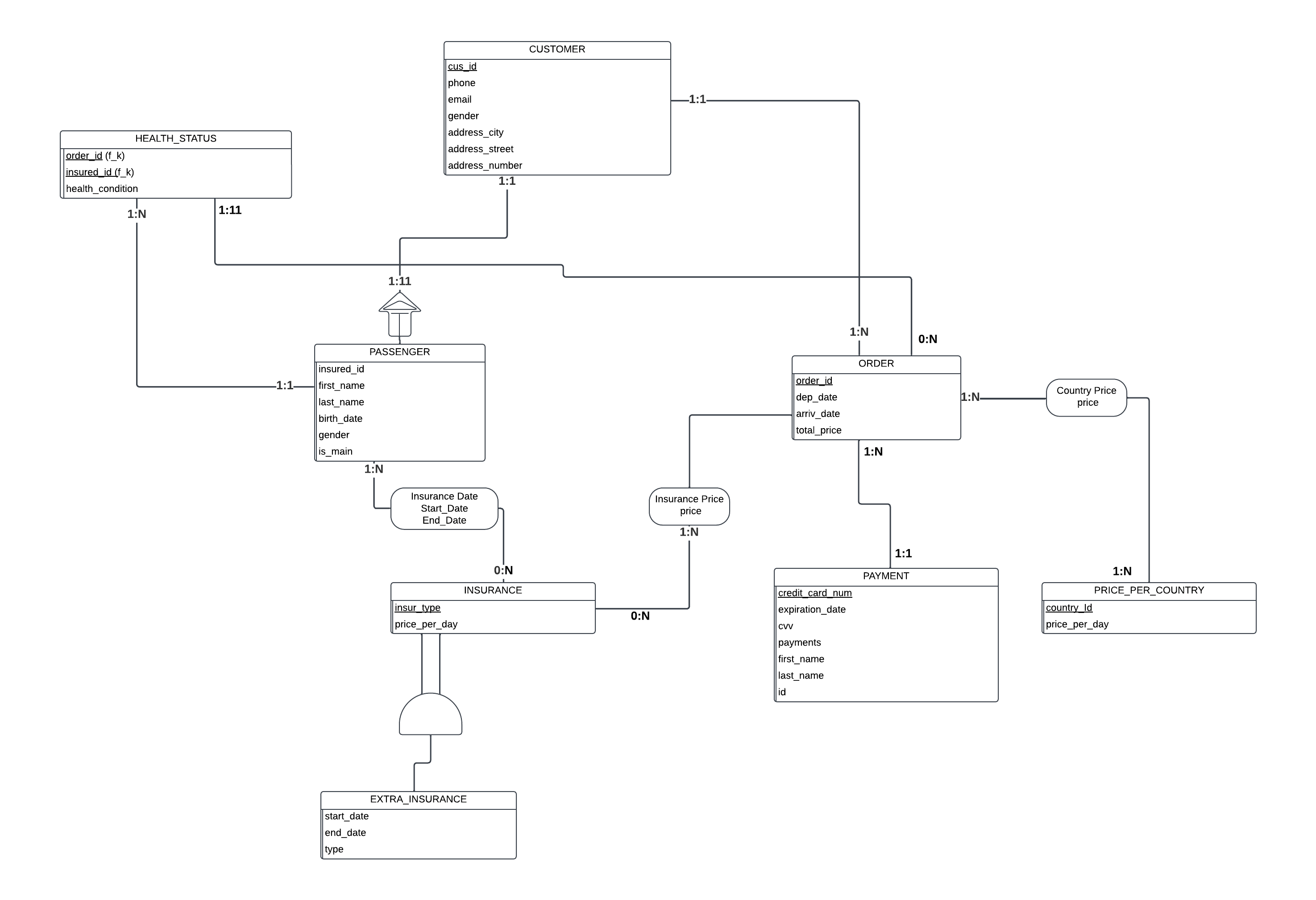
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מס' קבוצה | אתר | | | תאריך הגשה |
| 23 | [www.website.com](http://www.website.com) | | | DD/MM/YYYY |
| חברי הצוות - מספרי ת.ז | | | | |
| 206836686 | | 206947400 | 207975699 | |

