דוגמא לשאלת ייצוג נתונים - חברת תעופה

דרישות המערכת

- החברה מפעילה טיסות מיעדים שונים ברחבי העולם, ליעדים שונים
- כל דגם מטוס דורש אנשי צוות אחרים - לעיתים אין מהנדס טיסה, לעיתים צריך יותר דיילים
- אנשי הצוות הם בתפקידים שונים: טייסים, דיילים וכדומה.
- לטייסים יש הסמכות שונות והם יכולים להטיס רק מטוסים מסוג שהוסמכו לו
- יש לעקוב אחרי סכום שעות הטיסה של טייס היסטורית וברמה שנתית
- החברה מפעילה טיסות נוסעים, כמו כן גם טיסות אשר מכילות מטען ומיועדות לשילוח
- חלק מהטיסות הן טיסות המשך

1. כיתבו קוד ליצירת הטבלאות הנדרשות. הקפידו על נרמול.
2. פרטו נמקו את כל ההחלטות העיצוביות שקיבלתם:
   a. ציינו מה כל הישויות שאתם מייצגים, מה התרחיש בשבילו הם נדרשים, ומה היצוג שנבחר. אם יש ישויות רלוונטיות שאתם לא מייצגים ציינו והסבירו למה.
   b. הסבירו את השדות שבחרתם, התרחיש לו הם נדרשים, והטיפוס שבחרתם להם.
   c. נמקו את המפתחות שבחרתם, האם הם טבעיים (קיימים בעולם האמיתי) או פנימיים (לדוגמה auto increment)
   d. ציינו מה היחסים בין הישויות (1:1, n:1, או n:n) בעולם האמיתי. אם בחרתם ליצג יחס אחר, ציינו ונמקו. מספיק לציין יחסים רק בין ישויות שכנות, שיש בניהן קשר ישיר.
   e. פרטו מה ה constraints (לדוגמה: unique, FK, not null, check) שבחרתם ומה ההגנה שהם מספקים. אם יש נקודות שלא מוגנות על ידי בסיס הנתונים ציינו אותן ונמקו איך להגן.
3. מה האופציות ליצג אנשי צוות כללים וטייסים. פרטו יתרונות וחסרונות ונמקו את בחירתכם.
4. יש מטוסים היעודים למשלוח וכדומה ויש כאלו שיש להם כמה יעודים. כיצד ניתן ליצג את ההתאמה? פרטו יתרונות וחסרונות ונמקו את בחירתכם.
5. כיצד איזנתם את הצורך להתאים את אנשי הצוות למטוס לבין שליפה נוח של מקרים כמו כל הדיילים בטיסה מסויימת. נמקו את בחירתכם.
6. שילפו "פיספוסים שעומדים לקרות" - טייסים ששובצו לטיסה עתידית הנמצאת ביעד הרחוק ביותר שעות טיסה מהנדרש כדי להגיע מטיסה קודמת שלו. לדוגמה, טייס שהגיע לבנקוק וכעבור שעה יש לו טיסה מניו יורק.
7. שילפו יחד את כל היעדים שניתן להגיע אליהם בטיסה אחת, טיסה עם עצירה אחת, טיסה עם שתי עצירות

