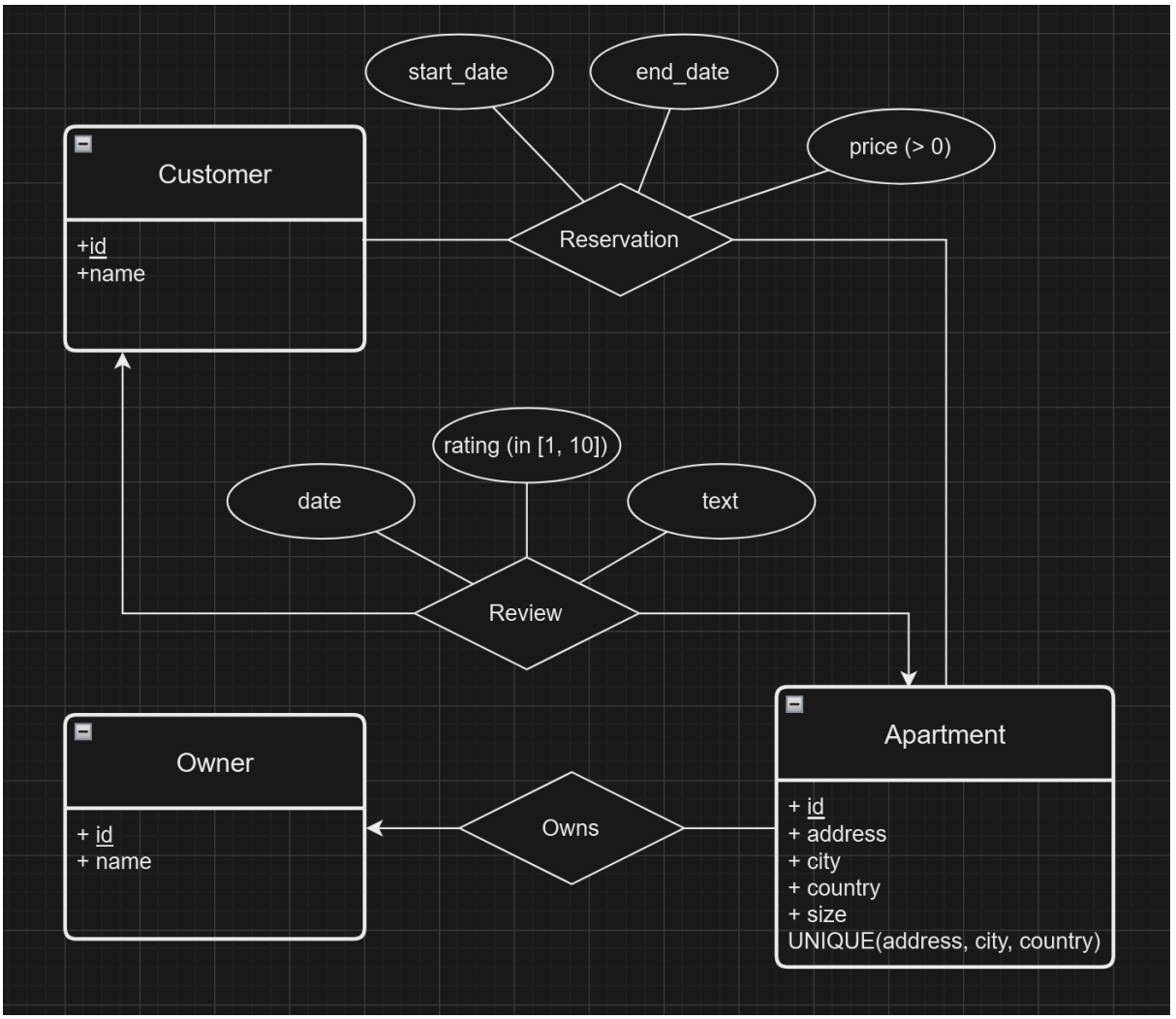
Database Systems 236363, Winter 2023/24

Homework 2

פרטי המגישים:

ת.ז.	
1.	214762718
2.	305764912

אפיון מסד הנתונים:



אנחנו מפתחים מסד נתונים שנקרא "airdbnb", שיאפשר לבעלי דירות להשכיר דירות ללקוחות.

- המערכת תדע לאחסן שמות בעלי דירות, שמות לקוחות ונכסים (דירות), ובנוסף תתמוך בפעולות מחיקה של משתמשים, בעלי דירות ונכסים מהמערכת.
- משתמשים יהיו מסוגלים לבצע הזמנות לדירות (בתאריכים מסוימים) ולהשאיר חוות דעת על הדירה. כמו כן, תתאפשר מחיקה של הזמנה ועדכון של חוות דעת.
- המערכת תתמוך בפונקציות מתקדמות כגון החזרת כל בעלי הדירות בכל עיר שיש בה נכס במערכת, החזרת הנכס בעל הערך הטוב ביותר (value for money), חישוב רווחים חודשיים וקבלת המלצות על דירות.

