הסבר כללי על מבנה הנתונים + יצירת הטבלאות:

בחרנו לממש את המסד נתונים ככזה המחזיק שלוש ישויות ראשיות:

Customers,Apartments,Owners

כאשר הקשרים ביניהם יתבסס על הטבלאות:

Owns: המקשר בין בעל דירה לדירתו.

Reviews: המקשר בין דירוג לקוח לבין דירה בה שהה.

Reserves: המקשר בין הזמנת לקוח לבין דירה.

המטרה העיקרית שלנו הייתה לפשט את הטבלאות ככל שניתן ולא לשמור מידע שלא רלוונטי לקשר מסוים בתוך הטבלה (למשל Owns **לא** יכיל שדה Owner name אלא רק שדה Owner_id).

בנוסף, המפתחות העיקריים בטבלאות הישויות הם המפתחות הזרים בשאר הטבלאות, וכל מחיקת ישות מתאפשרת בקלות מרבית.

כעת נתאר את **אופן פעולת יצירת הטבלאות**:

Create_tables:

מטרות הפונקציה היא לאתחל את הטבלאות איתן נעבוד לאורך התוכנית:

טבלת owners:

מכילה את שמות הowners לפי מפתח owner_id.

טבלת Apartments:

מכילה את השדות הבאים:

Apartment_id,address,city,country,size , המפתח הינו apartment_id והגדרנו בטבלה את הצירוף: address,city,country כUNIQUE בעקבות דרישת ה-PDF.

טבלת Customers:

בדומה לטבלת Owners, מכילה את שמות ה-customers לפי מפתח cust_id.

טבלת Owns:

טבלה המקשרת בין בעלי דירות לבין מפתח הדירה.

מכיוון שלכל דירה ישנו בעלים יחיד אזי המפתח בטבלה זו הוא apartment_id.

בנוסף, השדות הוגדרו ON DELETE CASCADE מכיוון שכאשר בעל דירה או דירה מוסרת מהמערכת היינו מעוניינים לבטל את הקשר Owns בניהם.

טבלת Reviews:

טבלה המכילה את השדות הבאים:

Cust_id,apartment_id, review_date,rating,review_text

כאשר cust_id ו apartment_id הם המפתחות של טבלה זו.
המטרה בטבלה זו היא לקשר בין הדירוגים לבין מזהה הדירות ומזהה הלקוחות.

בנוסף, בדומה לטבלת Owns גם כאן, הגדרנו שמחיקת לקוח/דירה תגרור מחיקת השורות הרלוונטיות מטבלה זו.

טבלת Reserves:

טבלה המכילה את השדות הבאים:

Cust_id,apartment_id,start_date,end_date,total_price.

המטרה בטבלה זו היא לקשר בקלות בין הזמנות חדשות של לקוחות (על פי המזהה שלהם) לבין מזהה דירות.

בנוסף, נבדקים כללים ספציפיים יותר כגון end_date>start_date על מנת לא להכניס הזמנות שגויות.

שרשור מחיקה קיים גם כאן בדומה לטבלה לעיל.

לא פירטנו במסמך זה על התנאים על השדות הספציפים על מנת לא להכביד בפירוט.

