Gasparello Alessandro - 878169 Pizzolato Davide - 881347 Simonato Alessandro - 882339

Gruppo : Mc.Noget

Progetto basi di dati 2020/2021

Sito web per la creazione di questionari online

Introduzione:

Il progetto in questione consiste nella creazione di un sito web dotato di un sistema di login sicuro, in grado di consentire agli utenti registrati nel sito di creare i propri questionari online e di renderli successivamente compilabili da qualsiasi tipologia di utente, quindi sia da utenti anonimi (non registrati nella piattaforma) sia utenti non anonimi.

Il seguente documento cercherà di illustrare tutte le funzionalità implementate motivando anche tutte le decisioni intraprese durante tutto il processo di progettazione, quindi partendo dal database fino al front-hand dell'applicazione. Verranno inoltre illustrate alcune delle query maggiormente utilizzate o comunque "degne di nota".

- **Database e DBMS :**

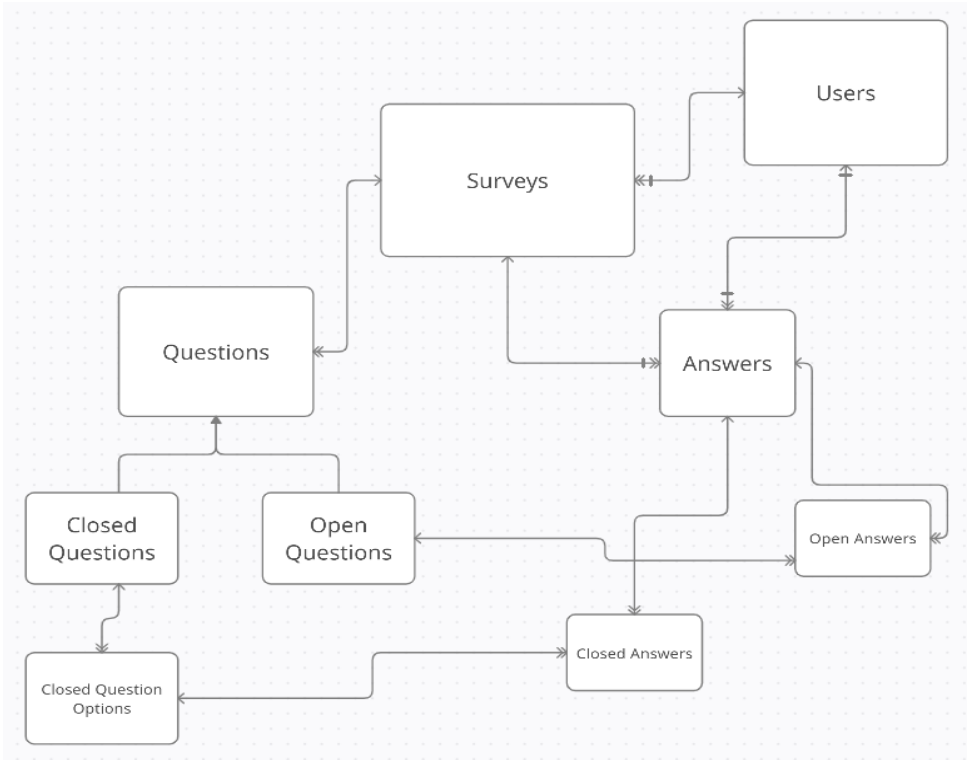

Fig. 1 – Schema a oggetti del database

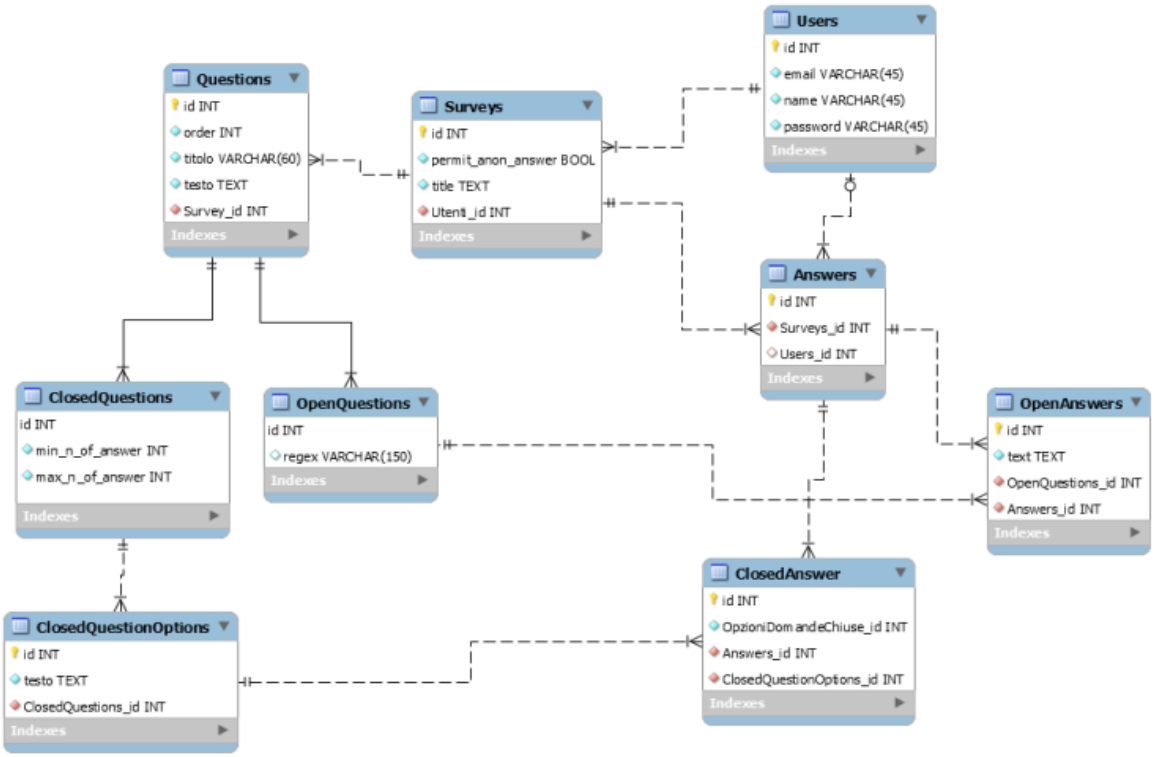

Fig. 2 – Schema relazionale database

Il database effettivo finale differisce per qualche campo che abbiamo deciso di cambiare in corso d'opera, per migliorarne l'efficienza e per renderlo il più esaustivo possibile, tuttavia la logica è rimasta la stessa, quindi oltre a qualche leggera modifica ai campi non è stato toccato altro.

Il concetto di partenza è che un questionario noi lo possiamo vedere come un insieme articolato di domande e relative risposte, le domande possono essere di differente tipo, abbiamo le domande chiuse e le domande aperte, per le domande chiuse abbiamo differenti opzioni, associate ad un numero arbitrario di riposte che oscillerà nel range fornito dal creatore del survey. Le domande aperte invece sono state gestite tramite l'uso di **RegEx** (Regular Expressions) che consentono all'utente di creare domande che richiederanno diverse tipologie di risposte, il tutto sfruttando un'unica tabella per tutte le differenti tipologie. Infine ad ogni utente, di cui richiederemo una mail e una password, saranno associati i differenti survey da esso creato e le risposte date in altri survey sia che abbia effettuato l'autenticazione nella piattaforma sia che non sia autenticato o eventualmente registrato e di conseguenza anonimo.