תכנון ושיקולים:

המערכת שלנו כוללת מספר ישויות ויחסים, בפורמט הבא:

1. ישות המתארת בעל דירה - Owner:

Attribute	Type	Comments	Constraints
id (primary key)	Int (INTEGER)	The owner's ID	ID > 0
name	String (TEXT)	The owner's name	Can't be NULL

- נקפיד שערך המספר המזהה יהיה גדול מאפס, ע"י CONSTRAINT.
- נדאג שהשם של הבעלים לא יהיה ריק ע"י ההגבלה NOT NULL.

```
CREATE TABLE Owner(
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    CONSTRAINT positive_owner_id CHECK (id >= 0)
)
```

2. ישות המתארת לקוח - Customer:

Attribute	Type	Comments	Constraints
id (primary key)	Int (INTEGER)	The customer's ID	ID > 0
name	String (TEXT)	The customer's name	Can't be NULL

- נקפיד שערך המספר המזהה יהיה גדול מאפס, ע"י CONSTRAINT.
- נדאג שהשם של הבעלים לא יהיה ריק ע"י ההגבלה NOT NULL.

```
CREATE TABLE Customer(
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    CONSTRAINT positive_customer_id CHECK (id > 0)
)
```

3. ישות המתארת דירה - Apartment:

Attribute	Type	Comments	Constraints
id (primary key)	Int (INTEGER)	The apartment's ID	ID > 0
address	String (TEXT)	The apartment's address	Can't be NULL
city	String (TEXT)	The apartment's city	Can't be NULL
country	String (TEXT)	The apartment's country	Can't be NULL
size	Int (INTEGER)	The apartment's size	size > 0, Can't be NULL

- נקפיד שערך המספר המזהה יהיה גדול מאפס ושגודל הדירה יהיה גדול מאפס, ע"י CONSTRAINT.
- נדאג שהשם של הבעלים לא יהיה ריק ע"י ההגבלה NOT NULL.
- נדאג שלא יהיו שתי דירות באותה הכתובת ע"י הגדרת סט האטריביוטים (כתובת, עיר, מדינה) כערך ייחודי ע"י שימוש במילת המפתח UNIQUE.

```
CREATE TABLE Apartment(
   id          INTEGER PRIMARY KEY,
   address     TEXT NOT NULL,
   city        TEXT NOT NULL,
   country     TEXT NOT NULL,
   size        INTEGER NOT NULL,
   CONSTRAINT positive_apartment_id CHECK (id > 0),
   CONSTRAINT positive_size CHECK (size > 0),
   CONSTRAINT unique_address UNIQUE (address, city, country)
)
```

4. יחס המתאר בעלות על דירה - Owns:

Attribute	Type	Comments	Constraints
owner_id	Int (INTEGER)	The owner's ID	Refers to an owner
apartment_id	Int (INTEGER)	The apartment's ID	Refers to an apartment. Appears only once.

- האטריביוטים owner_id, apartment_id הם מפתחות זרים (מהישויות Owner ו Apartment). כך נדאג שלא יתווספו טאפלים ליחס שכוללים דירות או בעלים שלא נמצאים במערכת.

- נדאג שמחיקה של בעל דירה או דירה מהמערכת תמחק את הטאפל המתאים ביחס Owns ע"י ההגבלה ON DELETE CASCADE עבור שני האטריביוטים ביחס.
- נגדיר את apartment_id כ-PRIMARY KEY מה שידאג שהוא מופיע רק פעם אחת בטבלה - כך נדאג שאין שני בעלים של אותה הדירה.

```
CREATE TABLE Owns(
   owner_id        INTEGER REFERENCES Owner(id) ON DELETE CASCADE,
   apartment_id    INTEGER REFERENCES Apartment(id) ON DELETE CASCADE,
   PRIMARY KEY(apartment_id)
)
```

5. יחס המתאר הזמנת דירה - Reservation:

Attribute	Type	Comments	Constraints
customer_id	Int (INTEGER)	The customer's ID. FK	Refers to a customer.
apartment_id	Int (INTEGER)	The apartment's ID. FK	Refers to an apartment.
start_date	DATE	reservation start date	Can't be NULL, start_date <= end_date
end_date	DATE	reservation end date	Can't be NULL, start_date <= end_date
price	Int (INTEGER)	reservation price	Can't be NULL, price > 0

