SR03 : RAPPORT 1

Étudiants : María Clara BEZZECCHI et Théodore BOURGEON

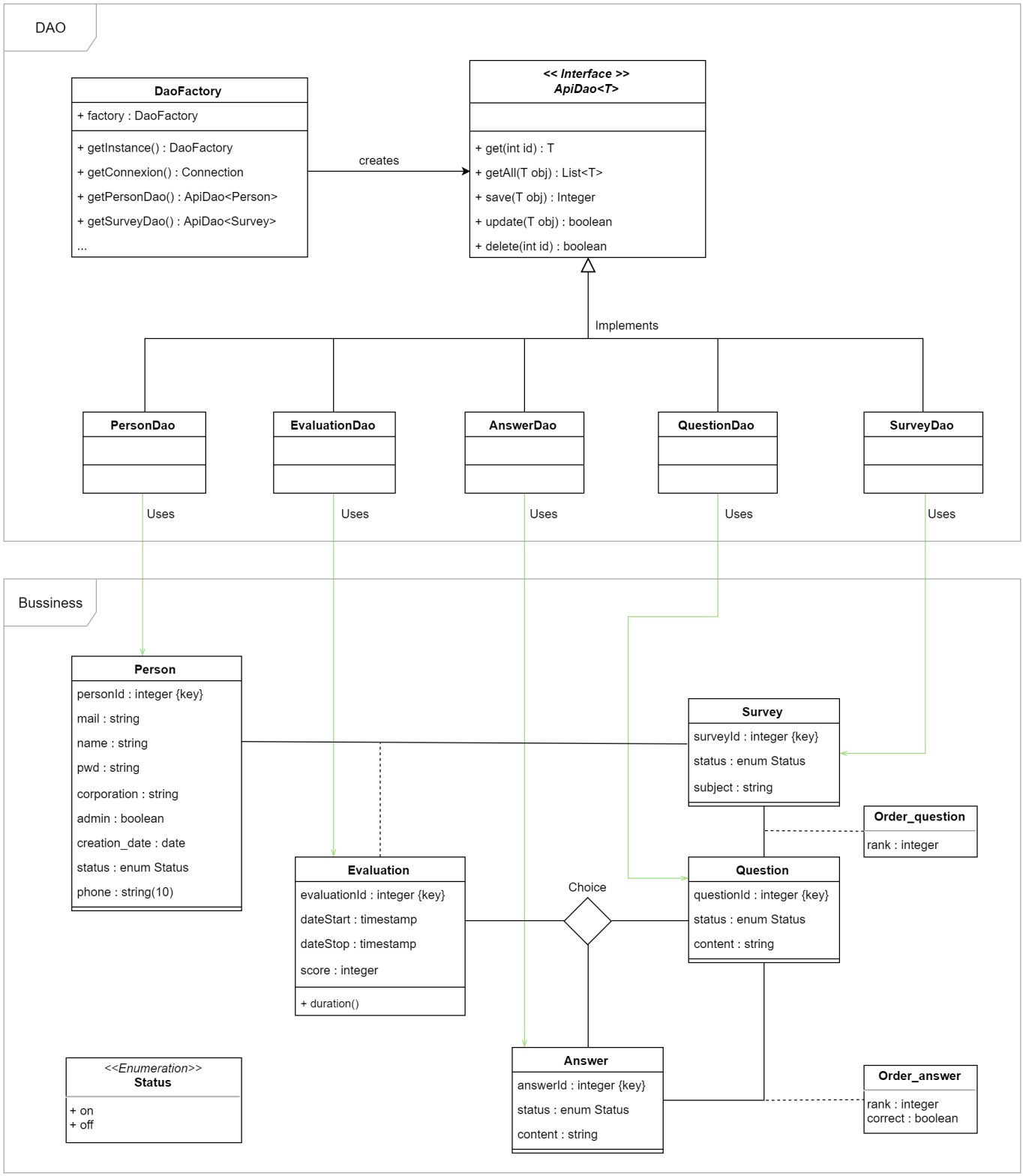
Objectifs : Diagramme de classes + illustration CRUD
Date de rendu : 07/05

Introduction

Ce rapport vise à donner un aperçu de l'avancement de notre projet SR03: Application d'évaluation de stagiaire. Tout d'abord en rappelant le modèle de nos données et son implémentation (java et SQL).
Puis, en présentant notre implémentation du design pattern DAO vu en cours. Celui-ci vise à permettre une abstraction de l'accès à notre base de données pour nos traitements effectués dans l'application J2EE.