Come DBMS d'appoggio abbiamo deciso di utilizzare PostgreSQL (visto che è stato anche quello principalmente affrontato durante le lezioni del corso)

In aggiunta in questa sezione abbiamo il metodo di autenticazione specificato nel file pg_hba.conf (file di Postgres configurabile a piacimento per definire il processo di autenticazione al Database) :

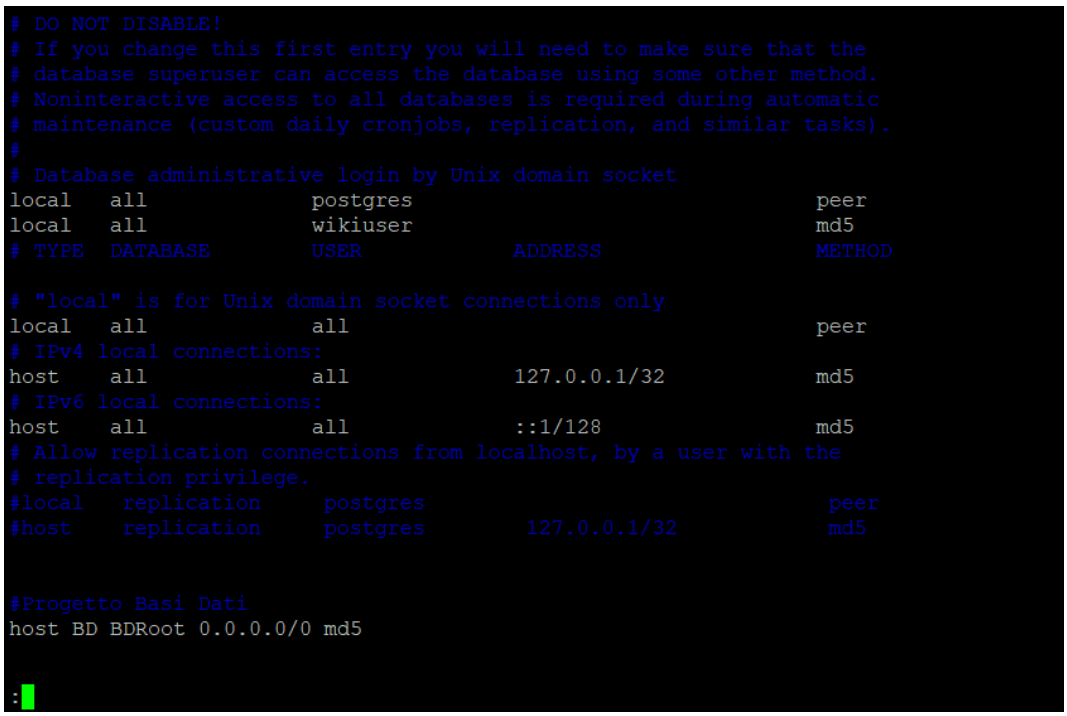

Fig. 3 – File pg_hba.cong

- ## Comunicazione con DBMS

Come tecnica per Interfacciarci con il DBMS sottostante abbiamo deciso di utilizzare la libreria **SQLAlchemy ORM**, ossia un **Object Relational Mapping**, con un approccio basato su l'uso di API, in grado di offrirci un mapping diretto tra classi del linguaggio utilizzato (nel nostro caso Phyton) e le tabelle che abbiamo definito nel DBMS.

- ## Applicazione web

Per sviluppare l'applicazione web è stato utilizzato un framework per Phyton chiamato Flask, che ha consentito lo sviluppo di tutto il Logic Tier della nostra applicazione web utilizzando il linguaggio di programmazione specifico.

Flask consente di utilizzare delle route per associare le varie funzioni create in Phyton ad una specifica sezione del nostro server, inoltre ci dà la possibilità di suddividere la nostra applicazione web in più moduli indipendenti, chiamati Blueprint, dando ordine al lavoro finale. Noi abbiamo deciso di sfruttare questa funzionalità suddividendo

l'applicazione in 4 Blueprint :

- answer : su cui viene gestita tutta la parte d'inserimento e visualizzazione delle risposte
- front : su cui sono gestite le route per la homepage e l'area riservata
- auth : su cui sono gestite tutte le route per la gestione del login, del sign up, del cambio password e log out
- survey : su cui sono gestite tutte le route per l'inserimento e eliminazione dei questionari

- **Gestione accessi e sicurezza**

Per gestire gli accessi alla nostra piattaforma abbiamo deciso di utilizzare sempre Flask, con la libreria di autenticazione Flask-Login che ci consente di implementare il sistema di autenticazione e accesso degli utenti tramite le sessioni native di Flask.