תשובה

.1

```
   CREATE TABLE Countries (
   country_id INT AUTO_INCREMENT PRIMARY KEY,
   country_name VARCHAR(100) NOT NULL,
   iso_code CHAR(2) NOT NULL UNIQUE
);


CREATE TABLE Airports (
   airport_id INT AUTO_INCREMENT PRIMARY KEY,
   country_id INT NOT NULL,
   airport_name VARCHAR(120) NOT NULL,
   city_name VARCHAR(100) NOT NULL,
   iata_code CHAR(3) NOT NULL UNIQUE,   #jfk, tlv
   FOREIGN KEY (country_id) REFERENCES Countries (country_id)
);


CREATE TABLE AircraftModels (
   model_id INT AUTO_INCREMENT PRIMARY KEY,
   model_name VARCHAR(100) NOT NULL,
   manufacturer VARCHAR(100),
   capacity INT NOT NULL CHECK (capacity > 0),
   cruise_speed_kmh INT NOT NULL CHECK (cruise_speed_kmh > 0)
);

CREATE TABLE Aircrafts (
   aircraft_id INT AUTO_INCREMENT PRIMARY KEY,
   model_id INT NOT NULL,
   registration_code VARCHAR(20) NOT NULL UNIQUE,
   FOREIGN KEY (model_id) REFERENCES AircraftModels(model_id)
);

CREATE TABLE AircraftPurposes (
   purpose_id INT AUTO_INCREMENT PRIMARY KEY,
   purpose_name VARCHAR(50) NOT NULL  #Passenger, Cargo, Mixed ect.
);

CREATE TABLE AircraftPurposeAssignment (
   aircraft_id INT NOT NULL,
   purpose_id INT NOT NULL,
```

```
    PRIMARY KEY (aircraft_id, purpose_id),
    FOREIGN KEY (aircraft_id) REFERENCES Aircrafts(aircraft_id),
    FOREIGN KEY (purpose_id) REFERENCES AircraftPurposes(purpose_id)
);


CREATE TABLE CrewRoles (
    role_id INT AUTO_INCREMENT PRIMARY KEY,
    role_name VARCHAR(50) NOT NULL      #Pilot, Flight Attendant, Flight Engineer...
);

CREATE TABLE CrewMembers (
    crew_member_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    role_id INT NOT NULL,
    FOREIGN KEY (role_id) REFERENCES CrewRoles(role_id)
);

CREATE TABLE PilotCertifications (
    crew_member_id INT NOT NULL,
    model_id INT NOT NULL,
    PRIMARY KEY (crew_member_id, model_id),
    FOREIGN KEY (crew_member_id) REFERENCES CrewMembers(crew_member_id),
    FOREIGN KEY (model_id) REFERENCES AircraftModels(model_id)
);


CREATE TABLE Routes (
    origin_airport_id INT NOT NULL,
    destination_airport_id INT NOT NULL,
    distance_km INT NOT NULL CHECK (distance_km > 0),
    PRIMARY KEY (origin_airport_id, destination_airport_id),
    FOREIGN KEY (origin_airport_id) REFERENCES Airports(airport_id),
    FOREIGN KEY (destination_airport_id) REFERENCES Airports(airport_id)
);


CREATE TABLE Flights (
    flight_id INT AUTO_INCREMENT PRIMARY KEY,
    aircraft_id INT NOT NULL,
    origin_airport_id INT NOT NULL,
    destination_airport_id INT NOT NULL,
    departure_time DATETIME NOT NULL,
```

```
    arrival_time DATETIME NOT NULL,
     continuation_flight_id INT NULL,
    FOREIGN KEY (aircraft_id) REFERENCES Aircrafts(aircraft_id),
    FOREIGN KEY (origin_airport_id) REFERENCES Airports(airport_id),
    FOREIGN KEY (destination_airport_id) REFERENCES Airports(airport_id),
    FOREIGN KEY (continuation_flight_id) REFERENCES Flights(flight_id),
    CHECK (arrival_time > departure_time)
);

CREATE TABLE FlightCrew (
    flight_id INT NOT NULL,
    crew_member_id INT NOT NULL,
    role_id INT NOT NULL,
    PRIMARY KEY (flight_id, crew_member_id),
    FOREIGN KEY (flight_id) REFERENCES Flights(flight_id),
    FOREIGN KEY (crew_member_id) REFERENCES CrewMembers(crew_member_id),
    FOREIGN KEY (role_id) REFERENCES CrewRoles (role_id)
);
```