- האטריביוטים customer, apartment_id הם מפתחות זרים (מהישויות Customer ו Apartment). כך נדאג שלא יתווספו טאפלים ליחס שכוללים דירות או בעלים שלא נמצאים במערכת.
- נדאג שמחיקה של לקוח או דירה מהמערכת תמחק את הטאפל המתאים ביחס Reservation ע"י ההגבלה ON DELETE CASCADE עבור שני האטריביוטים של המספרים המזהים ביחס.
- נדרג על ידי CHECK שתאריך הסוף בא אחרי תאריך ההתחלה של ההזמנה

```
CREATE TABLE Reservation(
   customer_id     INTEGER REFERENCES Customer(id) ON DELETE CASCADE,
   apartment_id    INTEGER REFERENCES Apartment(id) ON DELETE CASCADE,
   start_date      DATE NOT NULL,
   end_date        DATE NOT NULL,
   price           INTEGER NOT NULL,
   CONSTRAINT positive_price CHECK (price > 0),
   CONSTRAINT legal_dates CHECK (start_date <= end_date)
)
```

6. יחס המתאר ביקורת שלקוח השאיר על דירה - Review:

Attribute	Type	Comments	Constraints
customer_id	Int (INTEGER)	The customer's ID.	FK
apartment_id	Int (INTEGER)	The apartment's ID.	FK
date	DATE	review date	Can't be NULL, ללקוח חייבת להיות הזמנה לדירה שתאריך הסיום שלה מוקדם יותר מתאריך השארת הביקורת
rating	Int (INTEGER)	the rating of the review	Can't be NULL, 1<=rating<=10
review_text	String (TEXT)	review text	Can't be NULL

- האטריביוטים customer_id, apartment_id הם מפתחות זרים (מהישויות Customer ו Apartment). כך נדאג שלא יתווספו טאפלים ליחס שכוללים דירות או לקוחות שלא נמצאים במערכת.
- נדאג שמחיקה של לקוח או דירה מהמערכת תמחק את הטאפל המתאים ביחס Review ע"י ההגבלה ON DELETE CASCADE עבור שני האטריביוטים של המספרים המזהים ביחס.
- בדיקות נוספות יתבצעו ע"י CONSTRAINTS עבור הנכונות של ערכי הדירוג.
- זוג האטריביוטים customer_id, apartment_id יוגדרו כ PRIMARY KEY, מכיוון שלקוח יכול להשאיר סקירה על דירה רק פעם אחת.

```
CREATE TABLE Review(
   customer_id     INTEGER REFERENCES Customer(id) ON DELETE CASCADE,
   apartment_id    INTEGER REFERENCES Apartment(id) ON DELETE CASCADE,
   date            DATE NOT NULL,
   rating          INTEGER NOT NULL,
   review_text     TEXT NOT NULL,
   CONSTRAINT legal_rating CHECK (rating >= 1 and rating <= 10),
   PRIMARY KEY (customer_id, apartment_id)
)
```

מבטים (VIEWS):

בנוסף, נשתמש במבטים הבאים:

מבט OAP:

```
CREATE VIEW OAP AS
    SELECT A.owner_id, B.name, A.apartment_id, C.address, C.city,
C.country, C.size
    FROM Owns A, Owner B, Apartment C
    WHERE A.owner_id = B.id AND A.apartment_id = C.id
```

מטרת המבט הזה היא לחסוך שכפול קוד עתידי לפונקציות שצריכות להשתמש בבעלי דירה ודירות. המבט יתאר טבלה שבה נשמור את המספר המזהה של בעל נכס, השם שלו, המספר המזהה של הנכס, הכתובת המלאה שלו והגודל שלו.

מבט AverageApartmentRating:

```
   CREATE VIEW AverageApartmentRating AS
   SELECT apartment_id, average_rating
   FROM (
       SELECT apartment_id, AVG(rating) AS average_rating FROM Review
       GROUP BY apartment_id
       UNION ALL
       SELECT id AS apartment_id, 0 FROM Apartment
       WHERE id NOT IN (SELECT apartment_id FROM Review)
   )
```

המבט מחזיר את כל המזהים של דירות מותאמים לממוצע הדירוגים שלהם. הוא עושה זאת בכך שהוא לוקח את הממוצע של הדירוגים שלו מתוך Review ומוסיף גם את המזהים של דירות שלא דורגו כלל יחד עם ממוצע 0.

מבט CustomerRatio:

```
CREATE VIEW CustomerRatio AS
SELECT R1.customer_id AS id1,
      R2.customer_id AS id2,
      AVG(R1.rating / R2.rating::DECIMAL) AS ratio
FROM Review AS R1
JOIN Review AS R2 ON R1.apartment_id = R2.apartment_id
WHERE R1.customer_id != R2.customer_id
GROUP BY R1.customer_id, R2.customer_id
```

לכל שני לקוחות שונים זה מזה, שהתארחו באותן דירות, המבט יחשב את היחס בין הדירוגים שלהם, או ממוצע היחסים או מדובר במספר דירות שהם התארחו בהן. הדבר יתבצע ע"י JOIN של REVIEW פעמיים,

חישוב ממוצע הדירוגים (ניעזר ב GROUP BY לשם חישוב הממוצע עבור כל שני זוגות של לקוחות ששונים זה מזה). את היחס בין הדירוגים נכניס לעמודה ratio. התוצאה שמתקבלת היא טבלה בעלת העמודות הבאות: id_1, id_2, ratio. (השימוש ב-DECIMAL מאפשר להשתמש במספרים עם יותר ספרות אחרי הנקודה ומקטין את השגיאה שהייתה משמעותית בבדיקות שלנו).