Tramite l'utilizzo dell'annotazione @login_required posta sopra le varie funzioni di Phyton associate alle diverse route, possiamo decidere quali sezioni saranno accessibili esclusivamente dopo aver effettuato correttamente l'autenticazione nella piattaforma.

Inoltre, se si sta tentando di accedere ad una sezione per la quale è necessario essere loggati si verrà automaticamente reindirizzati alla pagina di autenticazione, e se si effettuerà il login correttamente si verrà riportati alla pagina precedente.

- **Template sito web**

Il sito è stato organizzato in diversi template, in particolare abbiamo utilizzato un template chiamato base_generic.html che sarà poi ereditato anche da tutti gli altri visto che contiene tutti gli elementi che devono essere condivisi dalle diverse pagine. Poi abbiamo strutturato il tutto aggiungendo le pagine dedicate a : login/sign up, creazione nuovi survey, risposta a nuovi survey e area riservata con visualizzazione risultati dei propri survey o cancellazione/condivisione survey creati.

Funzionalità Principali :

- ### Creazione questionario

Tramite l'apposita sezione Survey raggiungibile dalla toolbar, un utente registrato può creare il suo personale questionario che successivamente potrà condividere per ricevere le risposte da qualsiasi altro utente interessato a fornire una propria riposta.

Il processo di creazione avviene inizialmente tramite codice JS, i cui risultati saranno poi passati in un JSON da cui reperiremo le informazioni necessarie per inserire il questionario con tutte le relative domande all'interno della tabella corretta del database, ossia la tabella survey.

```python
@survey.route('/', methods=['POST'])
@login_required
def insert_survey():
    data = request.json
    survey = Survey(title=data['title'],
                    permit_anon_answer=data['permit_anon_answer'],
                    author_id=current_user.get_id(),
                    questions=[])
    for question_row in data['questions']:
        question = Question(order=question_row['order'],
                            title=question_row['title'],
                            text=question_row['text'])
        if question_row['type'] == str(QuestionTypes.OpenQuestion.value):
            question.open_question = OpenQuestion(regex=question_row['regex'],
                                                  regex_description=question_row['regex_description'],
                                                  mandatory=question_row['mandatory'])
        elif question_row['type'] == str(QuestionTypes.ClosedQuestion.value):
            question.closed_question = ClosedQuestion(min_n_of_answer=question_row['min'],
                                                      max_n_of_answer=question_row['max'])
            for option_row in question_row['options']:
                question.closed_question.closed_question_options.append(
                    ClosedQuestionOption(order=option_row['order'], text=option_row['text'])
                )
        else:
            pass  # TODO Exception
        survey.questions.append(question)

    session = Session()
    session.add(survey)
    session.commit()
    return flask.Response(status=200)
```

Fig. 4 – API inserimento nuovo questionario

- ### Risposte a questionario

Tramite appositi link condivisi dal creatore di un questionario, ogni utente registrato o anonimo (se consentito dall'utente creatore), potrà fornire le proprie risposte al suddetto questionario. Questo processo come quello per la creazione dei questionari avviene inizialmente tramite codice JS, i cui risultati saranno poi passati in un JSON da cui reperiremo le informazioni necessarie per salvare le diverse

risposte date da quell'utente al rispettivo questionario, nella tabella answers del database.

```python
@answer.route('/', methods=['POST'])
# @login_required
def insert_answer():
    data = request.json
    session = Session()
    survey = session.query(Survey).get(data['survey_id'])
    for answer_row in data['answers']:
        answer = Answer(user_id=current_user.get_id())
        if answer_row['type'] == str(AnswerTypes.OpenAnswer.value):
            answer.open_answers.append(OpenAnswer(open_question_id=answer_row['open_question_id'], text=answer_row['text']))
        elif answer_row['type'] == str(AnswerTypes.ClosedAnswer.value):
            answer.closed_answers.append(ClosedAnswer(closed_question_option_id=answer_row['closed_question_option_id']))
        else:
            pass  # TODO Exception
        survey.answers.append(answer)
    session.commit()
    return data
```