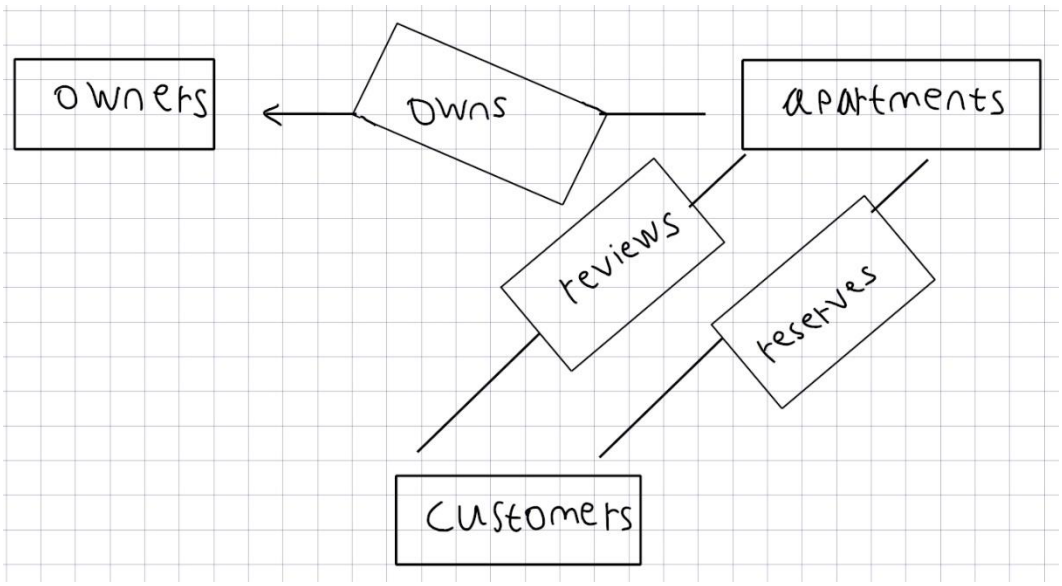
למשל, בעת יצירת טבלת Owners ישנו תנאי Check(Owner_id>0).
ובאופן דומה על שאר המזהים.

Clear_tables:

באמצעות שימוש ב-Truncate אנו מנקים את כלל הטבלאות שיצרנו ב-Create_tables לעיל.

Drop_tables:

באופן דומה לפונקציה לעיל, רק שהפעם על ידי שימוש ב-Drop table אנו מוחקים את הטבלאות שנוצרו ב-create_tables.



CRUD API

add_owner:

הוספת owner חדש לטבלת Owners תוך בדיקת הערכים.

get_owner:

גישה לטבלת Owners והחזרת הowner המתאים למזהה הנתון. מתבצע תוך כדי בדיקת תקינות הערכים.

delete_owner:

גישה לטבלה Owners ומחיקת הרשומה המתאימה למזהה שהתקבל, מכיוון שב Owns יש מפתח זר המתאים לowner_id, הרשומות יימחקו גם משם.

add_apartment:

הוספת דירה חדשה לטבלת Apartments תוך בדיקת הערכים.

get_apartment:

גישה לטבלת Apartments והחזרת הדירה המתאימה למזהה הנתון. מתבצע תוך כדי בדיקת תקינות הערכים.

delete_apartment:

גישה לטבלת Apartments ומחיקת הרשומה המתאימה למזהה שהתקבל, מכיוון שבReserves,Owns,Reviews ישנם מפתחות זרים בעת מחיקת הרשומה גם בטבלאות אלו ימחקו השורות המתאימות.

add_customer:

הוספת לקוח חדש לטבלת Customers תוך בדיקת הערכים.

get_customer:

גישה לטבלת Customers והחזרת הלקוח המתאים למזהה הנתון. מתבצע תוך בדיקת תקינות הערכים.

delete_customer:

גישה לטבלת Customers ומחיקת הרשומה המתאימה למזהה שהתקבל, מכיוון שבReserves,Reviews ישנם מפתחות זרים בעת מחיקת הרשומה גם בטבלאות אלו ימחקו השורות המתאימות.

owner_owns_apartment:

מתבצעת הכנסה של רשומה Owner_id,Apartment_id לטבלת הOwns רק במידה וקיים Owner בעל Owner_id תואם בטבלת הOwners (אם לא קיים, לא מתבצעת הכנסה לטבלה).
בנוסף, אם קיים owner_id אך אין דירה עם מזהה מתאים אז גם כאן לא תתבצע הכנסה.

owner_drops_apartment:

מחיקת הרשומה המתאימה בטבלת Owns אמ"מ קיימת רשומה עם המזהים המתאימים.

