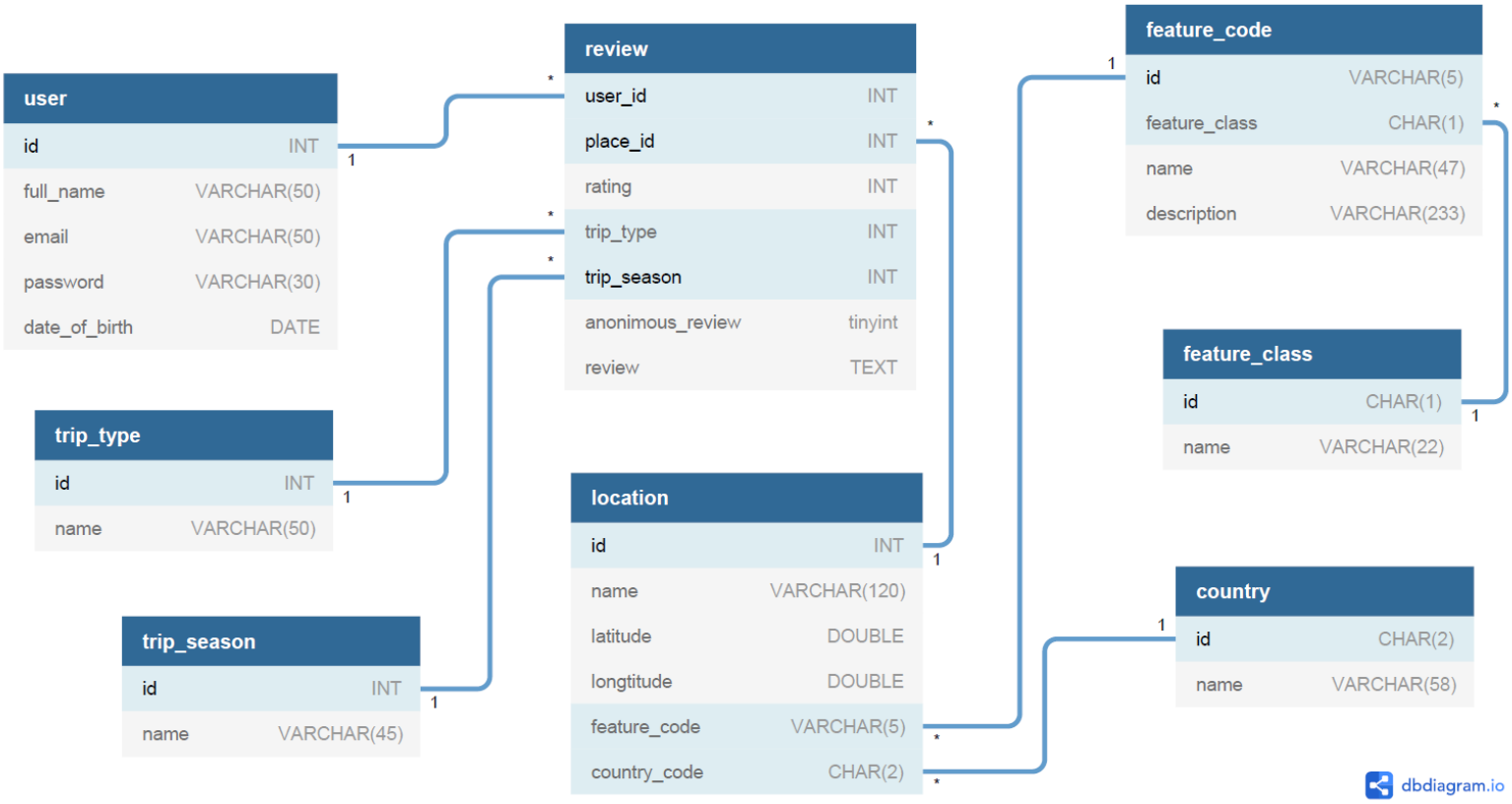
תיעוד תוכנה

סכמה:



מס' הסברים על הסכמה:

- ❖ טבלת location המכילה את המיקומים מהdataset, המדינה שהם נמצאים בה, המיקום הגיאוגרפי שלהם וכו'. lat - latitude הוא DECIMAL(10, 8) ו lng - longtitude הוא DECIMAL(11, 8). ישנו אינדקס עבור כל אחד מהם - רציונאל יוסבר בהמשך.
- ❖ טבלת country - ממפה בין קוד מדינה (לדוגמה FR) לבין שם המדינה (France)
- ❖ טבלת feature_class - ממפה בין feature class כמו A לבין התיאור המלא שלו: country, state, region,... (במילים אחרות, הcategory שמופיע בGUI)
- ❖ טבלת feature_code - ממפה בין feature code כמו PPL לבין התיאור המלא שלו populated place - a city, town, village... (במילים אחרות, הsub-category שמופיע ב GUI)
- ❖ טבלת review: מכילה את הביקורות שרשמו יוזרים על מיקומים. anonymous_review הוא בוליאני שאומר האם הרשומה תופיע ליוזרים אחרים באופן אנונימי

הדאטא:

עשינו שימוש בdataset ששמו geonames אשר מכיל כ11 מיליון רשומות. להלן קישור ל dataset באתר Kaggle. חלק ניכר מהעבודה הושקע בהתאמת הדאטא לסכימה שלנו, ביצירת דאטא משלים ממקורות אחרים ובג'נרות דאטא משלנו. למשל:

- השמטנו עמודות רבות בdataset המקורי אשר לא היו רלוונטיות עבורנו
- את טבלאות feature_class, feature_code, country אבלסנו ע"י דאטא משלים ממקורות אחרים ברשת, שכן הוא לא היה זמין מיידית ביחד עם geonames. למשל את טבלת הcountry אבלסנו ע"י קובץ csv ברשת אשר מכיל את קודי המדינות לפי תקן iso 3166 - התקן שנעשה בו שימוש בgeonames - בצירוף שמות המדינות. פעולות אלה היו לא טריוויאליות ודרשו מניפולציות על קובצי הcsv בטרם ניתן היה לבצע את האבלוס. אתגר נוסף בהקשר זה אשר התמודדנו איתו הוא פערי מידע בין הדאטא שבטבלאות - מטבע הדברים כאשר משתמשים בקבצים שונים ממקורות שונים לצורך האבלוס. איתרנו פערים אלה בזכות הגדרת הforeign keys. באמצעות שאילתה תוך שימוש באופרטור not in מצאנו שהיה מקבץ רשומות בgeonames אשר ה country_code שלהם לא הופיע בטבלת הcountry אותה יצרנו. מבדיקה שערכנו ברשת התברר שהסיבה לכך היא הגדרה לא חד משמעית הנוגעת לשאלה האם בכלל אותן המדינות אכן מוגדרות כמדינות. החלטנו להסיר רשומות אלה מהטבלה שלנו - הרציונאל היה להיצמד לתקן iso 3166
- הקפדנו על הגדרות טיפוסים וגדלים. למשל, בהגדרת name של טבלת country, חישבנו ראשית את האורך המקסימלי max של ערכי עמודה זו בקרב כל הרשומות בטבלה, ובהתאם הגדרנו עמודה זו להיות VARCHAR(max). באופן דומה פעלנו במס' מקומות נוספים.
- בנוסף לדאטא הקיים, על מנת להעניק בשר משמעותי לאפליקציה החלטנו להוסיף מנגנון של יוזרים אשר יכולים לכתוב ביקורות על מקומות שביקרו בהם. לצורך זאת, כתבנו סקריפט אשר מג'נרת כ 40,000 יוזרים וכ 220,000 ביקורות מצד אותם היוזרים
- נציין שכעת הסיסמה של היוזרים שמורה בטבלה כplaintext, אך בכוונתנו להמשיך לעבוד על הפרויקט גם לאחר ההגשה וכמובן כמובן לטפל בהיבט אבטחתי זה ולהצפין את הסיסמות. כרגע מפאת קוצר הזמן השארנו זאת כך, שכן ממילא לא זו מהות הפרויקט, אך בכל אופן אנו מודעים לחשיבות הקריטית של הצפנת סיסמות בעולם האמיתי.