Fig. 5 – API inserimento nuove risposte

- **Login / SignUp**

Tutta la parte di gestione accessi e sicurezza è stata gestita con la libreria Flask-Login, che ci ha fornito tutte le funzionalità necessarie a garantire la sicurezza della piattaforma nelle fasi di accesso o iscrizione alla stessa.

Abbiamo inoltre deciso di attivare un cookie in fase di accesso tramite il passaggio dei seguenti argomenti ai parametri remember e duration all'interno della funzione login_user di Flask-Login :

```python
login_user(user, remember=True, duration=datetime.timedelta(minutes=20))
```

con remember a True indichiamo che si andrà a generare un cookie per ricordarsi l'utente che ha fatto l'accesso anche dopo che la sua sessione è scaduta, mentre con duration indichiamo dopo quando deve eliminarsi questo cookie, nel caso nostro dopo 20 minuti.

Per il salvataggio della password invece abbiamo usato la funzione generate_password_hash(), che data una password in input ci consente di salvare un hash della password e non il valore effettivo, poi con la funzione check_password_hash() andiamo invece a verificare che la password inserita da un

utente nel client corrisponda con la password (trasformata in hash) salvata nel database.

- **Change password**

Abbiamo pensato di implementare anche la possibilità di cambiare password nel caso un utente non fosse più sicuro di quella che aveva precedentemente o volesse cambiarla visto che era rimasta la stessa da molto tempo.

- **Visualizzazione grafici risposte chiuse e risultati risposte aperte**

Nell'area riservata, dopo che un utente ha ricevuto delle risposte a questionari che ha creato, può vederne i risultati ottenuti cliccando su uno dei tre tasti disponibili. Nel caso delle risposte a domande aperte visualizzerà per ogni domanda una lista con tutte le risposte che ha ricevuto associate ad essa:

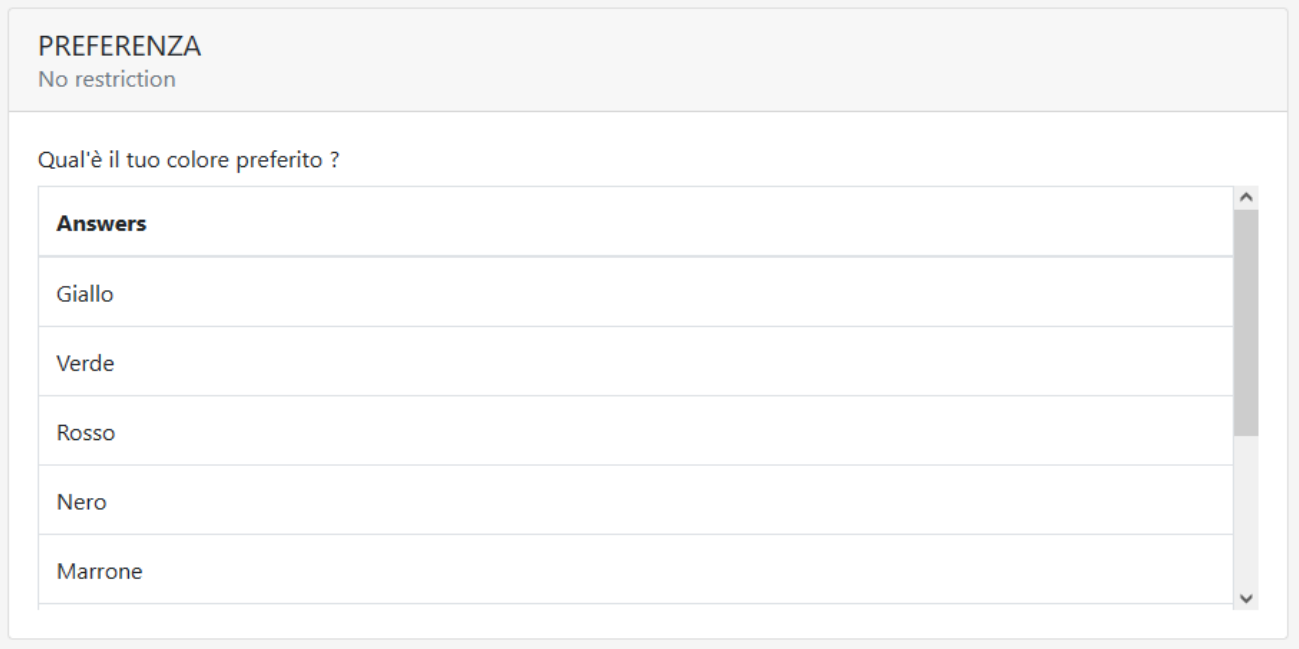

Fig.6 – Area visualizzazione risposte (domanda aperta)

Per quanto riguarda invece le risposte a domande chiuse abbiamo deciso di rappresentare i risultati come un grafico a torta, dove ogni colore rappresenta le diverse risposte date dagli utenti, e allo sfioramento col mouse delle diverse aree è

possibile vedere il numero effettivo di risposte di quel genere che sono state selezionate dagli utenti :

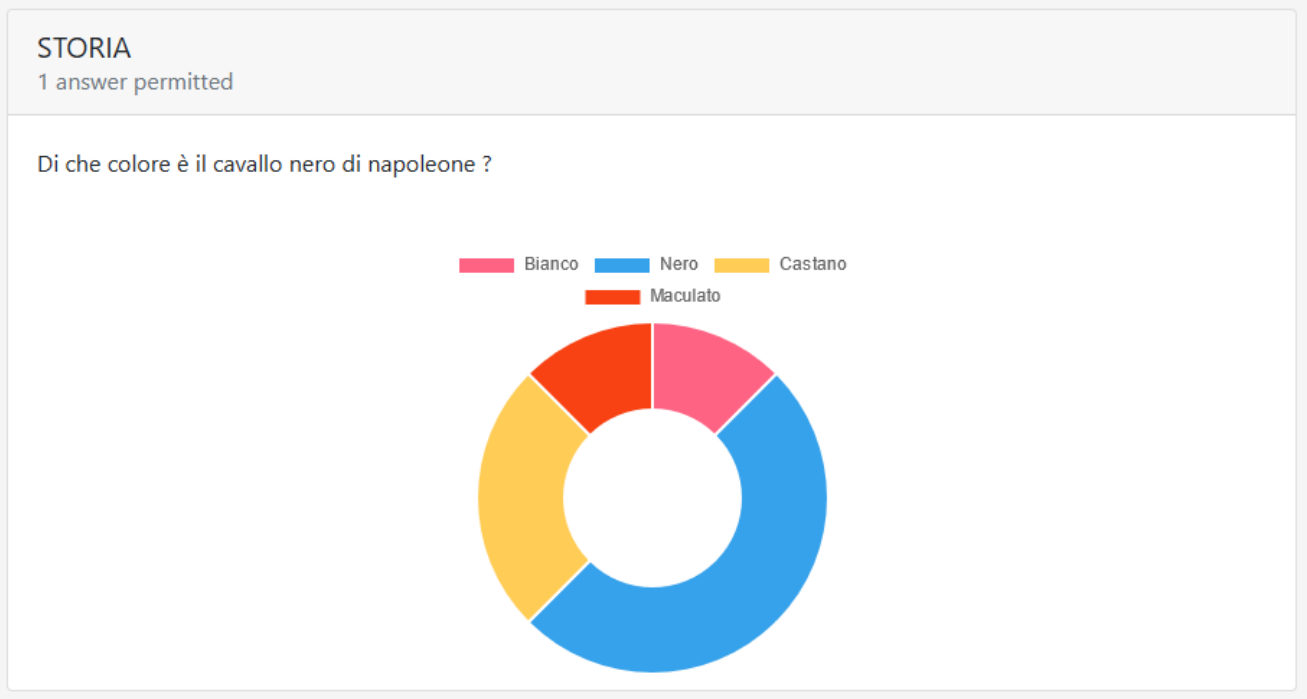

Fig.7 – Area visualizzazione risposte (domanda chiusa)