במידה ולא, נזרקת השגיאה המתאימה.

get_apartment_owner:

נשלוף מהטבלאות Owns,Owners את השדות Owner_id and Owner_name ,

כאשר מזהה הדירה שהתקבל בקלט שווה לApartment_id בOwns וגם מזהה הבעלי דירה בשתי הטבלאות שווה. ולכן, נקבל את מזהה בעל הדירה ואת שמו במקרה של הצלחה, אחרת, bad_owner.

אנו משתמשים בטבלת Owners על מנת שנוכל לחלץ את השם של בעל הדירה.

get_owner_apartments:

נחלץ מטבלת הOwns את מזהי כל הדירות השייכות לowner_id שקיבלנו בקלט.

ובנוסף, על מנת להחזיר את רשימת הדירות ולא רק את מזהם, נחלץ את השדות של דירה מתוך Apartments. (בדומה לפונקציה הקודמת, גם כאן ישנו שימוש בטבלת יישות (Apartments) על מנת להביא את שאר השדות הנדרשים לפלט).

customer_made_reservation:

אנו בודקים קודם אם ההזמנה מתנגשת (OVERLAPS) עם הזמנה קיימת ב-Reserves.

אם לא - השאילתא מבצעת Insert של הזמנה חדשה ל-Reserves.

אם כן - השאילתא לא משנה את Reserves.

customer_cancelled_reservation:

מוחק הזמנה מ-Reserves.

אם ההזמנה לא קיימת - השאילתא לא משנה את Reserves.

customer_reviewed_apartment:

אנו בודקים קודם אם קיימת הזמנה ב-Reserves שהלקוח ביצע בדירה, שהסתיימה לפני מועד הביקורת.

אם כן - אנו מוסיפים את הביקורת ל-Reviews.

אם לא - השאילתא לא משנה את Reviews.

customer_updated_review:

אנו בודקים קודם אם קיימת ביקורת על שם אותו לקוח ואותה דירה, שניתנה לפני תאריך העדכון.

אם כן - אנו מעדכנים את הביקורת ב-Reviews.

אם לא - השאילתא לא משנה את Reviews.

Basic API

reservations_per_owner:

השדות המחולצים בפונקציה זו הינם רשימת שמות הבעלים וכמות ההזמנות לדירות ברשות כל בעל דירה.

על מנת לחלץ את שם הבעלים השתמשנו בטבלת Owners.

תחילה, נבצע Left Join בין Owners וOwns על מנת לקבל את כלל הבעלים במערכת גם אם אינם בעלי דירות כרגע. ה-JOIN מתבצע על בסיס מזהה הבעלים.

לאחר מכן, נבצע Left Join עם התוצאה של הJOIN הקודם עם טבלת Reserves על בסיס מזהה הדירות.
כעת, ישנה טבלה גדולה המכילה בכל שורה הזמנה ואת השדות הבאים:
Owner_id,Owner_name,Apartment_id, (ופרטים על ההזמנה).

כעת, נבצע פעולת GROUP BY על שם הבעלים,
וביצוע COALSCALE(COUNT(apartment_id,0)) על מנת להתחשב בדירות שבהן לא היו הזמנות לבעלי הדירות.

הסכימה מתבצעת פר שם בעל מכיוון שבוצעה תחת GROUP BY.

get_apartment_rating:

לשם מימוש הפונקציה הזו והפונקציה get_owner_rating, אנו מגדירים view בשם AllApartmentsRating באופן הבא:
ה-view מבצע LEFT JOIN על Apartments,Reviews, מכיוון שאנו רוצים לתת דירוג לכל טבלה, גם אם אין לה ביקורות.
לאחר מכן אנו מבצעים אגרגציה לפי apartment_id, כי אנחנו רוצים לחשב לכל דירה את הדירוג הממוצע שלה.
ה-view עצמו מחזיר טבלה עם 2 עמודות - מספר דירה ודירוג ממוצע.

