

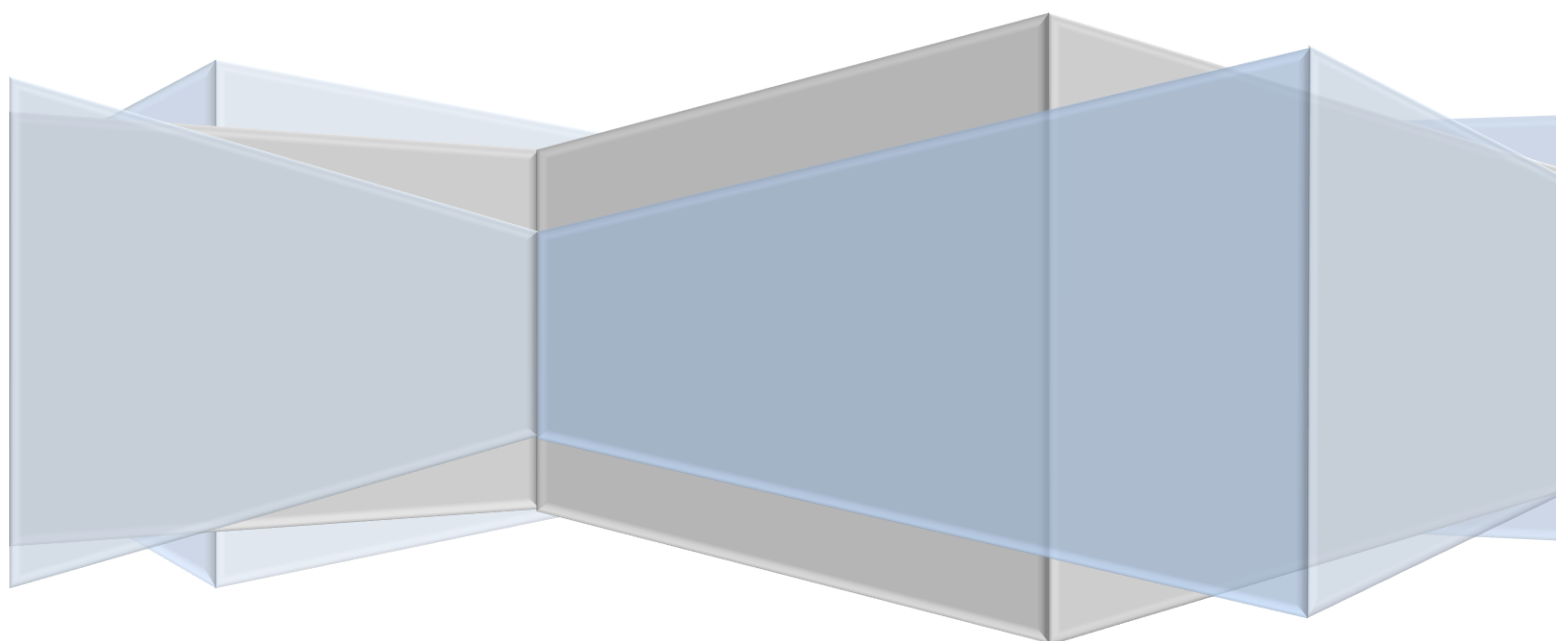
בס"ד

# מיני פרויקט בבסיסי נתונים

מגישים:

שחר מרקוביץ 211491766

נאור שמעון ממן 207341777



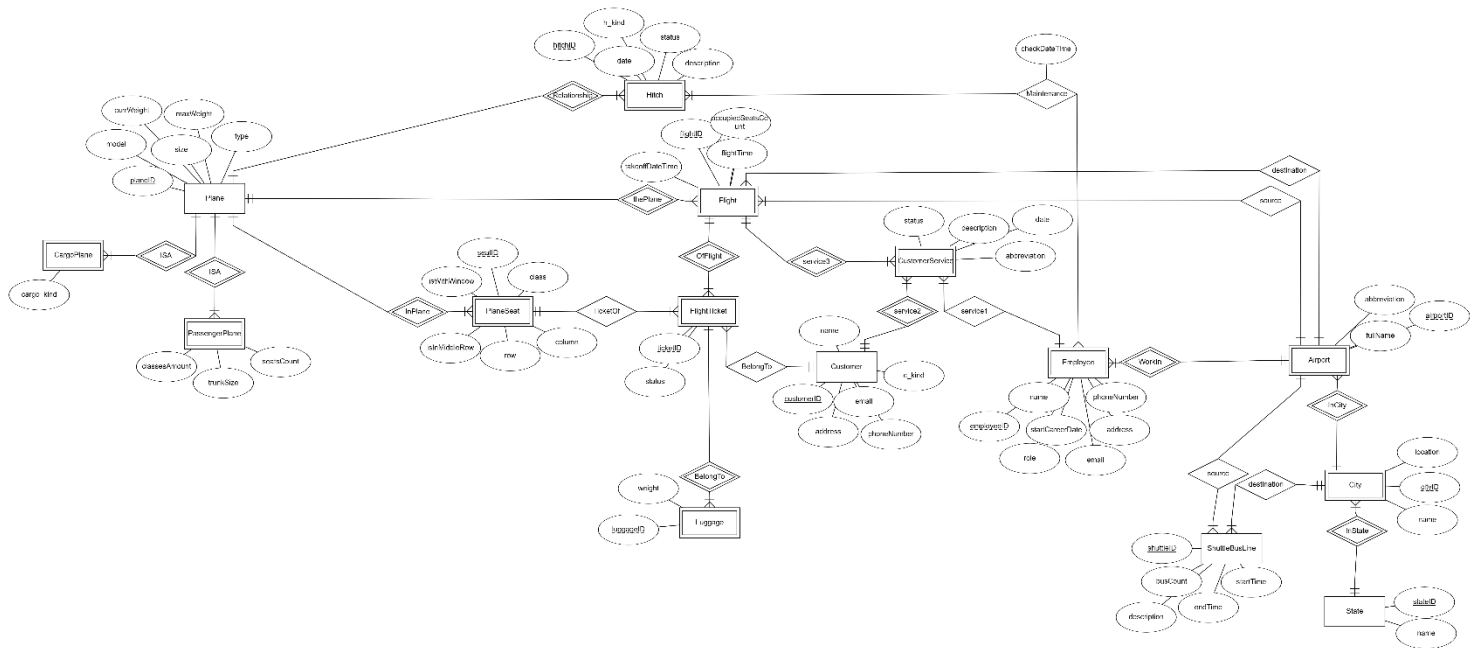
# תוכן עניינים

2.....	מבוא
3.....	עבודת הכנה והכרת התוכנה
3.....	תרשים ERD
4.....	תיאור הישויות והקשרים
4.....	ישויות
4.....	קשרים
4.....	נרמול הטבלאות
4.....	פרוקים
5.....	תרשים DSD
6.....	הפרויקט שלנו
6.....	תרשים ERD
7.....	תיאור הישויות והקשרים
7.....	ישויות
8.....	קשרים
8.....	נרמול הטבלאות
8.....	תרשים DSD
9.....	יצירת הטבלאות
10.....	הכנסת נתונים
11.....	שאלות SQL
11.....	בחירה - SELECT
12.....	אינדקסים
14.....	הרשאות
14.....	VIEWS
16.....	פונקציות
18.....	נספחים
18.....	נספח ראשון: שאלות ואינדקסים
21.....	נספח שני: עדכון, מחיקה
24.....	נספח שלישי: VIEWS
25.....	נספח רביעי: פונקציות

## מבוא

- **מטוס** - מזהה מטוס , גודל, דגם מטוס, משקל מקסימלי, משקל נוכחי, סוג
- **מטוס מטען** - מזהה מטוס, סוג תכולה
- **מטוס נוסעים** - מזהה מטוס, מס' מחלקות, גודל תא מטען, מס' כסאות
- **שדה תעופה** - מזהה שדה תעופה, מזהה עיר, שם ארוך, שם קצר
- **טיסה** - מזהה טיסה, מזהה מטוס, מזהה שדה תעופה מקור, מזהה שדה תעופה יעד, זמן טיסה מוערך, תאריך ושעת המראה
- **עיר** - מזהה עיר, מזהה מדינה, שם עיר, מיקום
- **מדינה** - מזהה מדינה, שם מדינה
- **שאטל** - מזהה שאטל, מזהה שדה תעופה, מזהה עיר יעד, מס' שאטלים בקו, שעת הפעלה, שעת סיום, תיאור
- **תקלה** - מזהה תקלה, מזהה מטוס, סוג תקלה, מצב תקלה, תיאור תקלה, תאריך פניה
- **תחזוקה** - מזהה תקלה, מזהה עובד, תאריך בדיקה
- **שירות לקוחות** - מזהה לקוח, מזהה טיסה, מזהה עובד, תיאור פנייה, תקציר פנייה, תאריך פנייה, סטטוס
- **כיסא במטוס** - מזהה כיסא, מזהה מטוס, מחלקה, מס' שורה, מס' עמודה, האם ליד חלון, האם בשורה אמצעית
- **כרטיס טיסה** - מזהה כרטיס, מס' כיסא, מזהה טיסה, מזהה לקוח, סטטוס
- **כבודת נוסעים** - מזהה כבודת נוסעים, מזהה כרטיס, משקל.