.2

א. **ישויות:**

1. **Airports**– מייצגת כל יעד בעולם (עיר, מדינה, קוד שדה תעופה).
   נדרשת כדי לתאר נקודת מוצא ויעד לכל טיסה.
2. **Routs**– מייצגת את המרחק בין כל שני יעדים.
3. **AircraftModels** – סוגי המטוסים, משפיע על גודל הצוות הנדרש.
4. **Aircrafts** – מטוסים בפועל.
5. **CrewRoles** – סוגי תפקידים בצוות (טייס, דייל, מהנדס).
6. **CrewMembers** – כל אנשי הצוות בחברה.
7. **PilotCertifications** – התאמות בין טייסים לדגמי מטוסים.
8. **Flights** – טיסות בפועל, כולל טיסות המשך.
9. **FlightCrew** – שיבוץ צוות לטיסות.
10. **PilotFlightHours** – היסטוריית שעות טיסה.
11. **AircraftPurposes** – סוגי ייעוד (נוסעים, מטען, משולב).
12. **AircraftPurposeAssignment** – התאמת מטוסים לייעודים.

לא ייצגנו את היישות cities.

לא ייצגנו אותה כי היא לא נדרשת לתרחיש העסקי של חברת תעופה. המערכת מתייחסת לטיסות בין שדות תעופה, לא בין ערים. גם אם לעיר יש כמה שדות תעופה, זה חשוב לנו בדיוק כי לכל אחד יש מזהה ייחודי (airport_id, iata_code). העיר עצמה משמשת רק כמידע תיאורי, ולכן מספיק לשמור אותה כתכונה בתוך Airports.

ב.

- כל המזהים (IDs) הם מזהים פנימיים. כלומר, אין להם משמעות בעולם האמיתי אלא הם משמשים רק לניהול שלמות הנתונים בבסיס הנתונים. לכן בחרתי להשתמש ב־INT AUTO_INCREMENT, שמאפשר יצירת מזהים חד־חד ערכיים באופן אוטומטי, מקל על השימוש בהם כמפתחות זרים ומונע כפילויות.
- שמות (כמו שם מדינה, שם שדה תעופה, שם דגם מטוס, שמות אנשי צוות ותפקידי צוות) אינם בעלי אורך קבוע, ויכולים להכיל מספר תווים משתנה, כלומר יש מגוון שמות אפשריים. לכן השתמשתי ב־VARCHAR באורך מתאים, כדי לאפשר גמישות ולחסוך מקום.
- קודים תקניים של שדות תעופה (IATA בן 3 תווים) הם בעלי אורך קבוע ומוגדר על פי תקן עולמי. לכן יושמו כ - CHAR(3). זה מבטיח שהקוד תמיד יהיה באורך הנכון ומונע טעויות לוגיות.
- תאריכים ושעות טיסה (שדות המראה ונחיתה) חייבים לשמור גם תאריך וגם שעה, כדי שנוכל לחשב הפרשי זמנים ולבדוק רצף טיסות. לכן השתמשתי בטיפוס DATETIME, שהוא המתאים ביותר לתרחיש הזה.
- ערכים כמותיים כמו קיבולת מטוס (capacity), מרחק נתיב (distance_km) ומהירות שיוט (cruise_speed_kmh) הם ערכים מספריים שלמים. לכן בחרתי בטיפוס INT, יחד עם הגבלות CHECK על ערכים חיוביים, כדי לוודא תקינות לוגית של הנתונים.

ג.

בסכמה שעיצבתי, בכל הישויות הראשיות השתמשתי במזהים פנימיים כמפתח ראשי (INT AUTO_INCREMENT). בחרתי במפתחות פנימיים משום שאין לנו מזהה טבעי יציב שנשאר קבוע לאורך זמן עבור רוב הישויות. לדוגמה, מטוס יכול להחליף מספר זנב, או ששמות מדינות ושדות תעופה עשויים להשתנות, ולכן עדיף להשתמש במזהה פנימי שאינו תלוי בעולם האמיתי. הדבר מקל מאוד על יצירת קשרים בין הטבלאות, כי תמיד יש מפתח קצר, חד־חד ערכי ופשוט לשימוש כמפתח זר.