נק' נוספות שנרצה להתייחס אליהן:

- אינדקסים - אוטומטית ישנם אינדקסים על כל הprimary keys + foreign keys. בנוסף, אתגר אשר התמודדנו איתו הוא תמיכה בשאילתה שמחזירה את כל המקומות ברדיוס מסוים סביב מיקום ספציפי אשר מיוצג ב (lat, lng (latitude, longitude אמנם יש נוסחה אשר בהינתן שני מיקומים מחזירה את המרחק ביניהם, אך מימוש נאיבי של השאילתה יגרור מעבר על כל 11 מיליון הרשומות וחישוב הנוסחה עבור כל רשומה - לא מתקבל על הדעת. חקרנו את הנושא וגילינו שישנו spatial index ב mysql אשר מבצע אינדקסינג לטיפוסים גיאוגרפיים ובכלל זה נק' דו-ממדיות. לכן

הוספנו עמודה של coordinates מטיפוס Point. אולם לאחר מכן, גילינו ששימוש בפונקציה מובנית לצורך חישוב המרחק בין שני מיקומים גיאוגרפיים אינו עושה שימוש באינדקס זה, ספציפית בmysql (אפילו שבstack overflow ניתן למצוא לא מעט המלצות להוסיף spatial index בהקשרים אלה בדיוק). החלטנו ללכת על פתרון אחר - להסיר את עמודת הcoordinates, ולהוסיף אינדקס לכל אחד מlat, lng בנפרד. בהינתן רדיוס מסוים, אנחנו מגדירים bounding box (מלבן) צמוד ככל הניתן מסביב למיקום הספציפי אשר מגדיר טווח ערכים אפשריים לlat, lng, וכך מנפים רשומות רבות מאוד תוך שימוש באינדקסים על lat, lng. על מקבץ הרשומות אשר עבר את הסינון, כדי להימנע מfalse positives אנו מבצעים את החישוב הכבד יותר של נוסחת המרחק. מעניין לציין בהקשר לbounding box שתיחום הlat היה קל מבחינת הגדרת הטווח המותר ע"פ הרדיוס, כי ישנה נוסחת המרה ישירה בין מעלות לבין ק"מ. אולם עבור lng זה מעט יותר מסובך, שכן הדבר תלוי גם בlat של המיקום הספציפי, כך שהיינו צריכים לקחת גם זאת בחשבון על מנת לדייק.

- יצרנו בעמוד המרכזי מנגנון pagination שבכל פעם, בעת לחיצה על כפתור, טוען עוד 50 רשומות. מימשנו זאת באמצעות הוספת limit 50 לסוף כל השאילתות, ביצוע order by id ושמירת state של הid האחרון שחזר last_id, כך שבשאילתה מופיע התנאי where id>last_id. מסיבה זו ומשיקולי יעילות (השדה id כמובן מאונדקס) התוצאות ממוינות כאמור לפי id (אחרת, עלול לקרות מצב של השמטת רשימות - כדי להימנע ממצב זה נחוץ שהעמודה לפיה ממיינים תהיה unique מבחינת הערכים שמקבלות הרשומות בעמודה זו)
- השתמשנו בparametrized queries, בעיקר כדי להימנע מהתקפת sql injection הקלאסית

חבילות שהשתמשנו בהן: tkcalendar, mathplotlib, ttkwidgets
יש להריץ את הקובץ main.py על להפעיל את האפליקציה
הקוד מחולק ל3 תיקיות - gui, controller, database. תיעוד מופיע בקוד עצמו.

שאילתות:

בזמן העלייה של האפליקציה מתבצעות מספר שאילתות אשר נועדו להציג מידע שהיוזר רואה במסך הפילטור של המיקומים.
השאילתות הבאות מחזירות את כל המידע ששמור בטבלאות -
country, feature_class, feature_code, trip_season, trip_type.
הטבלאות הללו מכילות מספר רשומות קטן וקבוע.

```
SELECT * FROM country;
SELECT * FROM feature_class;
SELECT * FROM trip_season;
SELECT * FROM trip_type;
```

שתי שאילתות הבאות מיועדות להביא רשימה של feature code מותאמים ל- feature class מסויים. (הרי יש בין טבלאות הללו קשר מסוג - one to many). פונקציה שמכילה שאילתות האלה מקבלת את שם של feature class, אך כדי לבצע שאילתת חיפוש בטבלה - feature_code אנו צריכים להחזיק את id של אותו feature class. לכם קודם אנו מפעילים שאילתה ראשונה.

```
SELECT id FROM feature_class WHERE name = %s;
SELECT * FROM feature_code WHERE feature_class = %s;
```

מספר שאילתות הבאות מיועדות לניהול רשימה של יוזרים, ביצוע פעולות: log in, log out, register.