<u>פונקציות המערכת:</u>

הערה: בכל הפונקציות, כאשר נדרשת בדיקה ספציפית לטיב המשתנים שהפונקציה מקבלת (בדיקה האם הערכים חוקיים), והבדיקה אינה יכולה להתבצע במסגרת שאילתה, עקב דרישה להחזרת שגיאה ספציפית, הבדיקה תתבצע בפייתון, (או דרך תוספת של Constraint לטבלה). זה נוגע להחזרת ערך BAD_PARAMS בלבד, כפי שתואר בפיאצה (וספציפית למקרים בהם נדרש שיוחזר BAD_PARAMS לפני FOREIGN_KEY_VOILATION). בכל שאר המקרים, כאשר ערך ההחזרה של הפונקציה הוא מהטיפוס RetValue, טיפוסו ייקבע לפי שגיאות שנזרקות ממסד הנתונים בעת הרצת השאילתה, כתלות במפתחות, בדיקות CHECK, UNIQUE, הגבלות נוספות וערכי מפתחות. כל המגבלות הנ"ל פורטו כאשר תיארנו את מסד הנתונים.

<u>א. CRUD API:</u>

1. **void createTables()**

פונקציה זו יוצרת את הטבלאות וה Views המתוארים בסעיף התכנון והשיקולים של מסד הנתונים.

2. **void clearTables()**

פונקציה זו מנקה את הטבלאות וה Views המתוארים בסעיף התכנון והשיקולים של מסד הנתונים. (ז"א - משאירה אותם קיימים, אך מסירה מהם את המידע שנשמר בהם)

3. **void dropTables()**

פונקציה זו מוחקת את הטבלאות וה Views (בעזרת Casdade) המתוארים בסעיף התכנון והשיקולים של מסד הנתונים.

4. **ReturnValue add_owner(owner: Owner)**

פונקציה זו מוסיפה בעל דירה חדש לטבלת Owner. הקלט הוא אובייקט מסוג Owner כפי שהוגדר בתרגיל, והפלט הוא ערך החזרה כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- ALREADY_EXISTS if a owner with the same ID already exists
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("INSERT INTO Owner(id, name) VALUES({ownerid},
{ownername})").format(
   ownerid=sql.Literal(owner.get_owner_id()),
   ownername=sql.Literal(owner.get_owner_name()))
```

5. **Owner get_owner(owner_id: int)**

פונקציה זו מחזירה אובייקט Owner מתוך המערכת, לפי מספר מזהה שניתן לה. הקלט הוא מספר מזהה, והפלט הוא אובייקט מסוג Owner כפי שהוגדר בתרגיל. אם המספר מייצג בעלים שאינו קיים במערכת (נגלה זאת ע"י בדיקת השורות עליהן השפיעה השאילתה), נחזיר bad_owner.

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("SELECT name FROM owner WHERE id =
{ownerid}").format(ownerid=sql.Literal(owner_id))
```

6. **ReturnValue delete_owner(owner_id: int)**

פונקציה זו מוחקת אובייקט Owner מתוך המערכת, לפי מספר מזהה שניתן לה. הקלט הוא מספר מזהה, והפלט הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if the owner does not exist
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("DELETE FROM owner WHERE id =
{ownerid}").format(ownerid=sql.Literal(owner_id))
```

7. **ReturnValue add_apartment(apartment: Apartment)**

פונקציה זו מוסיפה דירה חדש לטבלת Apartment. הקלט הוא אובייקט מסוג Apartment כפי שהוגדר בתרגיל, והפלט הוא ערך החזרה כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- ALREADY_EXISTS if an apartment in the same location (address, city and country) already exists or there is an apartment with the same id.
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("INSERT INTO Apartment(id, address, city, country, size)
VALUES({apartmentid}, "
               "{apartmentaddress}, {apartmentcity}, {apartmentcountry},
{apartmentsize})").format(
   apartmentid=sql.Literal(apartment.get_id()),
   apartmentaddress=sql.Literal(apartment.get_address()),
   apartmentcity=sql.Literal(apartment.get_city()),
   apartmentcountry=sql.Literal(apartment.get_country()),
   apartmentsize=sql.Literal(apartment.get_size()))
```

8. **Apartment get_apartment(apartment_id: int)**

פונקציה זו מחזירה אובייקט מסוג Apartment, כפי שמוגדר בתרגיל, לפי מספר מזהה (הקלט).
במקרה של שגיאה יוחזר bad_apartment.