כעת, השאילתא בפונקציה עצמה פשוט ניגשת לטבלה הזו ובוחרת את השורה עם ה-apartment_id שהועבר כארגומנט,
ומחזירה את הדירוג הממוצע של אותה דירה.

get_owner_rating:

גם בפונקציה זו יש שימוש ב-view שהוגדר בפונקציה הקודמת - AllApartmentsRating.
אנו מבצעים JOIN של טבלת ה-view הנ"ל עם Owns, ובוחרים רק את השורות שיש בהן את ה-owner_id שהועבר כארגומנט.
כעת יש לנו את כל הדירות שאותו owner_id מחזיק, ואנו מחזירים את הממוצע על הדירוגים הממוצעים שלהן.

get_top_customer:

אנו משתמשים ב-subquery שעושה את הדבר הבא:
הולכת ל-Reserves ועושה אגרגציה לפי cust_id, ממיינת את הקבוצות שקיבלנו לפי הגודל שלהן (ואם יש שיוויון, לפי ה-cust_id),
נשארת רק עם הקבוצה הראשונה לפי הסדר הנ"ל, ואז מחזירה את ה-cust_id שלה.
כעיקרון קיבלנו את הלקוח שרצינו - הלקוח עם הכי הרבה הזמנות; אבל צריך גם להחזיר את השם שלו,
לכן עם ה-cust_id שקיבלנו אנו מחפשים אותו ב-Customers כדי למצוא ולהחזיר את שמו.

Advanced API

get_all_location_owners:

תחילה ניצור שני VIEWS:

- AllCityCountryCombinations:
  הטבלה תחלץ רשימה ללא כפילויות של הצירוף city, country.
- CityCountryPerOwner:
  הטבלה החוזרת היא תוצאה של JOIN בין Owns וApartments על בסיס מזהה דירה, על מנת לקבל חזרה טבלה המכילה 3 עמודות: מזהה בעלים, עיר, מדינה.
  כך שאם לבעלים יש דירה בעיר ומדינה מסוימת אזי יופיע כשורה בטבלה זו.

  נבצע JOIN בין הCityCountryPerOwner לבין Owners על בסיס מזהה בעלים, על מנת להוסיף לVIEW השני את שם הבעלים.
  ונחלץ רק את השורות אשר הצירוף city,country מופיע בVIEW הראשון.

  מכיוון שנדרשנו להחזיר את כלל הבעלים שיש להם דירות בכל המקומות האפשריים במערכת, נבצע GROUP BY על בסיס הowner_id, ונבדוק שסכום השורות בטבלה שווה לסכום השורות שבAllcityCountryCombinations , תנאי זה יבטיח שowner יכנס לטבלה הסופית רק אם מכיל את כלל הצירופים הקיימים לעיר-מדינה במערכת.

best_value_for_money:

תחילה ניצור שני VIEWS:

- AverageRatingPerApartment:
  כאשר טבלה זו בעלת שתי עמודות: מזהה דירה והדירוג הממוצע שלה על בסיס GROUP BY apartment_id בטבלת REVIEWS.
- AverageCostPerApartment:
  באופן דומה גם כאן, ישנן שתי עמודות מזהה דירה והמחיר הממוצע שלה כתלות במחיר הממוצע ללילה.
  נבצע GROUP BY apartment_id על טבלת Reserves כאשר חישוב הממוצע יהיה המחיר הכולל חלקי כמות הימים ששהו ועל ערך זה נבצע ממוצע.
  (AVG(total_price / (end_date - start_date)) avg_cost)

נחשב את הדירה המשתלמת ביותר באופן הבא:

נבצע LEFT JOIN בין הVIEW השני לבין הVIEW הראשון על בסיס מזהה הדירה.
זאת על מנת לקבל שורה המכילה את מזהה הדירה, הדירוג שלה והמחיר המשוקלל שלה.

נדרש LEFT JOIN כדי לתמוך גם במצב שאין דירוג לדירה אך עדיין שהו בה.

לאחר מכן, על התוצאה נבצע JOIN עם Apartments כדי לקבל את המידע החסר עבור דירה. (city,address,Country,size).