## תרשים ERD



## תיאור הישויות והקשרים

### ישויות

- State - מאופיין במס' מזהה של מדינה ושם מדינה.
- City – מאופיין ע"י מס' מזהה עיר, מס' מזהה מדינה, שם עיר וקורדינאטות מיקום.
- Airport – מאופיין ע"י מס' מזהה נמל תעופה, מס' מזהה עיר, שם מלא, ושם מקוצר.
- Flight – מאופיין ע"י מס' מזהה טיסה, מס' מזהה מטוס, מס' מזהה נמל תעופה מקור, מס' מזהה נמל תעופה יעד, זמן טיסה בדקות ותאריך ושעת יציאה.
- PlaneSeat – מאופיין ע"י מס' מזהה כיסא, מס' מזהה מטוס, מחלקה(תיירים-'T', עסקים-'B', ראשונה-'F') מס' שורה, מס' טור, האם ליד חלון והאם בטור אמצעי.
- ShuttleBusLine – מאופיין ע"י מס' מזהה שאטל, כמות שאטלים בקו, שעת התחלה, שעת סיום, תיאור, מס' מזהה נמל תעופה מקור ומס' מזהה עיר יעד.

### קשרים

- לכל עיר - כל עיר יכולה להיות קיימת במדינה אחת. במדינה אחת יכולים להיות הרבה ערים.
- לכל נמל תעופה - כל נמל תעופה יכול להיות קיים בעיר אחת. בעיר אחת יכולים להיות הרבה נמלי תעופה.
- לכל טיסה – לכל טיסה יש נמלי תעופה מקור ויעד יחידים ומטוס אחד.
- לכל כיסא במטוס – יש רק מטוס אחד אליו הוא שייך. לכל מטוס יש הרבה כיסאות.

### נרמול הטבלאות

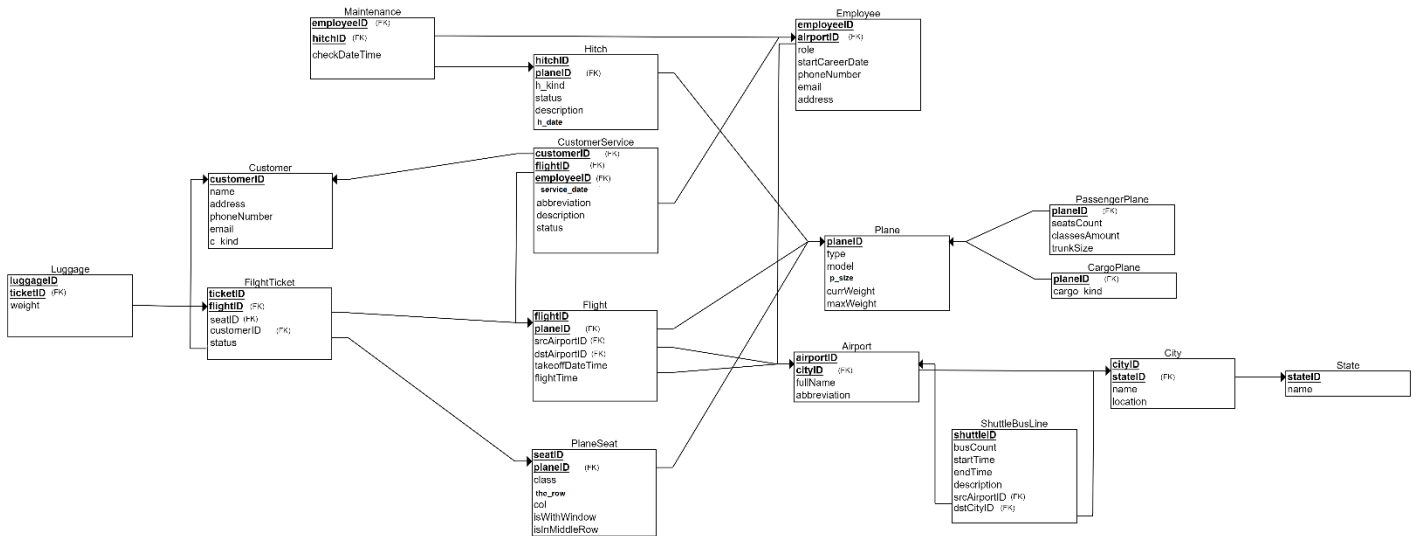
- State(stateID, name)
- City(cityID, stated, name, location)
- Airport(airportID, CityID, fullName, abbreviation)
- Flight(flightID, planeID, srcAirportID, dstAirportID, takeOffDateTime, flightTime)
- PlaneSeat(seatID, planeID, class, the\_row, col, isWithWindow, isInMiddleRow)
- ShuttleBusLine(shuttleID, busCount, startTime, endTime, description, srcAirportID, dstCityID)

### פרוקים

היחסים עומדים ב- 3NF וב- BCNF : מכיוון שבכל טבלה, התלויות הפונקציונאליות הלא-טריוויאליות הן מהמפתח אל תכונות נוספות לכן מתקיים שלכל  $X \rightarrow Y$ ,  $X$  הוא מפתח ולכן הם עומד בתנאים.

## DSD תרשים

כאן ניתן לראות את ה- DSD שהפקנו מתרשים ה- ERD שיצרנו.