I. Diagramme de classes

Ci-dessous nous retrouvons notre diagramme de classes.
Le paquetage Bussiness représente notre modèle de données tandis que le paquetage DAO, lui, liste les classes implémentées pour mettre en place le design pattern éponyme.



Justification des choix du modèle de données

Pour convenir au cahier des charges du projet, une relation ternaire à été mise en place entre la table des évaluations, des questions et des réponses. En effet, chaque questionnaire se voit attribuer des questions dans un ordre précis. Les questions peuvent être les memes dans plusieurs questionnaires.
De plus chaque question est reliée à plusieurs réponses (qui peuvent également changer pour une même question).
La table d'association Choice contiendra alors, pour chaque évaluation, pour chaque

question, les réponses de l'utilisateur.
Une évaluation est un parcours: pour une personne donnée, pour un questionnaire qui donnera des informations sur le score réalisé.

Dans un premier temps nous avions créé un heritage de Person pour les stagiaires et les administrateurs. Mais pour des raisons d'implémentation, nous avons intégré cette information au sein de la classe mère Person avec un attribut booléen admin.

II. Classes

Dans cette partie nous montrons l'implémentation de notre modèle de données/objets métiers sous forme de différentes classes java.

Person

```java
package model;

import java.time.LocalDateTime;

public class Person {
    private Integer id;
    private String mail;
    private String name;
    private String pwd;
    private String corporation;
    private String phone;
    private Status status;
    private LocalDateTime creation_date;
    private boolean admin;

    public Person () {
    }

    public Integer getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }

    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCorporation() {
        return corporation;
    }
    public void setCorporation(String corporation) {
        this.corporation = corporation;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    public Status getStatus() {
        return status;
    }
    public void setStatus(Status status) {
        this.status = status;
    }
    public LocalDateTime getCreation_date() {
        return creation_date;
    }
    public void setCreation_date(LocalDateTime creation_date) {
        this.creation_date = creation_date;
    }

    public String toString() {
        return this.name + " - " + this.mail;
    }
```

```
72
73      public boolean isAdmin() {
74          return admin;
75      }
76
77      public void setAdmin(boolean admin) {
78          this.admin = admin;
79      }
80  }
```

Survey

```
1   package model;
2
3   public class Survey {
4       private int id;
5       private String subject;
6       private Status status;
7
8       public Survey() {
9       }
10
11      public int getId() {
12          return id;
13      }
14      public void setId(int id) {
15          this.id = id;
16      }
17      public String getSubject() {
18          return subject;
19      }
20
21      public void setSubject(String subject) {
22          this.subject = subject;
23      }
24      public Status getStatus() {
25          return status;
26      }
27      public void setStatus(Status status) {
28          this.status = status;
29      }
30  }
31
```

Question

```java
package model;

import java.util.ArrayList;

public class Question {
    private int id;
    private String content;
    private Status status;
    private ArrayList<Ordered_answer> answers;

    public Question(int id, String content, Status status) {
        this.id = id;
        this.content = content;
        this.status = status;
        this.answers = new ArrayList<Ordered_answer>();
    }

    // no setter
    public int getId() {
        return id;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public Status getStatus() {
        return status;
    }
    public void setStatus(Status status) {
        this.status = status;
    }

    class Ordered_answer {
        private int order;
        private Answer answer;
        private boolean correct;

        public Ordered_answer(int order, Answer answer, boolean correct) {
            this.order = order;
            this.answer = answer;
            this.correct = correct;
        }

        public int getOrder() {
            return order;
        }
        public void setOrder(int order) {
            this.order = order;
        }
        public Answer getAnswer() {
            return answer;
        }
          public void setAnswer(Answer answer) {
              this.answer = answer;
          }
          public boolean isCorrect() {
              return correct;
          }
          public void setCorrect(boolean correct) {
              this.correct = correct;
          }
      }

}
```