הפונקציה תשתמש בשאילתה:

```
query = sql.SQL("SELECT id, address, city, country, size FROM Apartment WHERE id
= {apartmentid}").format(
   apartmentid=sql.Literal(apartment_id))
```

9. **ReturnValue delete_apartment(apartment_id: int)**

פונקציה זו מוחקת אובייקט Apartment מתוך המערכת, לפי מספר מזהה שניתן לה. הקלט הוא מספר מזהה, והפלט הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if the apartment does not exist
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("DELETE FROM Apartment WHERE id =
{apartmentid}").format(apartmentid=sql.Literal(apartment_id))
```

10. **ReturnValue add_customer(customer: Customer)**

פונקציה זו מוסיפה לקוח חדש לטבלת Customer. הקלט הוא אובייקט מסוג Customer כפי שהוגדר בתרגיל, והפלט הוא ערך החזרה כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- ALREADY_EXISTS if a customer with the same ID already exists
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("INSERT INTO Customer(id, name) VALUES({customerid},
{customername})").format(
   customerid=sql.Literal(customer.get_customer_id()),
   customername=sql.Literal(customer.get_customer_name()))
```

11. Customer get_customer(customer_id: int)

פונקציה זו מחזירה אובייקט מסוג Customer, כפי שמוגדר בתרגיל, לפי מספר מזהה (הקלט). במקרה של שגיאה יוחזר bad_customer.

הפונקציה תשתמש בשאילתה:

```
query = sql.SQL("SELECT name FROM customer WHERE id =
{customerid}").format(customerid=sql.Literal(customer_id))
```

12. ReturnValue delete_customer(customer_id: int)

פונקציה זו מוחקת אובייקט Customer מתוך המערכת, לפי מספר מזהה שניתן לה. הקלט הוא מספר מזהה, והפלט הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if the customer does not exist
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("DELETE FROM Customer WHERE id =
{customerid}").format(customerid=sql.Literal(customer_id))
```

13. ReturnValue customer_made_reservation(customer_id: int, apartment_id: int, start_date: date, end_date: date, total_price: float)

פונקציה זו מכניסה לטבלה Reservstion טאפל של מזהה לקוח, מזהה דירה, תאריך התחלה תאריך סיום ומחיר. מטרתה להוסיף הזמנה של לקוח לדירה במערכת. נציין כי המחיר של ההזמנה חייב להיות ערך חיובי, וכי תאריך סיום ההזמנה חייב להיות מאוחר יותר מתאריך ההתחלה של ההזמנה. בנוסף, על הדירה להיות פנויה בתאריכים המבוקשים. הפונקציה תזהה זאת ע"י תת שאילתה בתוך WHERE של השאילתה שלנו, שיגביל את ההכנסות רק לכאלו חוקיות. על מנת לבדוק האם הייתה הכנסה נשתמש ב-rows_affected שחוזר מ-execute.

פלט הפונקציה הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal or the apartment isn't available at the specified date (there is already a reservation for it)
- NOT_EXISTS if the customer or the apartment don't exist
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
       INSERT INTO Reservation(customer_id, apartment_id, start_date, end_date,
price)
       SELECT {customerid}, {apartmentid}, {startdate}, {enddate},
{apartmentprice}
       WHERE NOT EXISTS(
           SELECT 1 FROM Reservation AS selection
           WHERE selection.apartment_id = {apartmentid}
           AND (selection.start_date, selection.end_date) OVERLAPS ({startdate},
{enddate})
       )
   """).format(
   customerid=sql.Literal(customer_id),
   apartmentid=sql.Literal(apartment_id),
   startdate=sql.Literal(start_date),
   enddate=sql.Literal(end_date),
   apartmentprice=sql.Literal(total_price)
)
```

14. **ReturnValue customer_cancelled_reservation(customer_id: int, apartment_id: int, start_date: date)**

פונקציה זו מוחקת מטבלה Reservstion טאפל של מזהה לקוח, מזהה דירה, תאריך התחלה תאריך סיום ומחיר. מטרתה להסיר הזמנה של לקוח לדירה במערכת. הפונקציה מקבלת מזהה לקוח, מזהה דירה ואת תאריך ההתחלה של ההזמנה.