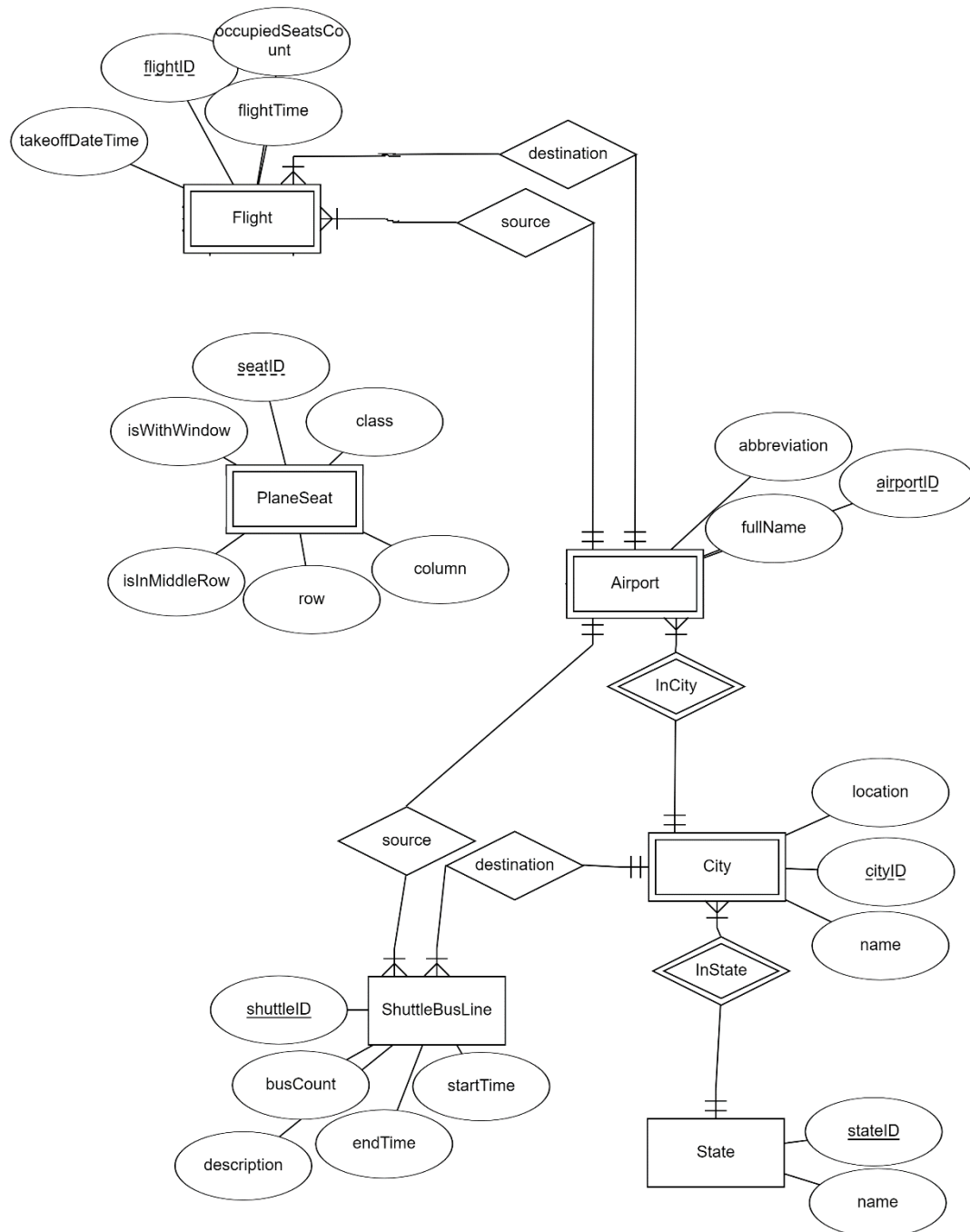


# הפרויקט שלנו

החלק שלנו התמקד ביצירת הגופים הקיימים במערכת.

## תרשים ERD

כפי שהזכרנו, במחלקה שלנו, ישנן 6 ישויות: שדה תעופה, טיסה, עיר, מדינה, שאטל וכיסא במטוס. בשלב הראשון יצרנו תרשים ERD שיתאר את הקשרים בין הישויות הללו ואת התכונות שלהן.



## תיאור הישויות והקשרים

### ישויות

- **Airport** – ישות זאת אחראית על כל השדות התעופה הנמצאים במערכת.  
ישות זאת הינה חלשה, כיוון שהיא תלויה בישות עיר.
  - airportID - מזהה שדה תעופה (PK)
  - cityID - מזהה עיר (FK)
  - fullName - שם ארוך
  - Abbreviation - שם קצר
- **טיסה** - ישות זאת אחראית על כל הטיסות בעבר ובעתיד הנמצאות במערכת.  
ישות זאת הינה חלשה, כיוון שהיא תלויה בישות מטוס.
  - flightID - מזהה טיסה (PK)
  - planeID - מזהה מטוס (FK)
  - SrcAirportID - מזהה שדה תעופה מקור (FK)
  - dstAirportID - מזהה שדה תעופה יעד (FK)
  - takeOffDateTime - תאריך ושעת המראה
  - flightTime - זמן טיסה בדקות
- **City** - ישות זאת אחראית על כל הערים הנמצאים במערכת.  
ישות זאת הינה חלשה, כיוון שהיא תלויה בישות מדינה.
  - cityID - מזהה עיר (PK)
  - stateID - מזהה מדינה (FK)
  - Name - שם עיר
  - Location - מיקום
- **State** - ישות זאת אחראית על כל המדינות הנמצאות במערכת.  
ישות זאת הינה חזקה, כיוון שיכולה להתקיים ללא תלות בישות אחרת.
  - StateID - מזהה מדינה (PK)
  - Name - שם מדינה
- **ShuttleBusLine** - ישות זאת אחראית על כל השאטלים הנמצאים במערכת.  
ישות זאת הינה חלשה, כיוון שהיא תלויה בישויות עיר ושדה תעופה.
  - shuttleID - מזהה שאטל (PK)
  - srcAirportID - מזהה שדה תעופה (FK)
  - dstAirportID - מזהה עיר יעד (FK)
  - busCount - מס' שאטלים בקו
  - startTime - שעת הפעלה
  - endTime - שעת סיום
  - Description - תיאור
- **PlaneSeat** - ישות זאת אחראית על כל הכיסאות במטוס הנמצאים במערכת  
ישות זאת הינה חלשה, כיוון שהיא תלויה בישות מטוס
  - SeatID - מזהה כיסא (PK)
  - PlaneID - מזהה מטוס (FK)
  - Class - מחלקה
  - The\_row - מס' שורה
  - Col - מס' עמודה
  - isWithWindow - האם ליד חלון
  - isInMiddleRow - האם בשורה אמצעית



## קשרים

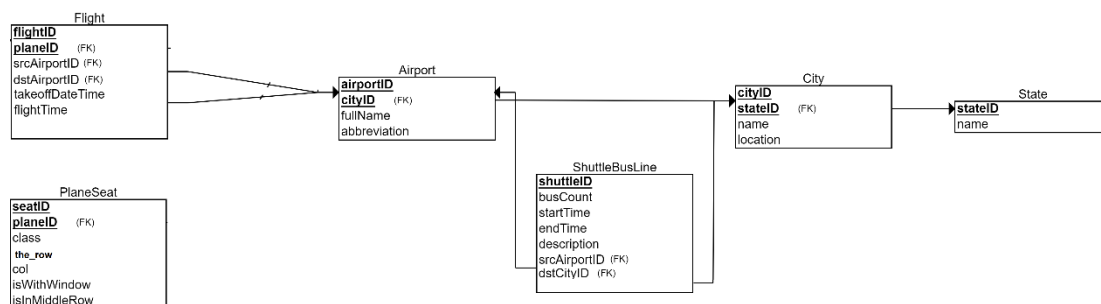
- destination – הקשר בין Flight ל-Airport. הקשר הוא M:1 משום שיכול להיות בנמל תעופה אחד הרבה טיסות, אבל טיסה אחת מגיעה לנמל תעופה אחד.
- source – הקשר בין Flight ל-Airport. הקשר הוא M:1 משום שיכול להיות בנמל תעופה אחד הרבה טיסות, אבל טיסה אחת יוצאת מנמל תעופה אחד.
- InCity – הקשר בין Airport ל-City. הקשר הינו חלש, כיוון שמקשר בין ישות חלשה לחזקה. הקשר הוא M:1 משום שיכול להיות בעיר אחת הרבה נמלי תעופה, אבל נמל תעופה אחד שייכת לעיר אחת.
- Instate – הקשר בין State ל-City. הקשר הינו חלש, כיוון שמקשר בין ישות חלשה לחזקה. הקשר הוא M:1 משום שיכול להיות במדינה אחת הרבה ערים, אבל נמל תעופה אחד נמצא בעיר אחת.
- destination – הקשר בין ShuttleBusLine ל-City. הקשר הוא M:1 משום שיכול להיות שהרבה שאטלים מגיעים לאותה עיר, אבל שאטל אחד מגיע לעיר אחת.
- source – הקשר בין ShuttleBusLine ל-Airport. הקשר הוא M:1 משום שיכול להיות שהרבה שאטלים יוצאים מאותו נמל תעופה, אבל שאטל אחד יוצא מנמל תעופה אחד.

נרמול הטבלאות

היחסים עומדים ב- 3NF וב- BCNF : מכיוון שבכל טבלה, התלויות הפונקציונאליות הלא-טריוויאליות הן מהמפתח אל תכונות נוספות לכן מתקיים שלכל  $X \rightarrow Y$ ,  $X$  הוא מפתח ולכן הם עומד בתנאים.

תרשים DSD

על פי תרשים ה- ERD ועל ידי הבנת הקשרים בין הישויות, יצרנו תרשים DSD עבור החלק שלנו במערכת: מחלקת מתקני המלון.



## יצירת הטבלאות

אחרי שהבנו כיצד בסיס הנתונים צריך להראות בצורה מדויקת, מה תכיל כל טבלה ומהם הקשרים בין כל הטבלאות, ניגשנו ליצירת הטבלאות בפועל בעזרת פקודות הcreate table.