שאילתה הבא בודקת האם קיים במערכת יוזר עם credentials הנתונים

```
SELECT * FROM user WHERE email = %s AND password = %s
```

שאילתה זו מחזירה מספר יוזרים במערכת שיש להם email מסויים. פרקטית נשתמש בשאילתה בזמן ביצוע register של יוזר חדש.

```
SELECT COUNT(*) FROM user WHERE email = %s
```

שאילתת הכנה יוזר חדש (אחרי ביצוע בדיקות של נכונות).

```
INSERT INTO
        user(full_name, email, password, date_of_birth)
        VALUES(%s, %s, %s, %s);
```

שאילתה הבאה סופרת כמות רשומות עם id's של יוזר, מקום, ושם של עונה. כמו בשאילתה של שליפה feature codes, פה אנו מקבלים שם של עונה ולא id שלה. לכן אנו מבצעים תת-שאילתה.

```
SELECT
        COUNT(*)
FROM
        review
```

```
WHERE
        user_id = %s AND place_id = %s
         AND trip_season = (SELECT id FROM trip_season WHERE name=%s)
```

שאילתת מחיקה של review מסויים של יוזר הנמצא בתוך המערכת.

```
DELETE FROM
        review
WHERE
        user_id = %s AND place_id = %s
        AND trip_season = (SELECT id FROM trip_season WHERE name=%s)
```

שאילתת הוספה של review חדש עבור יוזר הנמצא בתוך המערכת.

```
INSERT INTO
                review (user_id, place_id, rating, trip_type, trip_season, anonymous_review, review
                )
VALUES
                (%s,  %s, %s, (SELECT id FROM trip_type WHERE name=%s),
                (SELECT id FROM trip_season WHERE name=%s))
```

שאילתה הבאה מורכבת מכמב שאילתות מקוננות. מטרה של שאילתה - להחזיר טבלה של תגובות של יוזרים שונים עבור מיקום מסויים, כאשר יחד עם כל תגובה אנו מחזירים שמות trip type, trip season, וגם שם וגיל של יוזר, שכתב את השאילתה.
בשכבה הכי פנימית אנו בוחרים כל התגובות עבור מקום מסוים. את שאילתה הזאת אנו מכניסים לתוך שאילתה אחרת, שמבצעת JOIN בין טבלה שקיבלנו, וטבלה - trip_season.
זה נעשה כדי להחזיר שם של trip season, ולא את id שלו.
אותו דבר נעשה בשכבות הבאות של קוד - אנו מחליפים את id's של trip type ו- user לשמות שלהם, על ידי ביצוע של JOIN, ובחירה עמודות רלוונטיות.

```
first_layer =           SELECT * FROM review WHERE place_id = %s

second_layer =  SELECT
                        l.user_id, l.place_id, l.rating, l.trip_type,
                        r.name as trip_season, l.anonymous_review, l.review
                FROM
                        ({first_layer }) as l INNER JOIN trip_season as r
                ON
                        l.trip_season = r.id


third_layer =   SELECT
                        l.user_id, l.place_id, l.rating, r.name as trip_type,
                        l.trip_season, l.anonymous_review, l.review
                FROM
                        ({second_layer }) as l INNER JOIN trip_type as r
                ON
```

```
                    l.trip_type = r.id

last_layer =        SELECT
                     r.full_name,
                     FLOOR(YEAR(CURRENT_TIMESTAMP) - YEAR(r.date_of_birth)),
                   l.place_id, l.rating, l.trip_type, l.trip_season,
                     l.anonymous_review, l.review
                    FROM
                    ({third_layer }) as l INNER JOIN user as r
                    ON
                    l.user_id = r.id
```