פלט הפונקציה הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if the customer or the apartment don't exist or there isn't a reservation at the given apartment on the given date
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("DELETE FROM Reservation WHERE customer_id = {customerid} "
                "AND apartment_id = {apartmentid} "
                "AND start_date = {startdate}  ").format(
   customerid=sql.Literal(customer_id),
   apartmentid=sql.Literal(apartment_id),
   startdate=sql.Literal(start_date),
)
```

15. **ReturnValue customer_reviewed_apartment(customer_id: int , apartment_id: int, review_date: date, rating: int, review_text: str)**

פונקציה זו מכניסה לטבלה Review טאפל של מזהה לקוח, מזהה דירה, תאריך כתובת ביקורת, דירוג (מ 1 עד 10) ותיאור (טקסט). מטרתה להוסיף ביקורת של לקוח על דירה ששהה בה. יש לציין כי לקוח יכול להשאיר ביקורת על דירה פעם אחת בלבד, ומערכת מסד הנתונים אוכפת זאת ע"י הגדרת צמד המזהים (לקוח ודירה) כ PRIMARY KEY ולכן לא יכולים להופיע פעמיים יחדיו. בנוסף, לקוח יכול להשאיר חוות דעת על דירה רק לאחר שסיים להתארח בה, וגם זה נאכף במסגרת מסד הנתונים, במסגרת השאילתה.

פלט הפונקציה הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if the customer or the apartment don't exist or the customer doesn't have a reservation as described above
- ALREADY_EXISTS if the customer already reviewed the apartment
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
       INSERT INTO Review(customer_id, apartment_id, date, rating, review_text)
       SELECT {customerid}, {apartmentid}, {reviewdate}, {rating}, {reviewtext}
       WHERE EXISTS(
           SELECT 1 FROM Reservation AS selection
           WHERE selection.apartment_id = {apartmentid}
           AND selection.customer_id = {customerid}
           AND selection.end_date <= {reviewdate}
       )
   """).format(
   customerid=sql.Literal(customer_id),
   apartmentid=sql.Literal(apartment_id),
   reviewdate=sql.Literal(review_date),
   rating=sql.Literal(rating),
   reviewtext=sql.Literal(review_text)
)
```

16. ReturnValue customer_updated_review(customer_id: int, apartment_id:int, update_date: date, new_rating: int, new_text: str)

פונקציה זו מעדכנת ערכים בטבלה Review. הערכים שיעודכנו הם הדירוג, תוכן הביקורת (טקסט) והתאריך שלה. פונקציה זו מאפשרת ללקוחות לעדכן את חוות הדעת שלהם על הדירה. הדרישות לגבי תאריך השארת הביקורת זהות ל customer_reviewed_apartment.

פלט הפונקציה הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if there is no review of the given apartment by the given customer or a review was given after update_date
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
   UPDATE Review
   SET date = {reviewdate}, rating = {reviewrating}, review_text = {reviewtext}
   WHERE(
       customer_id = {customerid}
       AND apartment_id = {apartmentid}
       AND EXISTS(
           SELECT 1 FROM Review AS selection
           WHERE selection.apartment_id = {apartmentid}
           AND selection.customer_id = {customerid}
           AND selection.date <= {reviewdate}
       )
   )
""").format(
```

```
    customerid=sql.Literal(customer_id),
    apartmentid=sql.Literal(apartment_id),
    reviewdate=sql.Literal(update_date),
    reviewrating=sql.Literal(new_rating),
    reviewtext=sql.Literal(new_text)
)
```

17. ReturnValue owner_owns_apartment(owner_id: int, apartment_id: int)

פונקציה זו מוסיפה טאפל של מספר מזהה של בעלים ומספר מזהה של דירה, לטבלה Owns.
מטרתה להוסיף בעלי דירה למערכת. נציין כי לדירה יכול להיות בעל דירה אחד לכל היותר, והגבלה זו נאכפת במסגרת מסד הנתונים (ערך המזהה של הדירה הוא מפתח).

פלט הפונקציה הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if the owner or the apartment don’t exist
- ALREADY_EXISTS if the apartment already has an owner
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("INSERT INTO Owns(owner_id, apartment_id) VALUES({ownerid},
{apartmentid})").format(
    ownerid=sql.Literal(owner_id),
    apartmentid=sql.Literal(apartment_id))
```

18. ReturnValue owner_drops_apartment(owner_id: int, apartment_id: int)

פונקציה זו מסירה טאפל של מספר מזהה של בעלים ומספר מזהה של דירה, מהטבלה Owns.
מטרתה להסיר בעלי דירה למערכת.