יצרנו קוד לייצור הטבלאות באמצעות export SQL של האתר erdPlus, יצרנו קובץ SQL ואז העתקנו את קוד ה-SQL של כל טבלה אל תוכנת ה-plsql לשם יצירת הטבלאות בפועל.

```
CREATE TABLE State (
    stateID INT NOT NULL,
    name VARCHAR(50) NOT NULL,
    PRIMARY KEY (stateID)
);

CREATE TABLE PlaneSeat (
    seatID INT NOT NULL unique,
    class CHAR(1) NOT NULL,
    the_row INT NOT NULL,
    col INT NOT NULL,
    isWithWindow INT NOT NULL,
    isInMiddleRow INT NOT NULL,
    planeID INT NOT NULL,
    PRIMARY KEY (seatID, planeID),
    FOREIGN KEY (planeID) REFERENCES Plane(planeID)
);

CREATE TABLE City (
    cityID INT NOT NULL unique,
    name VARCHAR(50) NOT NULL,
    location VARCHAR(100) NOT NULL,
    stateID INT NOT NULL,
    PRIMARY KEY (cityID, stateID),
    FOREIGN KEY (stateID) REFERENCES State(stateID)
);

CREATE TABLE Airport (
    airportID INT NOT NULL unique,
    fullName VARCHAR(50) NOT NULL,
    abbreviation VARCHAR(8) NOT NULL,
    cityID INT NOT NULL,
    PRIMARY KEY (airportID, cityID),
    FOREIGN KEY (cityID) REFERENCES City(cityID)
);

CREATE TABLE Flight (
    flightID INT NOT NULL unique,
    flightTime DATE NOT NULL,
    takeOffDateTime DATE NOT NULL,
    planeID INT NOT NULL,
    srcAirportID INT NOT NULL,
    dstAirportID INT NOT NULL,
    PRIMARY KEY (flightID, planeID),
    FOREIGN KEY (planeID) REFERENCES Plane(planeID),
    FOREIGN KEY (srcAirportID) REFERENCES Airport(airportID),
    FOREIGN KEY (dstAirportID) REFERENCES Airport(airportID)
);
```

```
CREATE TABLE ShuttleBusLine (
    shuttleID INT NOT NULL,
    busCount INT NOT NULL,
    startTime DATE NOT NULL,
    endTime DATE NOT NULL,
    description VARCHAR(500) NOT NULL,
    srcAirportID INT NOT NULL,
    dstCityID INT NOT NULL,
    PRIMARY KEY (shuttleID),
    FOREIGN KEY (srcAirportID) REFERENCES Airport(airportID),
    FOREIGN KEY (dstCityID) REFERENCES City(cityID)
);
```

### **הכנסת נתונים**

על מנת לאכלס את הטבלאות שיצרנו בנתונים, השתמשנו בספריית פייתון הנותנת מידע על ערים, שדות תעופה, שמות, מקומים, וכו' ויצרנו כך בצורה אקראית טיסות משדות תעופה שונים לאחרים. יצרנו ישר את השאילות insert של SQL בפייתון ואחרי כן העתקנו והרצנו בsqlplus אותם.

קוד אכלוס הנתונים לטבלאות שלנו מצורף בנספח הראשון.

## שאלות SQL

### בחירה - SELECT

לאחר שיצרנו את בסיס הנתונים והכנסנו לתוכו מידע, כתבנו כמה שאלות מעניינות על מנת לתשאל אותו.

שמונת השאלות הבאות נעשו רק על בסיסי הנתונים שהיו באחריותנו:

1. שאלתה המחזירה מס' מזהה טיסה, זמן המראה, מודל מטוס, נמל תעופה מקור ויעד של כל הטיסות בהינתן שם של עיר מסוימת.
2. שאלתה המחזירה מס' מזהה של נמל תעופה, שמו וקיצור השם שלו בעיר מסוימת בהינתן שם של עיר.
3. שאלתה המחזירה עבור כל נמל תעופה כמה שאטלים יוצאים ממנו.
4. שאלתה המחזירה עבור כל מטוס את כמות הכיסאות הכוללת בו וכמות הכיסאות עם חלון.
5. שאלתה המחזירה זמן המראה של טיסה בהינתן שמות של נמלי תעופה מקור ויעד.
6. שאלתה המחזירה עבור כל מטוס כמה כיסאות יש בו במחלקה ראשונה, כמה במחלקת עסקים וכמה במחלקת תיירים.
7. שאלתה הבודקת האם קיים מטוס שמשקלו האמיתי גדול יותר ממשקלו המקסימלי המותר, אם יש מטוס כזה מוחזר מזהה שלו ומודל המטוס, משקל מקסימלי ומתן ומשקל נוכחי.
8. שאלתה המחזירה עבור כל עיר את כל הערים שניתן להגיע אליהם ע"י שאטלים, עם החלפות.

שלוש השאלות הבאות נעשו על כל בסיסי הנתונים של כל הפרויקט:

1. שאלתה המחזירה עבור כל טיסה את כמות הכרטיסים שנמכרו.
2. שאלתה המחזירה עבור כל נמל תעופה את פרטי העובד המצטיין – העובד שטיפל בהכי הרבה פניות משירות לקוחות.
3. שאלתה המחזירה את 40 הטיסות שבהם היו הכי הרבה פניות לשירות לקוחות.

הקוד של כל השאלות האלה נמצא בנספח השני.

## אינדקסים

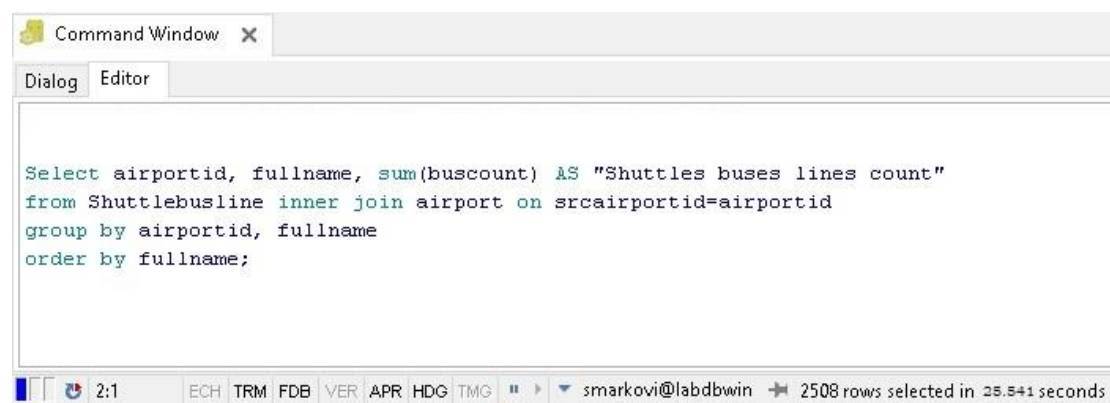
אינדקסים עוזרים למצוא במהירות גדולה יותר נתונים שנשמרו בטבלאות בבסיס הנתונים. אפשר לדמות את האינדקסים כמו מראה מקום בספר. במקום שנקרא את כל הספר כדי למצוא את מה שאנחנו מחפשים נלך למראה מקום שיראה לנו את כל המקומות שבהם מוזכר הנושא שאנחנו מחפשים. השימוש באינדקסים יחסוך לנו זמן ויהפוך את תהליך החיפוש ליעיל יותר. מהבחינה הזו האינדקסים בטבלאות של ה-SQL זהים לאינדקס בספר. במידה ולא נגדיר אינדקס לטבלה אז בכל שאילתה על הטבלה השאילתה תגרום למעבר על כל הרשומות בטבלה עד שתמצא את כל הרשומות העונות למה שחיפשנו. כשנגדיר אינדקס מתאים אז החיפוש יהיה מהיר יותר כי הפניה לבסיס הנתונים תגרום לזה שמנוע החיפוש בבסיס הנתונים יפנה קודם לאינדקס וילך לרשומות המתאימות על פי מה שרשום באינדקס.

לכן יצרנו את אינדקסים הבאים שמקצרים את תהליך ביצוע השאילתות:

```
create index bus_id_index on Shuttlebusline(srcAirportID);
```

- שאלנו שאילתה המחזירה עבור כל נמל תעופה כמה שאטלים יוצאים ממנו. האינדקס שיצרנו היה על שאטל ולכן כאשר עשינו `group by` בשאילתה לפי מס' מזהה שאטל החיפוש היה מהיר יותר וקיבלנו את התוצאה מהר יותר.