בטבלאות קשר רבות־לרבות (N:N) כמו PilotCertifications (קישור בין טייס לדגם מטוס), FlightCrew (שיבוץ אנשי צוות לטיסות) ו־AircraftPurposeAssignment (שיוך מטוס לייעודים שונים), בחרתי במפתחות מורכבים. כלומר, המפתח הראשי מורכב משני המזהים הזרים יחד. זה מבטיח שאי אפשר יהיה להזין פעמיים את אותו קשר, ובכך שומר על שלמות הנתונים.

לבסוף, קיימים גם מפתחות טבעיים בעלי משמעות בעולם האמיתי, כמו airport_code (קוד IATA של שדה תעופה). בחרתי שלא להשתמש בהם כמפתח ראשי אלא להגדירם כשדות ייחודיים (UNIQUE). הסיבה היא שמזהה טבעי עלול להשתנות במציאות (למשל אם רשות בינלאומית משנה קוד שדה תעופה), וכאשר משתמשים בו כמפתח ראשי הדבר יוצר בעיות חמורות של עקביות ושלמות נתונים. לעומת זאת, שמירתו כשדה ייחודי מאפשרת גם לוודא שאין כפילויות וגם לשמור על יציבות הקשרים בבסיס הנתונים.

ד.

- מדינה ↔ ערים: יחיד לרבים (מדינה אחת כוללת ערים רבות, כל עיר שייכת למדינה אחת).
- עיר ↔ שדות תעופה: יחיד לרבים (עיר אחת כוללת שדות תעופה רבים, כל שדה תעופה שייך לעיר אחת).
- דגם מטוס ↔ מטוסים: יחיד לרבים (דגם מטוס אחד מתאים להרבה מטוסים, כל מטוס שייך לדגם אחד).
- מטוסים ↔ ייעודי מטוס (באמצעות AircraftPurposeAssignment): רבים לרבים (מטוס יכול לשמש למספר ייעודים, וייעוד יכול להיות משויך למספר מטוסים).
- תפקידי צוות ↔ אנשי צוות: יחיד לרבים (תפקיד אחד יכול להיות משויך להרבה אנשי צוות, לכל איש צוות יש תפקיד אחד).
- טייסים ↔ דגמי מטוס (באמצעות PilotCertifications): רבים לרבים (טייס יכול להיות מוסמך לכמה דגמים, ודגם יכול להיות מוסמך על־ידי כמה טייסים).
- נתיבים ↔ שדות תעופה: יחיד לרבים לכל צד (כל נתיב מוגדר על ידי שדה מוצא יחיד ושדה יעד יחיד, שדה תעופה יכול להופיע בהרבה נתיבים).
- טיסות ↔ מטוסים: יחיד לרבים (טיסה אחת משתמשת במטוס אחד, מטוס יכול לשמש להרבה טיסות).
- טיסות ↔ נתיבים: יחיד לרבים (טיסה אחת שייכת לנתיב אחד, כל נתיב יכול להיות מקושר להרבה טיסות).
- טיסות ↔ טיסות (טיסת המשך): יחיד ליחיד אופציונלי (טיסה אחת יכולה להצביע על טיסת המשך אחת).
- טיסות ↔ אנשי צוות (באמצעות FlightCrew): רבים לרבים (טיסה כוללת הרבה אנשי צוות, ואיש צוות יכול להשתתף בהרבה טיסות).
- אנשי צוות ↔ תפקידי צוות (בטבלת FlightCrew): יחיד לרבים (הקצאה משויכת תמיד לתפקיד אחד, תפקיד יכול להופיע בהקצאות רבות).