Answer

```java
package model;

public class Answer {
    private int id ;
    private String content ;
    private Status status ;

    public Answer(int id, String content, Status status) {
        this.id = id;
        this.content = content;
        this.status = status;
    }

    // no setter
    public int getId() {
        return id;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public Status getStatus() {
        return status;
    }
    public void setStatus(Status status) {
        this.status = status;
    }


}
```

Status

Status.java

```java
package model;

import java.util.NoSuchElementException;

public enum Status {
    ON, OFF;

    public static Status fromString(String string) {
          for (Status status : values()) {
            if (status.name().equalsIgnoreCase(string)) {
              return status;
            }
          }
          throw new NoSuchElementException("Element with string " + string + " has not been found");
        }
}
```

III. Pattern DAO

Ici nous montrons l'implémentation de la structure de notre DAO.

ApiDao.java

Tout d'abord, nous avons donc créé une interface qui liste les opérations SCRUD (search, create, read, update et delete). Ainsi, nos différents DAO devront implémenter cette interface.

ApiDao.java

```java
package dao;

import java.util.List;

public interface ApiDao<T> {
    T get(int id);
    List<T> getAll(T t);
    Integer save(T t);  // return new id or null => Integer type (nullable)
    boolean update(T t);
    boolean delete(T t);
}
```

DAOFactory.java

Nous avons également fait le choix de créer une factory pour permettre plus tard de récupérer via celle-ci nos différents DAO correctement configurés.
Nous avons décidé d'en faire un singleton pour des raisons d'optimisation, ainsi que pour garantir une unique récupération du driver jdbc mysql: lors de la création de l'unique instance.

```java
package dao;

import java.sql.Connection;

public class DAOFactory {

    private static DAOFactory factory;

    private static final String DB_URL = "jdbc:mysql://localhost:8889/evaluation_stagiaire?serverTimezone=UTC";
    private static final String DB_USER = "root";
    private static final String DB_PSWD = "root";
    private static final String DRIVER = "com.mysql.cj.jdbc.Driver";

    private DAOFactory() {
        try {
            Class.forName(DRIVER);
        } catch ( ClassNotFoundException e ) {
            e.printStackTrace();
        }
    }

    /* r@cup@rer singleton factory */
    public static DAOFactory getInstance() {
        if(factory == null) {
            factory = new DAOFactory();
        }
        return factory;
    }

    /* M@thode charg@e de fournir une connexion @ la base de donn@es */
     /* package */ Connection getConnection() throws SQLException {
        return DriverManager.getConnection(DB_URL, DB_USER, DB_PSWD );
    }

    /* M@thodes de r@cup@ration de l'impl@mentation des diff@rents DAO  */
    public ApiDao<Person> getPersonDao() {
        return new PersonDao( this );
    }

    public ApiDao<Survey> getSurveyDao() {
        return new SurveyDao( this );
    }
```

IV. Détails des opérations SCRUD pour Person

Ici nous montrons une implémentation détaillée des opérations SCRUD de notre DAO pour un objet métier : l'utilisateur (Person).
Celles-ci sont donc dans la classe PersonDao qui implémente l'interface ApiDao présentée précédemment.

Classe PersonDao

Avant de commencer, dans l'image suivante, qui représente le squelette de notre classe, nous voyons deux aspects importants pour pouvoir mettre en place le DAO.

Tout d'abord la gestion d'une connexion avec la base de données. En effet, grace au fait que l'on possède une référence sur notre DaoFactory, nous pouvons récupérer facilement une connexion. La méthode closeAll() elle permet de fermer proprement toute connexion.

