CascadeType.REFRESH, CascadeType.MERGE}, orphanRemoval = true)
private List<EntryEntity> entries = new ArrayList<>();
```

Questionnaire Entity - Motivations

Bi-directional many-to-one association to Product (Questionnaire is the owner entity).

Bidirectional one-to-many association Questionnaire to Question.

- Question is defined as the owner entity.
- PERSIST, REFRESH, MERGE, REMOVE are cascaded to the entity Question. In particular, REMOVE allows to delete all the questions associated with the questionnaire when the latter is deleted.
- Orphanremoval = true allows to remove questions when they are removed from the parent entity.
- FetchType is set to EAGER in order to retrieve immediately the questions linked to the questionnaire when the questionnaire of the day is shown to the user.

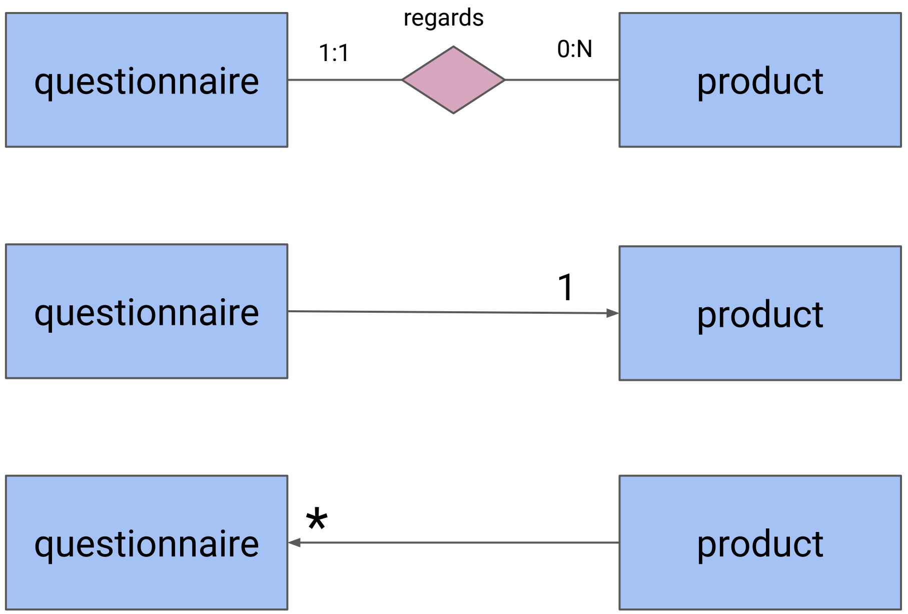
Bidirectional one-to-many association Questionnaire to Entry.

- Entry is defined as the owner entity.
- PERSIST, REFRESH, MERGE, REMOVE are cascaded to the entity Entry. In particular, REMOVE allows to delete all the entries associated with the questionnaire when the latter is deleted.
- Orphanremoval = true allows to remove entries when they are removed from the parent entity.

Questionnaire Entity - Named Queries

```
// Retrieve the Questionnaire belonging to a specific date
@NamedQuery(name = "QuestionnaireEntity.findByDate", query = "SELECT q FROM QuestionnaireEntity q WHERE q.date = :date")

// Retrieve a list of the Questionnaires existing before a given date
@NamedQuery(name = "QuestionnaireEntity.findAllUntilDate", query = "SELECT q FROM QuestionnaireEntity q WHERE q.date <
:date ORDER BY q.date DESC")

// Returns the list of all the Questionnaires present in the DataBase
@NamedQuery(name = "QuestionnaireEntity.findAll", query = "SELECT q FROM QuestionnaireEntity q")

// Retrieve the information about a Questionnaire. A DTO is used to simplify the access to data during templating.
@NamedQuery(name = "QuestionnaireEntity.getQuestionnairesInfos", query = "SELECT NEW
it.polimi.db2.gma.GMAEJB.utils.QuestionnaireInfo(q.id, q.date, p.name) FROM QuestionnaireEntity q INNER JOIN q.product p
ORDER BY q.date DESC")
```

Relationship Questionnaire “**regards**” Product



product ➔ questionnaire **`@OneToMany`**
is necessary to list the various questionnaire regarding a specific product

questionnaire ➔ product **`@ManyToOne`**
is not requested by the specifications, but we use it to retrieve the product related to a questionnaire

Product Entity

```java
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "Id", nullable = false)
private int id;

@Column(name = "Name", nullable = false, length = 45)
private String name;

@Column(name = "Image", nullable = false, length = 45)
private String image;

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

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

Product Entity - Motivations

Bidirectional one-to-many association Product to Review.

- Review is defined as the owner entity.
- No operations are cascaded.
- FetchType is set to EAGER in order to retrieve immediately the reviews linked to the product when the questionnaire of the day is shown to the user.

Bidirectional one-to-many association Product to Questionnaire.

- Questionnaire is defined as the owner entity.
- No operations are cascaded.