בסכמה שעיצבתי קיימים כמה סוגי יחסים בין הישויותֿ. לכל טיסה יש שדה תעופה מוצא ושדה תעופה יעד, ולכן היחס בין טיסות לשדות תעופה הוא של רבים־לאחד – טיסה אחת שייכת לשדה תעופה אחד כמוצא ולשדה תעופה אחד כיעד, אבל כל שדה תעופה יכול להופיע כמוצא או יעד עבור טיסות רבות. בנוסף, לכל טיסה משויך מטוס אחד ספציפי, אך כל מטוס יכול להפעיל טיסות רבות לאורך זמן – גם כאן מדובר ביחס של רבים־לאחד.

אנשי הצוות משויכים לתפקידים, כך שכל איש צוות מחזיק בתפקיד אחד (טייס, דייל, מהנדס וכדומה), אך כל תפקיד יכול להכיל אנשים רבים. זהו יחס של רבים־לאחד.

בחלק מהמקרים מדובר ביחסים רבים־לרבים. כך למשל, טייסים יכולים להיות מוסמכים להטיס יותר מדגם מטוס אחד, וכל דגם מטוס מוסמך על ידי טייסים רבים – הקשר הזה מיוצג בטבלת PilotCertifications. באופן דומה, כל טיסה כוללת צוות של כמה אנשי צוות, וכל איש צוות יכול להשתבץ למספר טיסות שונות – זהו קשר רבים־לרבים המיוצג בטבלת FlightCrew. גם הקשר בין מטוסים לייעודים (נוסעים, מטען, משולב) הוא רבים־לרבים, משום שמטוס יכול להיות מותאם למספר ייעודים שונים וכל ייעוד חל על מטוסים רבים.

לבסוף, קיימת גם אפשרות של קשר אחד־לאחד בין טיסות שונות, במקרה שבו טיסה מוגדרת כטיסת המשך של טיסה אחרת. כאן מדובר בקשר ייחודי ואופציונלי שמאפשר לחבר בין שתי טיסות רצופות.

ה.

השתמשתי במגבלות (constraints) כדי להבטיח שלמות לוגית ותקינות הנתונים. כך למשל, בכל השדות החיוניים – כמו שם מדינה, שם שדה תעופה, זמני המראה ונחיתה של טיסות או שמות אנשי צוות – הוגדר NOT NULL, כדי למנוע מצב שבו מוזנת רשומה חלקית בלי נתונים הכרחיים.

כדי למנוע כפילויות, הוגדר UNIQUE במקומות שבהם הערך חייב להיות חד־חד ערכי בעולם האמיתי. כך לדוגמה, קוד IATA של שדה תעופה הוא ייחודי ולכן שדה זה מוגדר כייחודי, וגם מספר רישום של מטוס לא יכול להופיע פעמיים.

בנוסף, נעשה שימוש ב-CHECK על ערכים מספריים כדי לשמור על תקינותם. כך למשל, בשדות כמו קיבולת מטוס, מהירות שיוט או מרחק נתיב הוגדר תנאי שמבטיח שהערך חייב להיות גדול מאפס. באופן דומה, לזמן תפעולי מינימלי של מטוס הוגדר תנאי שלא יאפשר ערך שלילי.

המפתחות הזרים (FOREIGN KEY) מחברים בין הישויות ומבטיחים שלמות בין הטבלאות. לדוגמה, כל טיסה חייבת להפנות לשדה תעופה קיים כמוצא וכיעד, וכל איש צוות שמופיע בטבלת השיבוצים חייב להיות קיים בטבלת אנשי הצוות.

בטבלאות קשר רבים לרבים, כמו PilotCertifications או FlightCrew, הוגדר מפתח ראשי מורכב שמורכב משני המזהים הזרים. הדבר מבטיח שלא תיווצר כפילות, למשל שלא ניתן יהיה לשייך את אותו טייס לאותו דגם פעמיים או לשבץ את אותו איש צוות פעמיים לאותה טיסה.