Enfin, le deuxième aspect important est de définir une méthode de mappage qui, à partir d'un résultat de requête, peut reconstruire l'objet métier java avec les informations récupérées.

```java
package dao;

import java.sql.Connection;..

public class PersonDao implements ApiDao<Person> {
    private DAOFactory daoFactory;

    PersonDao(DAOFactory daoFactory){
        this.daoFactory = daoFactory;
    }

    private static void closeAll(ResultSet result, PreparedStatement statement, Connection connexion) {
        if ( result != null ) {
            try {
                /* On commence par fermer le ResultSet */
                result.close();
            } catch ( SQLException ignore ) { }
        }
        if ( statement != null ) {
            try {
                /* Puis on ferme le Statement */
                statement.close();
            } catch ( SQLException ignore ) { }
        }
        if ( connexion != null ) {
            try {
                /* Et enfin on ferme la connexion */
                connexion.close();
            } catch ( SQLException ignore ) { }
        }
    }


    private Person map(ResultSet rs) throws SQLException {
        Person person = new Person();

        // maj attributs, mail, pwd, name, society, phone, status_person, creation_date
        person.setId(rs.getInt("personId"));
        person.setMail(rs.getString("mail"));
        person.setPwd(rs.getString("pwd"));
        person.setAdmin(rs.getBoolean("admin"));
        person.setName(rs.getString("name"));
        person.setCorporation(rs.getString("society"));
        person.setPhone(rs.getString("phone"));
        person.setStatus(Status.fromString(rs.getString("status_person")));
        person.setCreation_date(rs.getTimestamp("creation_date").toLocalDateTime());

        return person;
    }
```

Concentrons nous maintenant sur l'implémentation des méthodes SCRUD.

SEARCH

La méthode search prend en argument un objet person. En fonction de ses caractéristiques : si le champs mail et/ou mdp est renseigné on va pouvoir filtrer différents notre requete et récupérer la liste correspondante. On fera appel à la méthode map afin que, pour chaque objet retourner, on complète la liste d'objet personne avec un objet personne rempli.

```
@Override
public List<Person> getAll(Person model) {
    Connection connexion = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    List<Person> personList = new ArrayList<Person>();

    try {
        // r?cup?ration connexion depuis la Factory
        connexion = daoFactory.getConnection();
        String req = "SELECT personId, mail, pwd, admin, name, society, phone, status_person, creation_date FROM person";

        // build on model
        List<String> filters = new ArrayList<String>();
        if (model != null) {
            if(model.getMail() != null) {
                if (filters.isEmpty()) { req += " WHERE";}
                else { req += ", AND";}
                filters.add("mail");
                req += " mail=?";

            }
            if(model.getPwd() != null) {
                if (filters.isEmpty()) { req += " WHERE";}
                else { req += " AND";}
                req += " pwd=?";
                filters.add("pwd");
            }
        }
        // pr?paration requ?te
        preparedStatement = connexion.prepareStatement(req);
        int index = 1;
        for (String filter: filters) {
            switch(filter) {
                case "mail" :
                    preparedStatement.setString(index, model.getMail());
                    break;
                case "pwd" :
                    preparedStatement.setString(index, model.getPwd());
                    break;
            }
            index += 1;
        }
        resultSet = preparedStatement.executeQuery();

        /* Parcours des lignes resultSet retourn?es */
        while ( resultSet.next() ) {
            personList.add(map(resultSet));
        }
    } catch ( SQLException e ) {
        e.printStackTrace();
    } finally {
        closeAll(resultSet, preparedStatement, connexion );
    }
    return personList;
}
```