### לפני השימוש באינדקס:



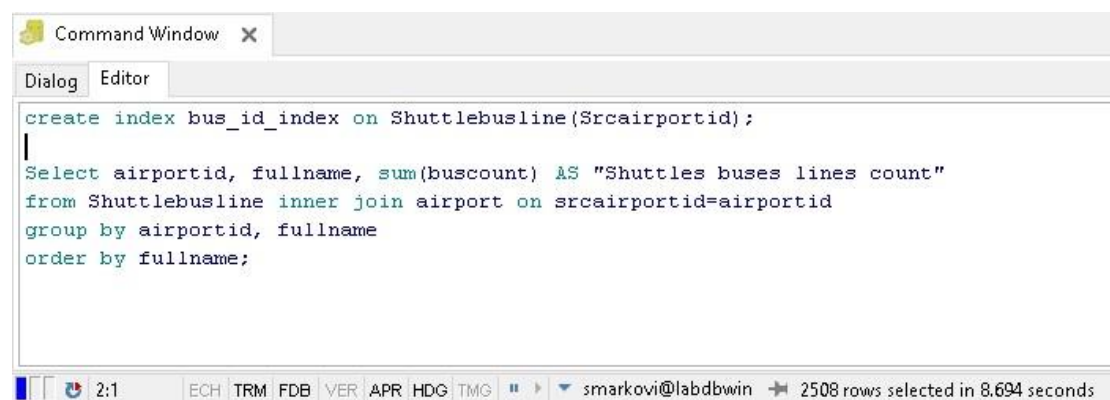
```

Select airportid, fullname, sum(buscount) AS "Shuttles buses lines count"
from Shuttlebusline inner join airport on srcairportid=airportid
group by airportid, fullname
order by fullname;

```

2:1 ECH TRM FDB VER APR HDG TMG smarkovi@labdbwin 2508 rows selected in 25.541 seconds

### אחרי השימוש באינדקס:



```

create index bus_id_index on Shuttlebusline(srcairportid);
Select airportid, fullname, sum(buscount) AS "Shuttles buses lines count"
from Shuttlebusline inner join airport on srcairportid=airportid
group by airportid, fullname
order by fullname;

```

2:1 ECH TRM FDB VER APR HDG TMG smarkovi@labdbwin 2508 rows selected in 8.694 seconds

```
create index plane_id_index on Shuttlebusline(planeid, model, p_size);
```

2. שאלנו שאילתה המחזירה עבור כל מטוס את כמות הכיסאות הכוללת בו וכמות הכיסאות עם חלון.

כמו כן, שאלנו שאילתה המחזירה עבור כל מטוס כמה כיסאות יש בו במחלקה ראשונה, כמה במחלקת עסקים וכמה במחלקת תיירים.

האינדקס שיצרנו היה על מזהה מטוס, גודלו והמודל שלו, וכאשר חיפשנו את הפרטים הבאים בשני השאילתות שהזכרנו קיבלנו את מבוקשתנו מהר יותר.

### לפני השימוש באינדקס:

Command Window

Dialog Editor

```
select planeid, model, p_size as "seats count", sum(isWithWindow) as "seats with windows"
from plane natural join planeSeat
group by planeid, model, p_size;
```

1:62 ECH TRM FDB VER APR HDG TMG smarkovi@labdbwin 400 rows selected in 5.162 seconds

### אחרי השימוש באינדקס:

Command Window

Dialog Editor

```
create index plane_id_index on plane(planeid, model, p_size);
select planeid, model, p_size as "seats count", sum(isWithWindow) as "seats with windows"
from plane natural join planeSeat
group by planeid, model, p_size;
```

1:62 ECH TRM FDB VER APR HDG TMG smarkovi@labdbwin 400 rows selected in 1.621 seconds

### לפני השימוש באינדקס:

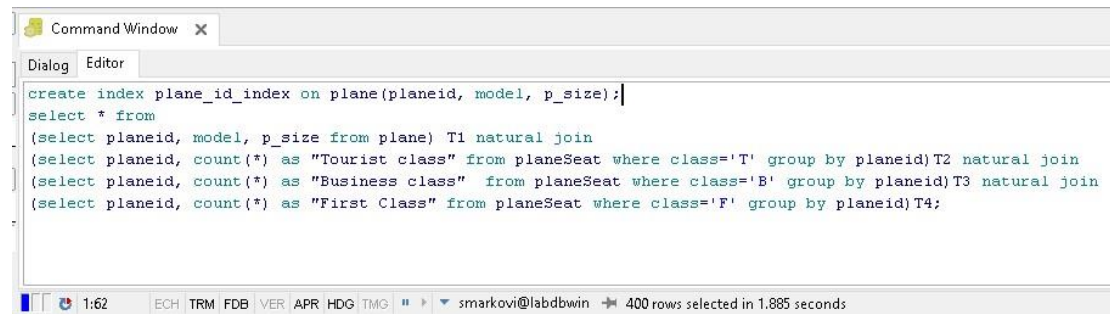
Command Window

Dialog Editor

```
select * from
(select planeid, model, p_size from plane) T1 natural join
(select planeid, count(*) as "Tourist class" from planeSeat where class='T' group by planeid) T2 natural join
(select planeid, count(*) as "Business class" from planeSeat where class='B' group by planeid) T3 natural join
(select planeid, count(*) as "First Class" from planeSeat where class='F' group by planeid) T4;
```

1:62 ECH TRM FDB VER APR HDG TMG smarkovi@labdbwin 400 rows selected in 4.771 seconds

## אחרי השימוש באינדקס:



```

Command Window
Dialog Editor
create index plane_id_index on plane(planeid, model, p_size);
select * from
(select planeid, model, p_size from plane) T1 natural join
(select planeid, count(*) as "Tourist class" from planeSeat where class='T' group by planeid)T2 natural join
(select planeid, count(*) as "Business class" from planeSeat where class='B' group by planeid)T3 natural join
(select planeid, count(*) as "First Class" from planeSeat where class='F' group by planeid)T4;
1:62 ECH TRM FDB VER APR HDG TMG smarkovi@labdbwin 400 rows selected in 1.885 seconds

```

## הרשאות

כדי שנוכל לעבוד ככיתה שלמה המפתחת בסיס נתונים כאשר כל קבוצה בונה טבלאות שונות לבסיס הנתונים ואנו צריכים לגשת לטבלאות שלהם כדי לקחת מידע אנו צריכים לקבל מיוצרי הטבלאות הרשאה כדי שנוכל להשתמש בטבלאות שלהם.

במקרה שלנו היינו צריכים לתת הרשאות לכולם מהסיבה הפשוטה שאנחנו היינו די תלויים באחרים והם בנו מכיוון שהזמנת כרטיס טיסה תלוי במושבים הפנויים שהם באחריותנו, וטיסות תלויות במטוסים אשר לא היו באחריותנו

## Views

VIEWS הם טבלאות וירטואליות. VIEWS מכילים הגדרות של עמודות וסוגי מידע שאותן עמודות יכולות להכיל. ההבדל בין הטבלאות לבין ה-VIEWS הוא שבטבלאות נשמרים נתונים באופן פיזי ואילו ב-VIEWS הנתונים לא נשמרים באופן פיזי בתוכם אלא הם רק מציגים נתונים הנשמרים בטבלאות. לכן לא ניתן לעדכן או להוסיף נתונים לחלק מה-VIEWS, כתלות בצורת הגדרתם שנלמדה בקורס התיאורטי, כפי שעושים לטבלאות.

יצרנו 3 views:

1. View המאחד את מס' הזיהוי של הערים שיש שאטל ביניהם,
2. View של שירותים המאחד בין נותן שירות, מבקש השירות והטיסה שבה ניתן השירות,
3. ו- View המספק מידע על שמות הערים שטיסה מסוימת יוצאת ומגיעה אליהם.

יצרנו view אלה כיוון שרבות מהשאלות שלנו מתייחסות לעמודות שנמצאות בהן ולכן יעיל יותר לעבוד עימן באופן נפרד, בלי התייחסות לטבלאות המלאות.