Siamo riusciti ad ottenere questo risultato creandoci una funzione che prende tutte le risposte ottenute da un questionario e le trasforma in un file JSON, successivamente questo file vien passato al file html ed elaborato con un codice JS, per farlo trasformare, per quanto riguarda le domande a risposta aperta in una lista di risposte (Fig.4), mentre per le domande a risposta chiusa in un grafico a torta (Fig.5)

- **Download risultati in file .csv**

Per dare ad un utente la possibilità di scaricare i proprio risultati sottoforma di file .cvs abbiamo creato una funzione in grado di fornirci tutte le risposte date dagli utenti ad un determinato questionario, e una volta ottenute grazie ad un libreria per phyton chiamata “xlsxwriter” ci convertiamo la risposta data in file .csv, che l’utente possessore del questionario può scaricare dalla sua area riservata, nella sezione dedicata al questionario che desidera scaricare

QUERY PRINCIPALI E TRIGGER:

Nel progetto non abbiamo utilizzato query particolarmente complesse, per lo più si trattano di semplici filter_by() in ORM, abbiamo sfruttato molto spesso invece le diverse relationship che l'ORM ci ha consentito di definire, e che ci hanno evitato di dover fare query molto complesse, sfruttando invece il concetto che ogni riga delle nostre tabelle rappresentano un'istanza nell'ORM. Per quanto riguarda le query "classiche" possiamo solo dire di aver utilizzato molto spesso quelle per recuperarci gli id dei diversi questionari, delle domande o degli utenti, come possiamo vedere nelle due sottostanti :

- query su Survey → `s = Session().query(Survey).get(id)`
- query su User → `User = s.query(User).filter_by(id=current_user.id).first()`

Per quanto riguarda i trigger invece ne abbiamo sviluppati diversi, nella relazione abbiamo di aggiungere i tre più significativi : uno per gestire le RegEx, uno per verificare che una risposta data ad un survey sia associata sempre allo stesso survey e uno per verificare che le risposte inserite in una domanda con risposte chiuse rientrino nel numero fornito dall'utente

Vediamo questi trigger singoli:

1. Nel primo trigger andiamo a controllare che l'espressione passata dalla risposta sia conforme con quanto richiesto dalla domanda, per far questo usiamo un before trigger per riga in modo da controllare tutte le singole righe passate prima di un loro eventuale inserimento nel database

```
@event.listens_for(OpenAnswer.__table__, 'after_create')
def receive_after_create(target, connection, **kw):
    connection.execute(
        """ CREATE OR REPLACE FUNCTION regex()
            RETURNS TRIGGER as $$
            DECLARE
                mandatory boolean;
                regex varchar;
            BEGIN
                mandatory = (SELECT q.mandatory FROM questions_open AS q WHERE q.id = new.open_question_id);
                regex = (SELECT q.regex FROM questions_open AS q WHERE q.id = new.open_question_id);

                IF (NOT mandatory OR regex IS NULL OR new.text ~ CONCAT('^', regex, '$')) THEN
                    RETURN NEW;
                ELSE
                    RETURN OLD;
                END IF;
            END;
            $$ LANGUAGE plpgsql""")

    connection.execute(
        """DROP TRIGGER IF EXISTS regex_trigger ON answers_open;
        CREATE TRIGGER regex_trigger
        BEFORE INSERT OR UPDATE ON answers_open
        FOR EACH ROW
        EXECUTE PROCEDURE regex();"""
    )
```