CREATE

Pour la création d'un utilisateur, on prépare la requete avec les différents champs renseignés dans l'objet renseigné en argument de la fonction et on l'exécute. On récupère l'id de statut avant de le renvoyer.

```
    @Override
    public Integer save(Person person) {
        Integer generatedKey = null;
        Connection connexion = null;
        PreparedStatement preparedStatement = null;

        try {
            // r@cup@ration connexion depuis la Factory
            connexion = daoFactory.getConnection();

            // pr@paration requ@te
            preparedStatement = connexion.prepareStatement(
                    "INSERT INTO person (mail, pwd, admin, name, society, phone, status_person, creation_date)"
                    + " VALUES(?, ?, ?, ?, ?, ?, ?, ?);",
                    PreparedStatement.RETURN_GENERATED_KEYS);

            preparedStatement.setString(1, person.getMail());
            preparedStatement.setString(2, person.getPwd());
            preparedStatement.setBoolean(3, person.isAdmin());
            preparedStatement.setString(4, person.getName());
            preparedStatement.setString(5, person.getCorporation());
            preparedStatement.setString(6, person.getPhone());
            preparedStatement.setString(7, person.getStatus().name());
            preparedStatement.setTimestamp(8, Timestamp.valueOf(person.getCreation_date()));

            /* Ex@cution de la requ@te */
            int statut = preparedStatement.executeUpdate();

            /* Si la requ@te s'est bien @x@cut@e */
            if ( statut != 0 ) {
                // on r@cup@re cl@e g@n@r@e
                ResultSet rs = preparedStatement.getGeneratedKeys();

                if (rs.next()) {
                    generatedKey = rs.getInt(1);
                }
            }
        } catch ( SQLException e ) {
            e.printStackTrace();
        } finally {
            closeAll( null, preparedStatement, connexion );
        }
        return generatedKey;
    }
```

READ

Pour la récupération d'un utilisateur, il nous suffit de filtrer sur un id la table des utilisateurs et de renvoyer le profil correspondant dans un objet person grace à la fonction map.

```
@Override
public Person get(int id){
    Connection connexion = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    Person person = null;

    try {
        // r@cup@ration connexion depuis la Factory
        connexion = daoFactory.getConnection();

        // pr@paration requ@te
        preparedStatement = connexion.prepareStatement(
                "SELECT personId, mail, pwd, admin, name, society, phone, status_person, creation_date "
                + "FROM person WHERE personId = ?"
                );
        preparedStatement.setInt(1, id);

        resultSet = preparedStatement.executeQuery();
        /* Parcours de la ligne de donn@es de l'@ventuel ResulSet retourn@ */
        if ( resultSet.next() ) {
            person = map(resultSet);
        }
    } catch ( SQLException e ) {
        e.printStackTrace();
    } finally {
        closeAll(resultSet, preparedStatement, connexion );
    }

    return person;
}
```

UPDATE

L'update met simplement à jour un objet person en filtrant sur son id et renvoie le statut de la requete.

```
@Override
public boolean update(Person person) {
    boolean status = false;
    Connection connexion = null;
    PreparedStatement preparedStatement = null;

    try {
        // r@cup@ration connexion depuis la Factory
        connexion = daoFactory.getConnection();

        // pr@paration requ@te
        preparedStatement = connexion.prepareStatement(
                "UPDATE person SET mail=?, pwd=?, admin=?, name=?, society=?, phone=?, status_person=? WHERE personId = ?");

        preparedStatement.setString(1, person.getMail());
        preparedStatement.setString(2, person.getPwd());
        preparedStatement.setBoolean(3, person.isAdmin());
        preparedStatement.setString(4, person.getName());
        preparedStatement.setString(5, person.getCorporation());
        preparedStatement.setString(6, person.getPhone());
        preparedStatement.setString(7, person.getStatus().name());
        preparedStatement.setInt(8, person.getId());

        // Ex@cution de la requ@te => OK = valeur retourn@ee non nulle
        status = preparedStatement.executeUpdate() != 0;

    } catch ( SQLException e ) {
        e.printStackTrace();
    } finally {
        closeAll( null, preparedStatement, connexion );
    }

    return status;
}
```