לבסוף, ישנם כללים עסקיים שלא מוטמעים ישירות באמצעות constraints בבסיס הנתונים, למשל מניעת מצב שבו טייס משובץ לשתי טיסות חופפות בזמן.

.3

אופציה 1 – טבלה אחת לכולם (עם שדה role_id)

- יתרונות: אין כפילות נתונים, קל להוסיף/לשנות תפקידים.
- חסרונות: מצריך סינון ב־JOIN לשליפת טייסים בלבד.

אופציה 2 – שתי טבלאות נפרדות (Pilots, OtherCrew)

- יתרונות: פשטות בשליפות ייעודיות.
- חסרונות: כפילות מבנה, תחזוקה קשה יותר, קושי בשיבוץ משותף.

בחרתי באופציה של טבלה אחת לכלל אנשי הצוות (CrewMembers) עם שדה role_id שמפנה ל-CrewRoles. את המידע הייחודי לטייסים אינני שומר בתוך CrewMembers, אלא מייצג בטבלאות אחרות שכבר קיימות בסכמה: ההסמכות של טייסים לדגמי מטוסים נשמרות ב-PilotCertifications (קשר רבים-לרבים בין crew_id ל-model_id, עם מפתח ראשי מורכב המונע כפילות), והשיבוץ בפועל של טייסים לטיסות נשמר ב-FlightCrew (רבים-לרבים בין אנשי צוות לטיסות). השילוב בין FlightCrew ל-Flights מאפשר לגזור בכל רגע נתון את סך

שעות הטיסה ההיסטוריות או השנתיות של כל טייס, בלי לשמור עותק נגזר ולא עקבי. בנוסף, ההבחנה הטיפוסית בין "טייס" לבין שאר אנשי הצוות נשענת על CrewRoles, כך שאין צורך בשדות ייעודיים שיישארו ריקים עבור דיילים או מהנדסי טיסה. באופן זה אנו שומרים על נרמול: המידע הייחודי לטייסים מיוצג במקום הנכון (PilotCertifications לשאלת "על אילו דגמים מותר לו להטיס", ו-FlightCrew/Flights להיסטוריית הטסה בפועל), בעוד CrewMembers נותרת טבלה כללית, נקייה מכפילות ומערכים חסרים עבור תפקידים שאינם טייסים.

.4

אופציה 1 – שדה יחיד ב־Aircrafts

יתרונות: שליפה פשוטה – נוח אם לכל מטוס יש ייעוד אחד בלבד (למשל, מטוס נוסעים בלבד).

חסרונות: בעולם האמיתי זה כמעט לא מתקיים – רוב החברות רוצות גמישות תפעולית. אותו מטוס יכול לשמש היום לטיסת נוסעים, מחר לטיסת מטען, ולעיתים גם למטרות מעורבות (נוסעים + מטען). שדה יחיד לא מאפשר לייצג מציאות כזו.

אופציה 2 – טבלת קשר (AircraftPurposeAssignment) N:N

- יתרונות: מאפשרת לשייך למטוס כמה ייעודים בו-זמנית, כפי שקורה בפועל בחברות תעופה. לדוגמה, מטוס דרימליינר יכול להיות רשום גם כ"נוסעים" וגם כ"צ'ארטר", לפי חוזים ומדיניות החברה. בעולם האמיתי שינוי ייעוד כזה נעשה לעיתים קרובות – לכן חשוב לשמור את הגמישות.
- חסרונות: נדרשת פעולת JOIN נוספת בשאילתות – אבל זה חיסרון טכני קטן יחסית לעומת היתרון של ייצוג נכון של המציאות.

בחירה: טבלת קשר (AircraftPurposeAssignment).

הסיבה: במציאות התפעולית של חברות תעופה, מטוסים עוברים שימושים שונים לאורך חייהם ולעיתים גם במקביל. ייצוג הקשרים בטבלה ייעודית משקף נאמנה את הצורך הזה, ומונע מצב שבו המודל מגביל את החברה לייעוד אחד בלבד – דבר שאינו ריאלי.