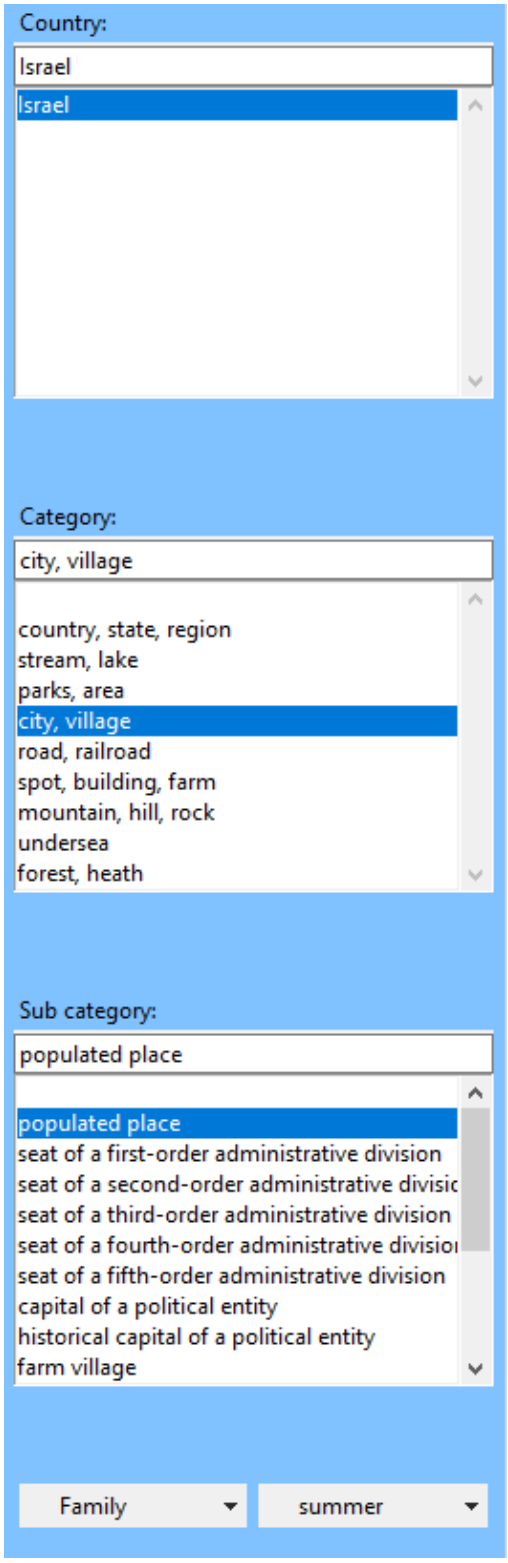
שאילתה הזו די דומה לשאילתה הקודמת, עם מספר הבדלים - פה אנו בוחרים כל תגובות של יוזר מסוים. לכן, בשלב ראשון אנו בוחרים כל תגובות עם user_id מסוים, (בשאילתה קודמת בחרנו לפי place_id). וגם בשלב אחרון (חלק החיצוני של שאילתה) אני מבצעים JOIN עם טבלה - location, כדי להחזיר שמות של מקומות, שיוזר כתב תגובות עליהם.

```
first_layer =       SELECT * FROM review WHERE user_id = %s

second_layer =  SELECT
                    l.user_id, l.place_id, l.rating, l.trip_type,
                    r.name as trip_season, l.anonymous_review, l.review
                    FROM
                    ({first_layer }) as l INNER JOIN trip_season as r
                    ON
                    l.trip_season = r.id


third_layer =   SELECT
                    l.user_id, l.place_id, l.rating, r.name as trip_type,
                    l.trip_season, l.anonymous_review, l.review
                    FROM
                    ({second_layer }) as l INNER JOIN trip_type as r
                    ON
                    l.trip_type = r.id

last_layer =        SELECT
                    r.name as place_name, l.place_id,
                    l.rating, l.trip_type, l.trip_season, l.anonymous_review, l.review
                    FROM
                    ({third_layer }) as l INNER JOIN location as r
                    ON
                    l.place_id = r.id
```

בעמוד הראשי באפליקציה, הפונקציה

```
find_locations(self, country_name, radius, lat, lng, fclass, fcode, trip_type,
trip_season, limit_size, last_id=0)
```