נחלץ את הRATIO כפי שנדרש Rating/Avg_cost כאשר המיון הוא בסדר יורד ומוגבל לערך בודד על מנת לקבל את הדירה הרווחית ביותר בלבד.

profit_per_month:

קודם כל, אנו יוצרים view בשם MonthsView, שהינו טבלה של עמודה אחת, עם הערכים 1-12 (מסמל את כל החודשים).
שנית, אנו יוצרים view בשם ApartmentsInYear, שלוקח מ-Reserves את כל ההזמנות שתאריך הסיום שלהן נופל בתוך השנה המבוקשת.
אותו view הינו טבלה של שתי עמודות - מחיר ההזמנה והחודש שבו ההזמנה נעשתה (החודש שבו נופל תאריך הסיום).

לבסוף, השאילתא עצמה מבצעת RIGHT JOIN על ApartmentsInYear,MonthsView, כי אנחנו רוצים להחזיר את הרווח עבור כל חודש, גם כאלה שלא היו בהם הזמנות בכלל.
על תוצאת ה-JOIN מבצעים אגרגציה לפי מספר החודש, ולכל קבוצה מחשבים את הרווח של אותו חודש
(סך ההכנסות מההזמנות של אותו חודש, כפול 0.15).
לבסוף הטבלה שחוזרת היא טבלה של 2 עמודות - מספר החודש והרווח של אותו חודש.

get_apartment_recommendation:

נשימה עמוקה, אוקיי, אז ככה:
ה-view הראשון הוא ReducedReviews:
אנחנו הולכים ל-Reviews ושולפים משם רק טאפלים עם דירות שהלקוח הנתון כארגומנט נתן עליהן ביקורת.

ה-view השני הוא JoinedWithRatios:
אנו מבצעים JOIN על ReducedReviews עם עצמו.
מתוך זה, אנו בוחרים רק שורות שה-cust_id שלהן הוא זה שניתן לנו כארגומנט.
מה קיבלנו? כל שורה מתווה לנו ratio בין הלקוח הנתון ללקוח אחר שנתן ביקורת על אותה דירה.
*חדי האבחנה ישימו לב שזה כולל טאפלים של הלקוח הנתון עם עצמו, אבל זה בסדר, כי הם יפולטרו בהמשך.

ה-view השלישי הוא averageRatioPerCustomer:
אנחנו מבצעים אגרגציה על JoinedWithRatios לפי cust_id, ובעצם מחשבים לכל cust_id את ממוצע ה-ratios שלו.
זה מחזיר לנו טבלה של שתי עמודות - cust_id וה-average_ratio שלו. בטבלה יש טאפל לכל לקוח.

זה מטפל לנו בחלק הראשון של הלוגיקה. כעת לחלק השני:

ה-view הרביעי הוא approximationPerApartment:
אנחנו מבצעים JOIN על Reviews,averageRatioPerCustomer, ובוחרים משם רק את הטאפלים עם דירות שהלקוח הנתון לא נתן עליהן ביקורת (כאן מפלטרים את השורות שהזכרנו ב-*).
לאחר מכן, מבצעים אגרגציה לפי apartment_id.
מה קיבלנו? בכל קבוצה (שמתאימה לדירה מסויימת) יש לנו את כל ה-ratios של אותה דירה.
כל טאפל בקבוצה הנ"ל כולל את ה-ratio של לקוח מסויים ואת הדירוג שהוא נתן לדירה המתאימה.
בעצם, כל טאפל כזה הוא approximation (לפי איך שהוגדר ב-pdf), ואנחנו לוקחים את הממוצע של כל ה-approximations בקבוצה.
אנחנו עושים זאת לכל קבוצה, כלומר כל דירה (שהלקוח הנתון אינו ביקר בה).
תוצאת ה-view זו טבלה עם 2 עמודות - מספר דירה, וה-approximation המתאים לה.

כעיקרון קיבלנו מה שרצינו - טבלת עם מספרי דירות ו-approximations; אבל נדרש להחזיר אובייקט דירה, ולא מספר דירה.
לכן בשאילתא עצמה אנחנו מבצעים JOIN על Apartments,approximationPerApartment, וכעת כל טאפל בטבלה המוחזרת כולל:
את כל המידע על הדירה, וגם את ה-approximation שלה.