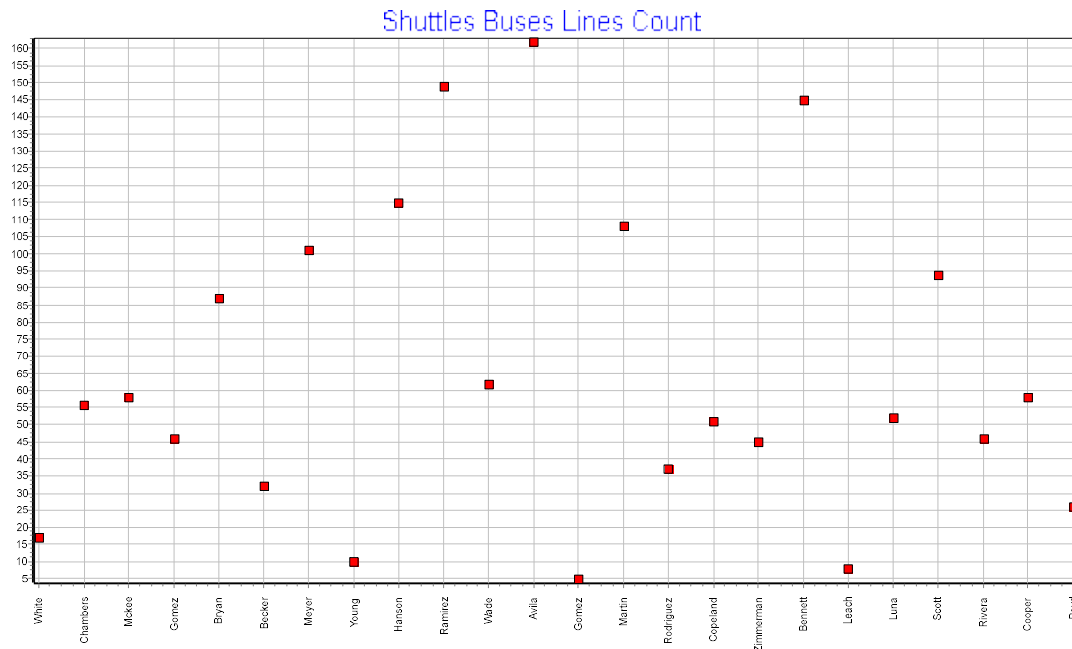
הקוד של יצירת ה-views ועדכון של השאלות המתאימות כך שישתמשו בהן ויהיו יעילות יותר מצויות בנספח השלישי.

## Graphs

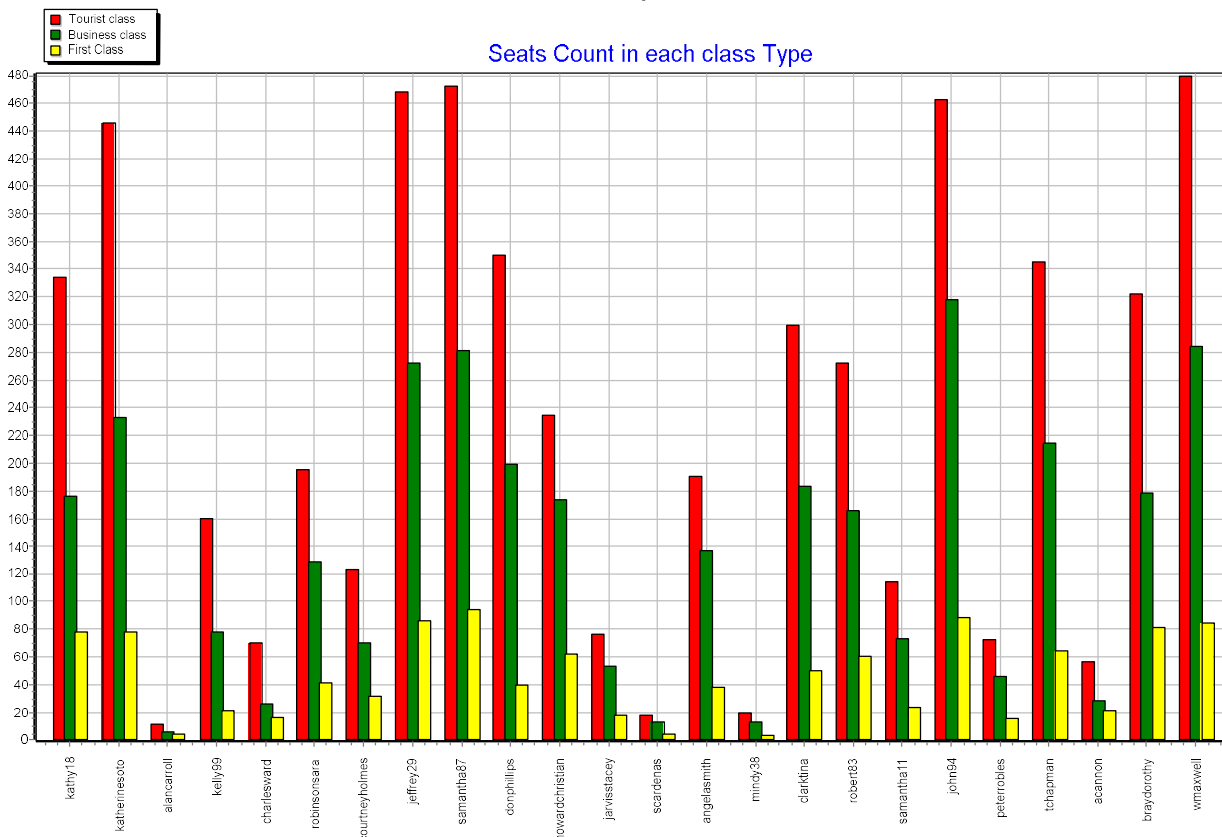
כיום Oracle PLSQL מאפשרים ליצור גרפים מהפלט של שאילתות SQL, מה שהופך אותן לברורות אף יותר ומאפשר לקרוא ולהסיק נתונים ברמה גבוהה יותר.

בפרויקט זה עשינו שימוש בשני גרפים כפי שניתן לראות:

1. הגרף שלפינו מתאר את השאילתה שמחזירה עבור כל נמל תעופה כמה שאטלים יוצאים ממנו. הגרף מציג את כמות השאלטים, כפונקציה של נמל תעופה:



2. הגרף שלפינו מתאר את השאילתה שמחזירה עבור כל מטוס כמה כיסאות יש בו במחלקה ראשונה, כמה במחלקת עסקים וכמה במחלקת תיירים. הגרף מציג את מס' הכיסאות בכל אחת מן המחלקות במודלים שונים של מטוס. כדי שנתונים יהיו יותר ברורים – מוצגים רק 40 דגמי מטוסים:





## פונקציות

פונקציה בשפת SQL היא צורה מיוחדת של פקודה אשר מבצעת פעולות שונות על הנתונים בבסיס הנתונים.

בעזרת פונקציות ופרוצדורות, יצרנו שאילתות חדשות המאפשרות עדכון/תשאול פרטים בצורה נוחה (ומאובטחת) יותר:

(הקוד שיוצר את הפונקציות והפרוצדורות הבאות נמצא בנספח הרביעי)

1. פרוצדורה המאפשרת לעדכן את שעת היציאה של טיסה בגלל עיכוב בהמראה. בהינתן מס' מזהה טיסה וזמן העיכוב בשעות, שעת ההמראה של אותה טיסה מעודכנת:

```
SQL> select flightid, TAKEOFFDATETIME from flight where flightid = 15;
```

FLIGHTID	TAKEOFFDATETIME
15	30/04/2021 07:0

```
SQL> select flightid, TAKEOFFDATETIME from flight where flightid = 15;
```

FLIGHTID	TAKEOFFDATETIME
15	30/04/2021 08:0

2. חלק מן המדינות מעוניינות שהתיאור של כל השאטלים שיוצאים מהמדינה שלהם, יצהיר שהוא יוצא מהמדינה שלהם. לכן יצרנו פרוצדורה שבהינתן שם של מדינה, מעדכנת את התיאור של כל השאטלים שיוצאים מנמל תעופה שבתוך אותה מדינה להצהיר שהם יוצאים ממנה:

```
SQL> select description from Shuttlebusline where description like 'State:%';
```