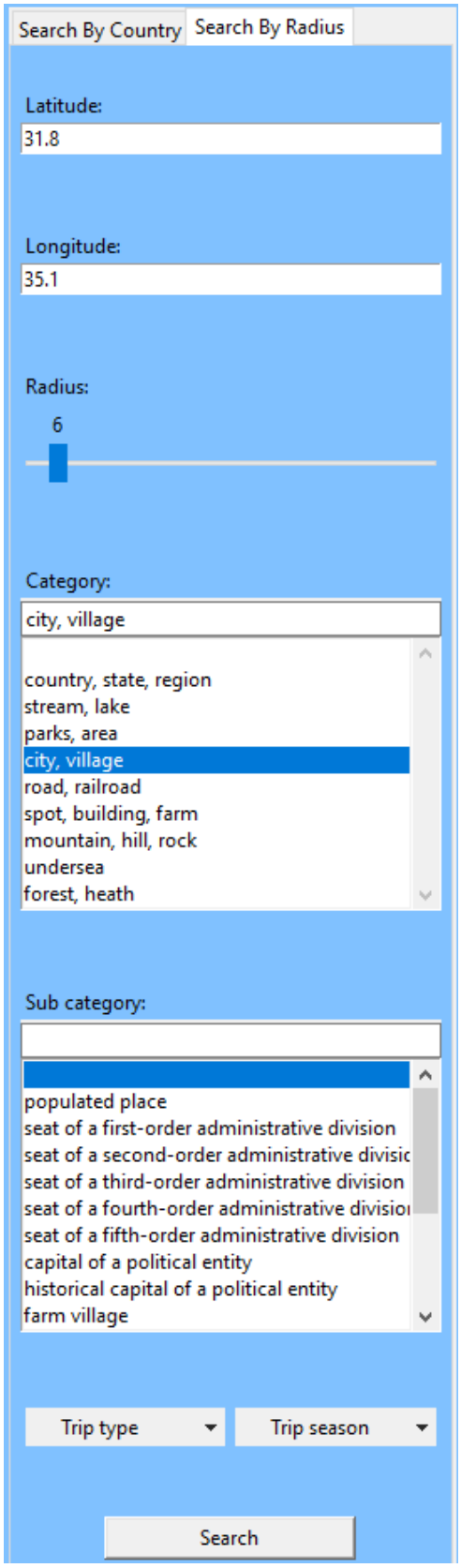
מייצרת באופן דינמי שאילתה כתלות בפרמטרים שנמסרים לה. החלטנו לייצר שאילתה זו באופן דינמי, שכן אחרת בtab של חיפוש לפי מדינה, מס' הקומבינציות של השאילתות הוא 2^4=16
(feature class, feature code, trip type, trip season כולם אופציונליים). לכן כדי להימנע מקוד לא נקי, החלטנו ליצור את השאילתה באופן דינמי בפונקציה זו. להלן השאילתה שמתקבלת כאשר מזינים את כל 4 הפרמטרים האופציונליים במסגרת הסינון:



```sql
SELECT DISTINCT
    l.id,
    l.name,
    lat latitude,
    lng longitude,
    (SELECT
            fclass.name
        FROM
            feature_code fcode
                JOIN
            feature_class fclass ON fcode.feature_class = fclass.id
        WHERE
            fcode.id = l.feature_code) category,
    (SELECT
            fcode.name
        FROM
            feature_code fcode
        WHERE
            fcode.id = l.feature_code) subcategory,
    (SELECT
            c.name
        FROM
            country c
        WHERE
            c.id = l.country_code) country,
    (SELECT
            AVG(rating)
        FROM
            review r
        WHERE
            r.place_id = l.id
                AND r.trip_season = (SELECT
                    id
                FROM
                    trip_season
                WHERE
                    name = 'Summer')
                AND r.trip_type = (SELECT
                    id
                FROM
                    trip_type
                WHERE
                    name = 'Family')) average_rating
FROM
    location l
```

```sql
        JOIN
    country c ON l.country_code = c.id
        JOIN
    review r ON l.id = r.place_id
WHERE
    c.id = (SELECT
            id
        FROM
            country
        WHERE
            name = 'Israel')
        AND l.feature_code = (SELECT
            id
        FROM
            feature_code
        WHERE
            name = 'populated place')
        AND r.trip_season = (SELECT
            id
        FROM
            trip_season
        WHERE
            name = 'Summer')
        AND r.trip_type = (SELECT
            id
        FROM
            trip_type
        WHERE
            name = 'Family')
        AND l.id > 0
ORDER BY l.id
LIMIT 50
;
```

הפונקציה לעיל משמשת גם עבור tab החיפוש לפי רדיוס. למשל, עבור הבחירה:



מקבלים:

```sql
SET @R= 6.0;
SET @lat = '31.8';
SET @lng = '35.1';
SET @earth_radius = 6378;
SET @km_per_lat_degree = @earth_radius * PI() / 180;
SET @lat_delta = @R /@km_per_lat_degree;
SET @lng_delta = @lat_delta / COS(@lat * PI() / 180);
SET @lat_min = @lat - @lat_delta;
SET @lat_max = @lat + @lat_delta;
SET @lng_min = @lng - @lng_delta;
SET @lng_max = @lng + @lng_delta;

SELECT DISTINCT
    l.id,
    l.name,
    lat latitude,
    lng longitude,
    (SELECT
            fclass.name
        FROM
            feature_code fcode
                JOIN
            feature_class fclass ON fcode.feature_class = fclass.id
        WHERE
            fcode.id = l.feature_code) category,
    (SELECT
            fcode.name
        FROM
            feature_code fcode
        WHERE
            fcode.id = l.feature_code) subcategory,
    (SELECT
            c.name
        FROM
            country c
        WHERE
            c.id = l.country_code) country,
    (SELECT
            AVG(rating)
        FROM
            review r
        WHERE
            r.place_id = l.id) average_rating
FROM
    location l
```

```
WHERE
    lat BETWEEN @lat_min AND @lat_max
        AND lng BETWEEN @lng_min AND @lng_max
        AND (((ACOS(SIN(@lat * PI() / 180) * SIN(lat * PI() / 180) + COS(@lat * PI() /
180) * COS(lat * PI() / 180) * COS((@lng - lng) * PI() / 180)) * 180 / PI()) * 60 *
1.1515) * 1.609344) < @R
        AND feature_code IN (SELECT
            fcode.id
        FROM
            feature_code fcode
                JOIN
            feature_class fclass ON fcode.feature_class = fclass.id
        WHERE
            fclass.name = 'city, village')
        AND l.id > 0
ORDER BY l.id
LIMIT 50
;
```

שאילתה שמציגה את 20 המיקומים המדורגים ביותר:

```
SELECT
    l.id,
    l.name,
    lat latitude,
    lng longitude,
    (SELECT
            fclass.name
        FROM
            feature_code fcode
                JOIN
            feature_class fclass ON fcode.feature_class = fclass.id
        WHERE
            fcode.id = l.feature_code) category,
    (SELECT
            fcode.name
        FROM
            feature_code fcode
        WHERE
            fcode.id = l.feature_code) subcategory,
    (SELECT
            c.name
        FROM
            country c
        WHERE
            c.id = l.country_code) country,
```

```
    temp.average_rating average_rating
FROM
    location l
        JOIN
    (SELECT
        place_id, AVG(rating) average_rating
    FROM
        review
    GROUP BY place_id
    ORDER BY average_rating DESC
    LIMIT 20) temp ON l.id = temp.place_id;
```

שאילתה שמציגה לכל זוג של trip type, trip season את ממוצע הגילאים של המשתמשים אשר כתבו ביקורת עם אותם ה trip type, trip season. לכל זוג כזה, כל משתמש נכלל פעם אחת בלבד בעת חישוב הממוצע גם אם כתב יותר מביקורת אחת שעונה על התנאי לעיל

```
SELECT
    trip_season,
    trip_type,
    FLOOR(YEAR(CURRENT_TIMESTAMP) - AVG(year_of_birth)) average_age
FROM
    (SELECT
        YEAR(date_of_birth) year_of_birth,
            ttype.name trip_type,
            tseason.name trip_season
    FROM
        trip_type ttype
    JOIN review r ON ttype.id = r.trip_type
    JOIN trip_season tseason ON tseason.id = r.trip_season
    JOIN user u ON r.user_id = u.id
    GROUP BY u.id , ttype.id , tseason.id) temp
GROUP BY trip_type , trip_season
ORDER BY trip_season , trip_type;
```

שאילתה שמחזירה בהינתן מיקום מסוים את מס' הביקורות שנכתבו עליו עבור כל trip_type. שאילתה זהה ישנה גם עבור trip_season.

```sql
SELECT
    (SELECT
            name
        FROM
            trip_season tseason
        WHERE
            id = trip_season) trip_season,
    COUNT(*) num_reviews
FROM
    review
WHERE
    place_id = 'f'
GROUP BY trip_season;
```