פלט הפונקציה הוא ReturnValue כפי שהוגדר בתרגיל, באופן הבא:

- OK in case of success
- BAD_PARAMS if any of the params are illegal
- NOT_EXISTS if the owner or the apartment don’t exist or the owner doesn’t own the apartment
- ERROR in case of database error

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("DELETE FROM Owns "
                "WHERE owner_id = {ownerid} "
                "AND apartment_id = {apartmentid}").format(
                ownerid=sql.Literal(owner_id),
                apartmentid=sql.Literal(apartment_id))
```

19. Owner get_apartment_owner(apartment_id: int)

פונקציה זו מחזירה אובייקט מסוג Owner כפי שהוגדר בתרגיל, לפי מספר מזהה של דירה, מהטבלה Owner. מטרתה להחזיר את הבעלים של דירה מסוימת לפי המספר המזהה שלה.

פלט הפונקציה הוא אובייקט מסוג Owner אם קיים בעל דירה, ו bar_owner אחרת.

הפונקציה תשתמש בשאילתה הבאה (נעזרת במבט OAP):

```
query = sql.SQL("SELECT owner_id, name FROM OAP WHERE apartment_id =
{apartmentid}").format(
apartmentid=sql.Literal(apartment_id))
```

20. List[Apartment] get_owner_apartments(owner_id: int)

פונקציה זו מחזירה אובייקט מסוג רשימה של Apartment כפי שהוגדר בתרגיל, לפי מספר מזהה של בעלי דירה. מטרתה להחזיר את כל הדירות של בעל דירה מסוים, לפי המספר המזהה שלו.

פלט הפונקציה הוא הרשימה הנ"ל אם קיימות דירות, ואחרת רשימה ריקה (בנוסף, תתקהל רשימה ריקה בכל שגיאה שניתקל בה במהלך הקריאה לפונקציה).

הפונקציה תשתמש בשאילתה הבאה (נעזרת במבט OAP):

```
query = sql.SQL("SELECT owner_id, apartment_id, address, city, country, size
FROM OAP WHERE owner_id = {ownerid}").format(
   ownerid=sql.Literal(owner_id))
```

ב. BASIC API:

1. **float get_apartment_rating(apartment_id: int)**

פונקציה זו מחזירה את ממוצע הדירוגים של נכס מסוים. הקלט של הפונקציה הוא המספר המזהה של הנכס, והפלט הוא ממוצע הדירוגים.

הפונקציה תשתמש בשאילתה הבאה (נעזרת במבט AverageApartmentRating):

```
query = sql.SQL("""
   SELECT average_rating
   FROM AverageApartmentRating
   WHERE apartment_id = {apartment_id}
""").format(apartment_id=sql.Literal(apartment_id))
```

2. **float get_owner_rating(owner_id: int)**

פונקציה זו מחזירה את הממוצע של ממוצע הדירוגים של כל הנכסים בבעלותו של בעלים מסוים. הקלט של הפונקציה הוא המספר המזהה של הבעלים, והפלט הוא ממוצע הדירוגים.

הפונקציה תשתמש בשאילתה הבאה (נעזרת במבט AverageApartmentRating, ובפונקציה COALESCE בשביל לטפל בבעלים שאין להם דירות):

```
query = sql.SQL("""
   SELECT COALESCE(AVG(average_rating), 0) AS average_rating
   FROM AverageApartmentRating, Owns
   WHERE owner_id = {owner_id}
   AND AverageApartmentRating.apartment_id = Owns.apartment_id
""").format(owner_id=sql.Literal(owner_id))
```

3. Customer get_top_customer()

פונקציה זו מחזירה אובייקט מסוג Customer שמייצג את הלקוח שביצע את מספר ההזמנות הגדול ביותר. אם יש יותר מלקוח אחד שביצע את מספר ההזמנות הגבוה ביותר, יוחזר זה עם המספר המזהה המינימאלי.

השאילתה עובדת בעזרת תת-שאילתה שמתאימה לכל לקוח את מספר ההזמנות שלו, או 0 אם לא הזמין כלל.

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
   SELECT customer_id, name
   FROM Customer, (
       SELECT customer_id, COUNT(*) AS reservations
       FROM Reservation
       GROUP BY customer_id
       UNION ALL
       SELECT id AS customer_id, 0
       FROM Customer
       WHERE id NOT IN (SELECT customer_id FROM Reservation)
   ) as T
   WHERE Customer.id = T.customer_id
   ORDER BY reservations DESC, customer_id ASC
   LIMIT 1
""").format()
```

4. List[Tuple[str, int]] reservations_per_owner()

פונקציה זו מחזירה רשימה של טאפלים בפורמט הבא: (מספר הזמנות כולל, מספר מזהה של בעל נכס). רשימה זו מייצגת את מספר ההזמנות שנעשו לנכסיו של כל בעל נכס במערכת.