DESCRIPTION

```
State: Odonnellmouth . Description: System third attention heavy. Put dark type.
State: Odonnellmouth . Description: Almost arrive produce listen drive expect co
State: Odonnellmouth . Description: Five support fine on health. Environment gen
State: Odonnellmouth . Description: It matter oil daughter when such identify. O
State: Odonnellmouth . Description: Blue as Mrs information sister question. Fie
```

SQL> |

235:6 ECH TRM FDB VER APR HDG TMG smarkovi@labdbwin 5 rows selected in 0.055 seconds

3. אנחנו רוצים לאפשר לנמלי תעופה להציג את הטיסה הבאה שיוצאת מהן. לכן יצרנו פונקציה שבהינתן מס' מזהה של נמל תעופה מחזירה את התאריך והשעה הבאים שיוצא ממנה מטוס:

Command Window: GetNextFlight.fnc Script for GETNEXTFLIGHT@LABDBWIN

Test script DBMS Output Statistics Profiler Trace

```

1 begin
2   -- Call the function
3   :takeovertime := getnextflight(airport_id => :airport_id);
4 end;
```

Variable	Type	Value
airport_id	Float	125
takeovertime	String	16-JUN-21

4. הלקוחות שלנו רוצים לדעת האם קיימת טיסה בין שני ערים מסוימות ואם כן מתי היא יוצאת.

לכן יצרנו פונקציה המספקת מידע זה:

The screenshot shows a database IDE interface. At the top, there are tabs for 'Command Window', 'IsHasFlight.fnc', and 'Script for ISHASFLIGHT@LABDBWIN'. Below the tabs, there are buttons for 'Test script', 'DBMS Output', 'Statistics', 'Profiler', and 'Trace'. The main area displays a SQL script:

```

1 begin
2     -- Call the function
3     :IsHasFlight := ishasflight(srccityname => :srccityname,
4                               dstcityname => :dstcityname,
5                               takeovertime => :takeovertime);
6 end;
7 s

```

Below the script, there is a table showing the results of the execution:

Variable	Type	Value
result	String	
srccityname	String	Johnson
dstcityname	String	Brewer
takeovertime	Date	04/07/2021 13:30:00
IsHasFlight	String	true

# נספחים

## נספח ראשון: הכנסת נתונים

```
from functools import reduce
import random
import pandas as pd
import names
from faker import Faker
```

```
FAKE = Faker()
```

### i. Generate States:

```
def gen_states(n: int):
    r = range(1, n + 1)
    stateID = r
    city_name = list(map(lambda a: FAKE.city(), r))
    coloms_name = ("stateID", "name")
    t_name = "state"
    data = ""
    for i in r:
        data += f"insert into {t_name} ({', '.join(coloms_name)}) values ({stateID[i-1]}, '{city_name[i-1]}');\n"
    with open("gen_states.sql", "w") as f:
        f.write(data)
gen_states(500)
```

### ii. Generate Cities:

```
def gen_cities(n: int):
    r = range(1, n + 1)
    cityID = r
    city_name = list(map(lambda a: FAKE.city(), r))
    location = list(map(lambda a: FAKE.coordinate(), r))
    stateID = list(map(lambda a: random.randint(1, 501), r))
    coloms_name = ("cityID", "name", "location", "stateID")
    t_name = "city"
    data = ""
    for i in r:
        data += f"insert into {t_name} ({', '.join(coloms_name)}) values ({cityID[i-1]}, '{city_name[i-1]}', {str(location[i-1])}, {stateID[i-1]});\n"
    with open("gen_cities.sql", "w") as f:
        f.write(data)
gen_cities(4000)
```

## iii. Generate Airports:

```
def gen_airports(n: int):
    r = range(1, n + 1)
    airportID = r
    name = list(map(lambda a: FAKE.last_name(), r))
    abbreviation = list(map(lambda a: "".join(FAKE.random_letters()[:4]), r))
    cityID = list(map(lambda a: random.randint(1, n), r))
    coloms_name = ("airportID", "fullName", "abbreviation", "cityID")
    t_name = "airport"
    data = ""
    for i in r:
        data += f"insert into {t_name} ({','.join(coloms_name)}) values ({airportID[i-1]}, '{name[i-1]}', '{abbreviation[i-1]}', {cityID[i-1]});\n"
    with open("gen_airports.sql", "w") as f:
        f.write(data)
gen_airports(4000)
```

## iv. Generate Flights:

```
MAX_PLANE_ID = 500
from random import randint, choice
from datetime import datetime, timedelta

flight_sql_file = open("gen_flightdata.sql", "w+")
SIZE = 20000
PLAIN_COUNT = 4000

def main():
    for id in range(1, SIZE):
        flight_date = datetime.now() + timedelta(days=randint(-50, 100), hours=randint(-100, 100))
        flight_date = flight_date.replace(second=0, minute=choice([0, 15, 30, 45, 0, 30, 0, 30, 0]))
        flight_time = randint(30, 1050)
        # take only 2021-06-29 11:00 from 2021-06-29 11:00:00.977900
        flight_date = str(flight_date)[-10].replace("-", "/")
        plane_id = randint(1, MAX_PLANE_ID)
        src_airport_id = randint(1, PLAIN_COUNT)
        dst_airport_id = src_airport_id
        while dst_airport_id == src_airport_id:
            dst_airport_id = randint(1, PLAIN_COUNT)
        to_print = f"INSERT INTO FLIGHT(flightid, takeoffdatetime, flighttime, planeid, srcairportid, dstairportid) values ({id},to_date('{flight_date}', 'YYYY/MM/DD HH24:MI'),{flight_time},{plane_id},{src_airport_id},{dst_airport_id});\n"
        flight_sql_file.write(to_print)

if __name__ == "__main__":
    main()
```

**v. Generate Planes Seats:**

```

# gen all seats in all planes
def genSeats():
    with open("gen_planes.sql", "r") as planes:
        f = open("gen_PlaneSeat2.sql", "w")
        # Assuming each plane has 7 seats in a row:
        COLS = 7
        seatID = 1
        data = ""
        cs = ["T"] * 5 + ["B"] * 3 + ["F"]
        lr8 = [*list(range(1, 8))]
        win = [*([1] + [0] * (COLS - 2) + [1])]
        mid = [*([0] * 2 + [1] * (COLS - 4) + [0] * 2)]
        coloms_name = ("seatID", "class", "the_row", "col", "isWithWindow",
            "IsInMiddleRow", "PlaneID")
        t_name = "PlaneSeat"
        head = f"insert into {t_name} ({', '.join(coloms_name)}) values ("
        for plane in planes.read().split("\n"):
            # each line look like:
            # insert into Plane (planeID, type, model, p_size, currWeight, maxW
eight) values (1, 'P', 'anthony27', 496, 496, 496);
            plane_i_data = plane.split(") values ")[1].split(", ")
            planeID = int(plane_i_data[0])
            pi_size = int(plane_i_data[3])
            rows_num = pi_size // COLS
            _class = [random.choice(cs) for _ in range(pi_size)]
            the_row = []
            for ri in range(rows_num):
                the_row += [*[ri] * COLS]
            col = lr8 * rows_num
            isWithWindow = win * rows_num
            IsInMiddleRow = mid * rows_num
            PlaneID = [*[planeID] * COLS * rows_num]
            for i in range(COLS * rows_num): # write to file
                data = f"{head}{seatID}, '{_class[i]}', {the_row[i]}, {col[i]},
{isWithWindow[i]}, {IsInMiddleRow[i]}, {PlaneID[i]}});\n"
                f.write(data)
                if seatID % 10000 == 0:
                    print(seatID)
                    print(data)
                    seatID += 1
            f.close()
if __name__ == "__main__":
    genSeats()

```

**vi. Generate Shuttles Buses:**

```

def times(_):
    flag = True
    while flag:
        fake_start_time = FAKE.time()[:-3]
        t = time.strptime(fake_start_time, "%H:%M")