Product Entity - Named Queries

```
// Returns the list of all the Products present in the DataBase
@NamedQuery(name = "ProductEntity.findAll", query = "SELECT p FROM ProductEntity p")

// Retrieve the Product belonging to a specific date
@NamedQuery(name = "ProductEntity.findByDate", query = "SELECT p FROM ProductEntity p INNER JOIN p.questionnaires q WHERE
q.date = :date")
```

Trigger - Offensive Words detection

```
CREATE DEFINER=`dev`@`localhost` TRIGGER `answer_BEFORE_INSERT` BEFORE INSERT ON `answer` FOR EACH ROW BEGIN
    IF ((SELECT COUNT(*) FROM `offensiveword` WHERE NEW.Answer LIKE CONCAT('%', LOWER(word) ,'%')) > 0) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Offensive word detected!';
    END IF;
END
```

Before the insert of a new answer, the text of the response is checked. If an offensive word is found an exception is thrown and the transaction is aborted.
The exception will be catched by the EJB and the user will be banned from it.

Trigger - Update entry points

```
CREATE DEFINER=`dev`@`localhost` TRIGGER `answer_AFTER_INSERT` AFTER INSERT ON `answer` FOR EACH ROW BEGIN
    UPDATE `entry` e
    SET e.Points = e.Points + 1
    WHERE e.Id = NEW.EntryId;
END

CREATE DEFINER=`dev`@`localhost` TRIGGER `stats_AFTER_INSERT` AFTER INSERT ON `stats` FOR EACH ROW BEGIN
    UPDATE `entry` e
    SET e.Points = e.Points + 2 * ((NOT ISNULL(NEW.Age)) + (NOT ISNULL(NEW.Sex)) + (NOT ISNULL(NEW.ExpertiseLevel)))
    WHERE e.Id = NEW.EntryId;
END
```

After the insert of a new answer or stats, the trigger computes the **new** total point of the current entry.

Trigger - Update user points

```
CREATE DEFINER=`dev`@`localhost` TRIGGER `entry_AFTER_UPDATE` AFTER UPDATE ON `entry` FOR EACH ROW BEGIN
    IF NEW.Points <> OLD.Points THEN
             UPDATE `user` u
             SET u.Points = u.Points - OLD.Points + NEW.Points
             WHERE u.Id = NEW.UserId;
      END IF;
END

CREATE DEFINER=`dev`@`localhost` TRIGGER `entry_AFTER_DELETE` AFTER DELETE ON `entry` FOR EACH ROW BEGIN
      UPDATE `user` u
    SET u.Points = u.Points - OLD.Points
    WHERE u.Id = OLD.UserId
             AND OLD.IsSubmitted = 1;
END
```

After the update of an entry, if the points of the entry changed, the points of the user are recomputed.
After the delete of an entry, the total score of the users who submitted the entry is decremented by the entry points.

Business Tier Components - I

```
@Stateless EntryService
public Entry getUserQuestionnaireAnswers(int questionnaireID, int userID) throws BadEntryException
public Entry getDefaultQuestionnaireAnswers(int questionnaireID) throws BadEntryException
public void addEmptyEntry(int userId, int questionnaireId) throws BadEntryException
public void addEmptyEntryToday(int userId, int questionnaireId) throws BadEntryException, BadQuestionnaireException
public void addNewEntry(int userId, int questionnaireId, List<String> strAnswers, Integer age, Sex sex, ExpertiseLevel
expLevel) throws BadEntryException, BadWordException
public void addNewEntryToday(int userId, int questionnaireId, List<String> strAnswers, Integer age, Sex sex,
ExpertiseLevel expLevel) throws BadEntryException, BadWordException, BadQuestionnaireException
public EntryEntity getEntryByIds(int questionnaireId, int userId) throws BadEntryException

@Stateless QuestionnaireService
public QuestionnaireEntity findQuestionnaireById(int id)
public List<QuestionnaireEntity> findAllQuestionnaires()
public QuestionnaireEntity findQuestionnaireByDate(LocalDate localDate)
public QuestionnaireEntity addNewQuestionnaire(LocalDate localDate, int productId, List<String> strQuestions) throws
BadProductException, BadQuestionnaireException
public void deleteQuestionnaire(int questionnaireId) throws BadQuestionnaireException
public List<QuestionEntity> findAllQuestionsByQuestionnaire(int questionnaireId)
public List<QuestionnaireInfo> getQuestionnairesInfos()
public List<QuestionnaireEntity> findQuestionnairesUntil(LocalDate localDate)
```

Business Tier Components - II

```
@Stateless ProductService
public List<ProductEntity> findAllProducts()
public ProductEntity findProductByDay(Date date)

@Stateless AdminService
public AdminEntity checkCredentials(String username, String password) throws CredentialsException,
NonUniqueResultException

@Stateless UserService
public UserEntity findUserById(int userId)
public UserEntity findUserByUsername(String username)
public UserEntity findUserByEmail(String email)
public UserEntity checkCredentials(String username, String password) throws CredentialsException,
NonUniqueResultException
public UserEntity addNewUser(String username, String password, String email) throws CredentialsException
public List<LeaderboardRow> getLeaderboardByDate(Date date)
public List<UserInfo> getQuestionnaireUserInfo(int questionnaireID, int isSubmitted) throws BadQuestionnaireException
public void blockUser(int userId)
public void addLoginLog(int userId) throws BadUserException
```

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