השאילתה סופרת לכל בעלים של דירה את כמות ההזמנות שמשוייכות לדירות בשמו, ולאלו שאין בידיהם דירה, היא מוסיפה רשומה שלהם משוייכים ל-0

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
   SELECT name, COUNT(*) AS reservations
   FROM Owner, Owns, Reservation
   WHERE Owner.id = Owns.owner_id
   AND Owns.apartment_id = Reservation.apartment_id
   GROUP BY name

   UNION ALL

   SELECT name, 0
   FROM Owner
   WHERE NOT EXISTS (
       SELECT *
       FROM Owns, Reservation
       WHERE Owner.id = Owns.owner_id
       AND Owns.apartment_id = Reservation.apartment_id
   )
""").format()
```

ג. Advanced API:

5. List[Owner] get_all_location_owners()

פונקציה זו מחזירה את הבעלים שלהם יש דירות בכל עיר שיש במסד. זה שקול לרשימת הבעלים שאין עיר שלא קיימת לבעלים דירה בה. נבדוק זאת באמצעות WHERE במעבר על Owner שיעבור על כל הדירות על מנת לבדוק שבכל עיר יש לבעלים דירה בעיר הזאת. לשם מימוש הפונקציה ניעזר במבט OAP, שמאגד בעלי דירות ואת הדירות שלהם לטבלה אחת.

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
   SELECT DISTINCT id, name
   FROM Owner
   WHERE NOT EXISTS (
       SELECT * FROM Apartment as A
       WHERE NOT EXISTS (
           SELECT * FROM OAP
           WHERE OAP.city = A.city
           AND OAP.country = A.country
           AND OAP.owner_id = Owner.id
       )
   )
""").format()
```

6. Apartment best_value_for_money()

נשתמש ב-AverageApartmentRating ובתת שאילתה דומה שיצמיד למזהי דירות את מחיר ההזמנה הממוצע שלהם. נקרא מתת השאילתות ומ-Apartment וניצור טבלה שמסדרת את הדירות לפי היחס בין הדירוג הממוצע למחיר הממוצע. ניקח רק את הדירה הראשונה.

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
   SELECT Apartment.id AS id, address, city, country, size
   FROM Apartment, AverageApartmentRating, (
       SELECT
           apartment_id,
           COALESCE(AVG(price / (end_date - start_date)), 0) AS avg_night_price
       FROM Reservation
       GROUP BY apartment_id
   ) AS T
   WHERE Apartment.id = AverageApartmentRating.apartment_id
   AND Apartment.id = T.apartment_id
   ORDER BY average_rating / avg_night_price DESC
   LIMIT 1
""").format()
_, result = conn.execute(query)
```

7. **List[Tuple[int, float]] profit_per_month(year: int)**

נעבור על כל ההזמנות שיש במסד, ולכל אחת מהן נצמיד לחודש של תאריך הסיום, את המחיר של ההזמנה כפול 15%. נשתמש ב-WHERE על מנת להסתכל רק על הזמנות שהסתיימו בשנה הרצויה. נעשה ממוצע על כל חודש. נוסיף לתוצאה הסופית טבלה שתתאים לכל חודש את המספר 0 אם בחודש הזה לא נגמרה אף הזמנה מהשנה.

```
query = sql.SQL("""
   (
       SELECT EXTRACT(MONTH FROM end_date) AS month, 0.15 * SUM(price) AS profit
       FROM Reservation
       WHERE EXTRACT(YEAR FROM end_date) = {year}
       GROUP BY month
       UNION ALL
       SELECT month, 0
       FROM generate_series(1, 12) AS month
       WHERE month NOT IN (
           SELECT EXTRACT(MONTH FROM end_date) FROM Reservation
           WHERE EXTRACT(YEAR FROM end_date) = {year}
       )
   ) ORDER BY month
""").format(year=sql.Literal(year))
```

8. **List[Tuple[Apartment, float]] get_apartment_recommendation(customer_id: int)**

נצמיד לכל דירה, את הממוצע של ההערכה עבור הלקוח שלנו. נעשה זאת בכך שלכל דירה, ולכל דירוג של הדירה, שלא על ידי הלקוח שלנו, ההערכה שלנו תהיה המכפלה של "היחס" בין הלקוח שדירג והלקוח שלנו, בדירוג של הלקח האחר. את היחס ניקח מתוך CustomerRatio. ניקח את הממוצע של הקירובים האלו באמצעות AVG (ונחסום אותם בין 1 ל-10).

הפונקציה תשתמש בשאילתה הבאה:

```
query = sql.SQL("""
   SELECT A.id AS apartment_id, address, city, country, size,
          AVG(LEAST(GREATEST(ratio * R.rating, 1), 10)) AS approx
   FROM Apartment as A, CustomerRatio, Review as R
   WHERE A.id = R.apartment_id
   AND CustomerRatio.id1 = {customer_id}
   AND CustomerRatio.id2 = R.customer_id
   AND A.id NOT IN (
       SELECT apartment_id
       FROM Reservation
       WHERE customer_id = {customer_id}
   )
   GROUP BY A.id, address, city, country, size
""").format(customer_id=sql.Literal(customer_id))
```