```

```

hour = t.tm_hour
minute = t.tm_min
start = datetime.datetime(2000, 1, 1, hour=hour, minute=minute)
try: # in case that end date is before start date
    hour = random.randint(1, 23 - hour)
    minute = random.randint(1, 59)
    end = start + datetime.timedelta(hours=hour, minutes=minute)
    flag = False
except:
    pass
# take only HH:MM from datetime object in format YYYY-MM-DD HH:MI:SS
return str(start).split()[1][:-3], str(end).split()[1][:-3]

def genSuttle(n: int):
    r = range(1, n + 1)
    shuttleID = r
    busCount = list(map(lambda a: random.randint(1, 60), r))
    startTime, endTime = list(zip(*map(times, r)))
    description = list(map(lambda a: FAKE.text().replace("\n", " "), r))
    srcAirportID = list(map(lambda a: random.randint(1, n), r))
    dstCityID = list(map(lambda a: random.randint(1, n), r))
    coloms_name = ("shuttleID", "busCount", "startTime", "endTime", "descriptio
n", "dstCityID", "srcAirportID")
    t_name = "ShuttleBusLine"
    data = ""
    for i in r:
        data += (
            f"insert into {t_name} ({','.join(coloms_name)}) values "
            + f"({shuttleID[i - 1]}, '{busCount[i - 1]}', to_date('{startTime[i
- 1]}', 'HH24:MI'), to_date('{endTime[i - 1]}', 'HH24:MI'), '{description[i - 1]
}', {srcAirportID[i - 1]}, {dstCityID[i - 1]});\n")
        with open("gen_Suttle.sql", "w") as f:
            f.write(data)
    genSuttle(4000)

```

## נספח שני: שאלות

Our DB's queries:

1) All flights from 'Martinez'

```

select flight.flightid,
       flight.takeoffdatetime,
       p.model,
       srcP.fullname as "src airport",
       dstP.fullname as "dst airport"
from (((flight natural JOIN Plane p)
      INNER JOIN airport srcP ON srcP.Airportid = flight.srcairportid)
     INNER JOIN airport dstP ON dstP.Airportid = flight.dstairportid)
where srcP.Fullname = 'Martinez';

```

## 2) All airport in Brownborough's City

```

Select airportid, fullname, ABBREVIATION
from airport natural join city
where city.name = 'Port Donald'
order by fullname;

```

## 3) Shuttles buses lines count per airport

```

Select airportid, fullname, sum(buscount) AS "Shuttles buses lines count"
from Shuttlebusline inner join airport on srcairportid = airportid
group by airportid, fullname;

```

4) get count of seats in plane and how much with windows,  
if seat is with window - planeSeat.isWithWindow val is 1, else - 0

```

select planeid, model, p_size as "seats count",
       sum(isWithWindow) as "seats with windows"
from plane natural join planeSeat
group by planeid, model, p_size
order by fullname;

```

## 5) get takeOffDateTime from given src &amp; dst airports name:

```

select flightID, model, takeOffDateTime
from (((flight natural JOIN Plane p)
      INNER JOIN airport srcP ON srcP.Airportid = flight.srcairportid)
      INNER JOIN airport dstP ON dstP.Airportid = flight.dstairportid)
where srcP.fullname = 'Duffy' and dstP.fullname = 'Banks';

```

## 6) for each plane, get how much seats in first, Business and Tourist classes

```

select * from (
  select planeid, model, p_size
  from plane) T1
natural join (
  select planeid, count(*) as "Tourist class"
  from planeSeat
  where class = 'T'
  group by planeid) T2
natural join (
  select planeid, count(*) as "Business class"
  from planeSeat
  where class = 'B'
  group by planeid) T3
natural join (
  select planeid, count(*) as "First Class"
  from planeSeat
  where class = 'F'
  group by planeid) T4;

```

## 7) get all planes which the actual weight passing the max allowed weight

```

select planeid, model, currWeight, maxWeight

```

```
from plane
where currWeight > maxWeight;
```

### 8) get a full path of Shuttlebusline from city to final city

```
create or replace view citiesID2(srcID, dstID) as
select airport.cityID, Shuttlebusline.dstCityID
from Shuttlebusline join airport on airport.airportID = Shuttlebusline.srcAirportID;
with citiesID(srcID, dstID)
as (select airport.cityID, Shuttlebusline.dstCityID
    from Shuttlebusline join airport on airport.airportID = Shuttlebusline.srcAirportID
    union all
    select C1.srcID, C2.dstID
    from citiesID C1 join citiesID2 C2 on C1.dstID != C2.srcID)
select cSrc.name, cDst.name
from ((citiesID JOIN City cSrc ON cSrc.cityID = citiesID.srcID)
    JOIN City cDst ON cDst.cityID = citiesID.dstID);
```

### Queries with all DBs:

#### 1) get for each flight how much tickets sold

```
select flightid, count(*)
from flightticket natural join flight natural join Plane
where status = 'S'
group by flightID;
```

#### 2) get in each airport the employee that service most

```
select E.employeeID, ep.fullname, Airport.fullname, E.servicesCount
from Airport
    natural join Employee ep
    natural join ( select employeeID, count(*) as servicesCount
        from CustomerService
        group by employeeID
        order by servicesCount OFFSET 0 ROWS FETCH NEXT 1 ROWS ONLY
    ) E;
```

#### 3) get the top 40 flights with most customerService calls

```
select flightID, srcA.fullname, dstA.fullname, count(*) as servicesCount
from CustomerService natural join (
    (flight inner join airport srcA on flight.srcairportid = srcA.airportID
    )
    inner join airport dstA on flight.dstairportid = dstA.airportID)
group by flightID
order by servicesCount OFFSET 0 ROWS FETCH NEXT 40 ROWS ONLY;
```



## נספח שלישי: Views

```
-- The Views:
-- get src & dst shuttle bus's cities
-- used in the recursive with query

create or replace view citiesID2(srcID, dstID) as
select airport.cityID, Shuttlebusline.dstCityID
from Shuttlebusline join airport on airport.airportID = Shuttlebusline.srcAirportID;
-- view of services

create or replace view services(employeeID, flightid, servicesCount) as
select employeeID, flightid, count(*) as servicesCount
from CustomerService
group by employeeID;
-- data about flight and it src & dst airports

create or replace view FlightsCitiesNames
(flightid, model, takeoffdatetime, srcFullname, dstFullname) AS
select flight.flightid, p.model, flight.takeoffdatetime, srcP.fullname, dstP.fullname
from (((flight natural JOIN Plane p)
      INNER JOIN airport srcP ON srcP.Airportid = flight.srcairportid)
      INNER JOIN airport dstP ON dstP.Airportid = flight.dstairportid);

-- update the following queries to use it:
-- All flights from 'Martinez'

select *
from FlightsCitiesNames
where srcFullname = 'Martinez';
-- get takeoffdatetime from given src&dst airports name:

select flightid, model, takeoffdatetime
from FlightsCitiesNames
where srcFullname = 'Duffy' and dstFullname = 'Banks';
```

**נספח רביעי: פונקציות ופרוצדורות**

```

1) add delayHour for all flight's takeoffDateTime from given airport id to
create or replace procedure AddDelay2flight(delayHour in Int, flight_id in int)
is begin
update flight
set takeoffDateTime = takeoffDateTime + delayHour / 24
where flightID = flight_id;
end AddDelay2flight;

2) update all the Shuttlebusline's description in the given state to start
with the state name.
create or replace procedure AddStateName(stateName in varchar) is begin
update Shuttlebusline
set description = concat(concat(concat('State: ',stateName),'. Description: ')
,description)
where srcAirportID IN (select airportID
from ((Airport inner join City on Airport.Cityid = City.Cityid)
inner join State on City.stateID = state.stateid)
where State.name = stateName);
end AddStateName;

3) get the next coming time flight from given airport
create or replace function GetNextFlight(airport_id in int) return date is take
overtime date;
begin
select min(takeoffdatetime) into takeovertime
from flight
join airport on srcAirportID = airportID
where airportID = airport_id
and takeoffdatetime > ALL (select sysdate from dual);
return(takeovertime);
end GetNextFlight;

4) return if flight exists and take of time of flight from given src city to gi
ven dst city
create or replace function IsHasFlight(
srcCityName in varchar, dstCityName in varchar, takeOfTime out date)
return varchar is
flightExists varchar(5) := 'false';
begin
select 'true' into flightExists
from FlightsCitiesNames
where srcCityName = srcFullname and dstCityName = dstFullname;
select takeoffdatetime into takeOfTime
from FlightsCitiesNames
where srcCityName = srcFullname and dstCityName = dstFullname;
return(flightExists);
end IsHasFlight;

```