Fig. 8 – TRIGGER controllo RegEx

2. Nel secondo trigger andiamo a verificare che una risposta data ad una domanda in un questionario specifico sia effettivamente associata a quel questionario e che non sia associata invece ad un questionario differente. Per fare questo tipo di controllo usiamo un before trigger per riga che in caso di insert nella tabella answers va appunto a controllare con una funzione che la risposta sia associata al questionario corretto

```
@event.listens_for(Answer.__table__, 'after_create')
def receive_after_create(target, connection, **kw):
    connection.execute("""
            CREATE OR REPLACE FUNCTION same_survey()
            RETURNS TRIGGER as $$
            BEGIN
            IF EXISTS (
                SELECT *
                FROM answers_closed AS ac
                  INNER JOIN question_closed_option AS qco ON ac.closed_question_option_id = qco.id
                  INNER JOIN question AS q ON q.id = qco.closed_question_id
                WHERE ac.answer_id = NEW.id
                    AND q.survey_id <> NEW.survey_id)

            OR EXISTS (
                SELECT *
                FROM answers_open AS ao
                  INNER JOIN question AS q ON q.id = ao.open_question_id
                WHERE ao.answer_id = NEW.id
                    AND q.survey_id <> NEW.survey_id) THEN

                DELETE FROM answers WHERE id = NEW.id;

            END IF;

              RETURN NULL;
            END;
            $$ LANGUAGE plpgsql""")
```

```
connection.execute("""
        DROP TRIGGER IF EXISTS TrigSameSurvey ON answers;
        CREATE TRIGGER TrigSameSurvey
        AFTER INSERT ON answers
        FOR EACH ROW
        EXECUTE PROCEDURE same_survey()""")
```

Fig. 9 – TRIGGER controllo stesso questionario

3. Nel terzo trigger andiamo a verificare che tutte la somma delle risposte date da un singolo utente a una certa risposta chiusa siano comprese tra il numero minimo e massimo di risposte accettate dal questionario (impostato dal creatore del questionario in fase di creazione delle domande). Per far questo usiamo un after trigger per statement, dove di volta in volta andiamo a verificare tutte le nuove risposte inserite e che nel caso delle risposte a domande chiuse non sia superato il range preimpostato

```
connection.execute("""
CREATE OR REPLACE FUNCTION max_Ans()
       RETURNS TRIGGER as $$
       DECLARE my_cursor refcursor;
       DECLARE idQ integer;
       DECLARE numAns integer;
       BEGIN
              OPEN my_cursor FOR (SELECT DISTINCT qc.id
                                  FROM questions_closed AS qc
                                  INNER JOIN questions AS q ON q.id = qc.id
                                  WHERE q.survey_id = NEW.survey_id);
              FETCH NEXT FROM my_cursor INTO idQ;
              WHILE FOUND LOOP
                   numAns = (SELECT COUNT(*) FROM answers_closed WHERE closed_question_option_id = idQ AND answer_id = NEW.survey_id);

                   IF (numAns > (SELECT q.max_n_of_answer
                              FROM questions_closed AS q
                              WHERE q.id = idQ)
                   OR numAns < (SELECT q.min_n_of_answer
                                FROM questions_closed AS q
                                WHERE q.id = idQ)) THEN

                       CLOSE my_cursor;
                       DELETE FROM answers WHERE id = NEW.id;
                       RETURN NULL;
                   END IF;
                   FETCH NEXT FROM my_cursor INTO idQ;
              END LOOP;
              CLOSE my_cursor;
       END;
       $$ LANGUAGE plpgsql""")
```

```
connection.execute("""
        DROP TRIGGER IF EXISTS TrigMaxMinClosedAnswer ON answers;
        CREATE TRIGGER TrigSameSurvey
        AFTER INSERT ON answers
        FOR EACH ROW
        EXECUTE PROCEDURE max_Ans()""")
```

Fig. 10 – TRIGGER controllo risposte a domanda chiusa nel range impostato