DELETE

La suppression d'un utilisateur se fait par filtrage sur son id et renvoie également le statut de la requete.

```
@Override
public boolean delete(Person person) {
    boolean status = false;
    Connection connexion = null;
    PreparedStatement preparedStatement = null;

    try {
        // r@cup@ration connexion depuis la Factory
        connexion = daoFactory.getConnection();

        // pr@paration requ@te
        preparedStatement = connexion.prepareStatement("DELETE FORM person WHERE personId = ?");

        preparedStatement.setInt(1, person.getId());

        // Ex@cution de la requ@te => OK si valeur retourn@e non nulle
        status = preparedStatement.executeUpdate() != 0;

    } catch ( SQLException e ) {
        e.printStackTrace();
    } finally {
        closeAll( null, preparedStatement, connexion );
    }

    return status;
}
```

ANNEXE

Schéma de base de données

Pour des raisons de facilité d’implémentation, nous avons rajouté des clés artificielles aux tables, même si certaines possédaient déjà une clé naturelle. (Ex: mail pour les personnes).

```
CREATE TABLE PERSON (
   personId INTEGER AUTO_INCREMENT PRIMARY KEY,
   mail VARCHAR(255) UNIQUE NOT NULL,
   pwd VARCHAR(255) NOT NULL,
   admin BOOLEAN,
   name VARCHAR(255) NOT NULL,
   society VARCHAR(255),
   phone VARCHAR(10),
   status_person ENUM('on', 'off'),
   creation_date TIMESTAMP,
   CHECK (char_length(pwd) >6)
   );


CREATE TABLE SURVEY (
   surveyId INTEGER AUTO_INCREMENT PRIMARY KEY,
   subject VARCHAR(255) NOT NULL,
   status_survey ENUM('on', 'off')
   );


CREATE TABLE EVALUATION (
   evaluationId INTEGER AUTO_INCREMENT PRIMARY KEY,
   personId INTEGER NOT NULL,
   surveyId INTEGER NOT NULL,
   dateStart TIMESTAMP NOT NULL,
   dateStop TIMESTAMP  DEFAULT 'SELECT SYSDATE();',
   score INTEGER,
   FOREIGN KEY (personId) REFERENCES PERSON(personId),
   FOREIGN KEY (surveyId) REFERENCES SURVEY(surveyId)
   );


CREATE TABLE QUESTION (
   questionId INTEGER AUTO_INCREMENT PRIMARY KEY,
   content VARCHAR(255) NOT NULL,
   status_survey ENUM('on', 'off')
   );


CREATE TABLE ANSWER (
   answerId INTEGER AUTO_INCREMENT PRIMARY KEY,
   content VARCHAR(255) NOT NULL,
   status_answer ENUM('on', 'off')
   );


CREATE TABLE ORDER_QUESTION (
   surveyId INTEGER,
   questionId INTEGER,
   rank INTEGER NOT NULL,
   UNIQUE(questionId, rank),
   PRIMARY KEY (surveyId, questionId),
   FOREIGN KEY (surveyId) REFERENCES SURVEY(surveyId),
```

```
    FOREIGN KEY (questionId) REFERENCES QUESTION(questionId)
    );


CREATE TABLE ORDER_ANSWER (
    questionId INTEGER,
    answerId INTEGER,
    rank INTEGER NOT NULL,
    correct BOOLEAN,
    UNIQUE(answerId, rank),
    PRIMARY KEY (questionId,answerId),
    FOREIGN KEY (questionId) REFERENCES QUESTION(questionId),
    FOREIGN KEY (answerId) REFERENCES ANSWER(answerId)
    );


CREATE TABLE CHOICE (
    evaluationId INTEGER,
    questionId INTEGER NOT NULL,
    answerId INTEGER NOT NULL,
    PRIMARY KEY (evaluationId,questionId,answerId),
    FOREIGN KEY (evaluationId) REFERENCES EVALUATION(evaluationId),
    FOREIGN KEY (questionId) REFERENCES QUESTION(questionId),
    FOREIGN KEY (answerId) REFERENCES ANSWER(answerId)
    );
```