.5

האיזון הושג באמצעות בנייה של מבנה נתונים דו־שלבי. בשלב הראשון, נעזרת טבלת **PilotCertifications** כדי להגדיר באופן חד־משמעי אילו טייסים מוסמכים להטיס אילו סוגי מטוסים. המשמעות היא שלא ניתן לשבץ טייס לטיסה אם אין לו הסמכה מתאימה, ובכך נמנעת בפועל האפשרות לטעויות קריטיות כמו טיסה בלי טייס כשיר, או לחלופין מצב אבסורדי שבו משובצים יותר מדי טייסים לאותה טיסה. המבנה הזה משמש למעשה כהגנה רגולטורית, ומחייב שכל שיבוץ יעבור דרך בדיקת ההתאמות הנדרשות.

בשלב השני, נעזרת טבלת **FlightCrew** כדי לשקף את ההרכב בפועל של הצוות בכל טיסה נתונה – הן טייסים והן דיילים. הטבלה הזו מתעדכנת בהתאם לשיבוץ בזמן אמת, ומאפשרת שליפה פשוטה וגמישה של מידע תפעולי, למשל כל הדיילים ששובצו לטיסה מסוימת, או כל אנשי הצוות שהוקצו לקו טיסות מסוים. היתרון כאן הוא ביעילות של ניתוח ושאילתות, מבלי לפגוע בהגבלות שהוטמעו מראש דרך טבלת ההסמכות.

בכך מושג איזון כפול: מצד אחד, נשמרת הקפדה על התאמה בין הצוות למטוס ברמה הבטיחותית והמקצועית, ומצד שני נשמרת יכולת תפעולית לנהל ולשלוף מידע על הצוותים בצורה נוחה ומהירה. כלומר, כל שיבוץ בפועל

עובר בדיקה מול ההסמכות, והטבלאות יחד מאפשרות גם שמירה על כללי בטיחות והגנות מפני שיבוצים לא חוקיים וגם יעילות תפעולית בשגרה.

כדי למנוע בפועל שיבוץ של טיסה ללא טייס כלל, או לחלופין מצב של ריבוי טייסים ראשיים באותה טיסה, יש להיעזר בטריגרים במסד הנתונים. טריגר כזה יופעל בעת הכנסת רשומה חדשה לטבלת FlightCrew, ויבדוק האם כבר קיים טייס ראשי אחד לפחות בטיסה – כך שלא תישאר טיסה ללא טייס. במקביל הטריגר יוודא שאין יותר ממספר מקסימלי של טייסים ראשיים (למשל אחד או שניים, לפי החלטת החברה) לאותה טיסה. אם התנאים לא מתקיימים, הטריגר ימנע את הכנסת הנתון ויזרוק שגיאה. בדרך זו משלבים בין נרמול הטבלאות לבין מנגנוני שלמות עסקיים שמבטיחים עמידה בכללים התפעוליים הקריטיים.

.6

```
CREATE VIEW PilotFlights AS

SELECT

  fc.crew_member_id AS pilot_id,

  f.flight_id,

  f.origin_airport_id,

  f.destination_airport_id,

  f.departure_time,

  f.arrival_time,

  a.model_id

FROM FlightCrew AS fc

JOIN CrewRoles AS cr ON cr.role_id = fc.role_id AND cr.role_name = 'Pilot'

JOIN Flights AS f ON f.flight_id = fc.flight_id

JOIN Aircrafts AS a ON a.aircraft_id = f.aircraft_id;


# Find "missed connections" using the view once (prev) and again (next)

SELECT

  prev.pilot_id,

  cm.first_name,
```

```
  cm.last_name,

  prev.flight_id AS prev_flight_id,

  ap_prev.iata_code AS prev_arrival_airport,

  prev.arrival_time AS prev_arrival_time,

  next.flight_id AS next_flight_id,

  ap_next.iata_code AS next_origin_airport,

  next.departure_time AS next_departure_time, # Hours available between flights

  ROUND(TIMESTAMPDIFF(MINUTE, prev.arrival_time, next.departure_time) / 60.0, 2) AS
hours_available,

  # Hours required based on route distance and next aircraft model cruise speed

  ROUND(r.distance_km / am.cruise_speed_kmh, 2) AS hours_required, # Shortfall (positive
means “missed connection” risk)

  ROUND((r.distance_km / am.cruise_speed_kmh) - (TIMESTAMPDIFF(MINUTE,
prev.arrival_time, next.departure_time) / 60.0), 2) AS shortfall_hours

FROM PilotFlights AS prev

JOIN PilotFlights AS next ON next.pilot_id = prev.pilot_id

  AND next.departure_time > prev.arrival_time

  # Ensure adjacency: the next flight is the earliest one after prev.arrival_time for this pilot

  AND next.departure_time = (

      SELECT MIN(pf2.departure_time)

      FROM PilotFlights AS pf2

      WHERE pf2.pilot_id = prev.pilot_id

        AND pf2.departure_time > prev.arrival_time

  )

JOIN Routes AS r ON r.origin_airport_id = prev.destination_airport_id AND
r.destination_airport_id = next.origin_airport_id
```

```
JOIN Airports AS ap_prev ON ap_prev.airport_id = prev.destination_airport_id

JOIN Airports AS ap_next ON ap_next.airport_id = next.origin_airport_id

JOIN CrewMembers AS cm ON cm.crew_member_id = prev.pilot_id

JOIN AircraftModels AS am ON am.model_id = next.model_id

WHERE next.departure_time > NOW()

  AND (r.distance_km / am.cruise_speed_kmh) > (TIMESTAMPDIFF(MINUTE, prev.arrival_time,
next.departure_time) / 60.0)

ORDER BY shortfall_hours DESC, next.departure_time ASC;
```

.7

```
CREATE VIEW v_direct AS

SELECT f1.origin_airport_id AS origin_id,

      f1.destination_airport_id AS dest_id

FROM Flights AS f1;


CREATE VIEW v_one_stop AS

SELECT f1.origin_airport_id AS origin_id,

      f2.destination_airport_id AS dest_id

FROM Flights AS f1

JOIN Flights AS f2 ON f1.destination_airport_id=f2.origin_airport_id;
```

```
CREATE VIEW v_two_stops AS
SELECT f1.origin_airport_id AS origin_id,
       f3.destination_airport_id AS dest_id
FROM Flights AS f1
JOIN Flights AS f2 ON f1.destination_airport_id=f2.origin_airport_id
JOIN Flights AS f3 ON f2.destination_airport_id=f3.origin_airport_id;


SELECT
  ap_from.iata_code AS origin_iata,
  ap_to.iata_code AS destination_iata,
  CASE p.min_stops
    WHEN 0 THEN 'Direct flight'
    WHEN 1 THEN 'One stop'
    WHEN 2 THEN 'Two stops'
  END AS connection_type,
  p.min_stops AS stops_count
FROM (
  SELECT origin_id,dest_id,MIN(stops) AS min_stops
  FROM (
    SELECT d.origin_id,d.dest_id,0 AS stops FROM v_direct AS d
    UNION ALL
    SELECT o.origin_id,o.dest_id,1 AS stops FROM v_one_stop AS o
    UNION ALL
    SELECT t.origin_id,t.dest_id,2 AS stops FROM v_two_stops AS t
```

```
  ) AS all_paths

  WHERE origin_id<>dest_id

  GROUP BY origin_id,dest_id

) AS p

JOIN Airports AS ap_from ON ap_from.airport_id=p.origin_id

JOIN Airports AS ap_to ON ap_to.airport_id=p.dest_id

ORDER BY
p.min_stops,ap_from.city_name,ap_from.iata_code,ap_to.city_name,ap_to.iata_code;
```