סריקה של תרשים הerd ממטלה 2:

מבנה הטבלאות ממטלה 2:

**customers** (cus\_Id, phone, email, gender, address\_city, address\_street, address-number)

**passengers** (cus\_Id (customers), insured\_Id, first\_name, last\_name, Birth\_date, gender, is\_main)

**orders** (order\_Id, depdate, arrivdate, price, cus\_Id (customers), credit\_card\_num (payments))

**prices per country** (Country\_id, Price per day)

**payments** (credit\_card\_num, expiration\_date, CVV, payments, first\_name, last\_name, Id)

**Healthstatus** (order\_Id (orders), {insurance\_id, cus\_Id} (passenger), health\_condition)

**insurances** (insurance\_type, price per day)

**insurance\_dates** (insur\_type (insurances), {insured\_id, cus\_Id} (passenger), start\_date, end\_date)

**insurance\_price**s (insur\_type (insurances), order\_Id (orders), Price)

**extra\_insurance**s (extra\_insurance\_type (insurance), start\_date, end\_date, type)

**countries price** (country\_Id (prices\_per\_country), order Id (orders), price)

הערות ממטלה 2:

מודל ERD:

* יש פה שני תהליכים שונים שאיחדתם לתהליך אחד – רישום לעומת התחברות. הייתם צריכים להפריד אותם כך שכל אחד מתחיל בנקודה ברורה.
* יש פה ערבוב בין צורת רישום של ERD בין צורת רישום של מודל טבלאי, אין דבר כזה ששדות של יישות יהיו מפתח זר בerd לא מובן פה. איך אנחנו מזהים את הצהרות הבריאות.
* אין שמות על כל הקשרים.
* לא הסברתם למה passengers הוא יישות חלשה.
* חסרים שדות להצהרת בריאות, יש כמה שאלות שעונים עליהן.
* המשמעות של הקרדינליות בין health\_status ו-order לא ברורה, איך אפשר שתהיה הצהרת בריאות שמולאה בלי הזמנה? בנוסף, לכל הזמנה ממלאים מחדש הצהרת בריאות, אז גם לא מובן למה אתם מאפשרים לה להיות שייכת לכמה הזמנות.
* גם אחרי ההערה שלכם על country לא מובן מה קורה כאן. אם אנחנו מניחים שלכל מדינה יש תוספת שונה למחיר הפוליסה, מספיקה היישות price per country שיצרתם, ונייצר קשר בינה ליישות policy כדי לדעת אילו מדינות נכללות בכל פוליסה. לא צריך עוד שדה של מחיר.
* אין סימון של מזהה בpassenger.
* התכונה על הקשר insurance price מיותרת. אם אנחנו יודעים את הכיסויים הביטוחיים שיש (insurance) בכל הזמנה, ולכל כיסוי כזה אנחנו יודעים כמה הוא עולה, אז אין צורך לשמור זאת שוב.
* מחיר הביטוח אינו קבוע ותלוי בשאלון הרפואי שכל נוסע ממלא. למשל, ביטוח קיצור נסיעה יקר יותר עבור אנשים מעל גיל 60. המודל שלכם לא מאפשר את הגמישות הזו.
* ההורשה פה צריכה להיות חלקית כי אחרת אתם לא מאפשרים ביטוחים מסוג שאינו erxta.
* השדות strat\end date מופיעים פעמיים גם ביישות extra insurance וגם בקשר insurance date וזה מיותר. שימו לב שאם תאריכים ההתחלה והסיום הם תכונה של extra insurance זה אומר שכל מי שעושה ביטוח ספורט יהיה באותם תאריכים – וזה כמובן לא המצב.

המודל הטבלאי:

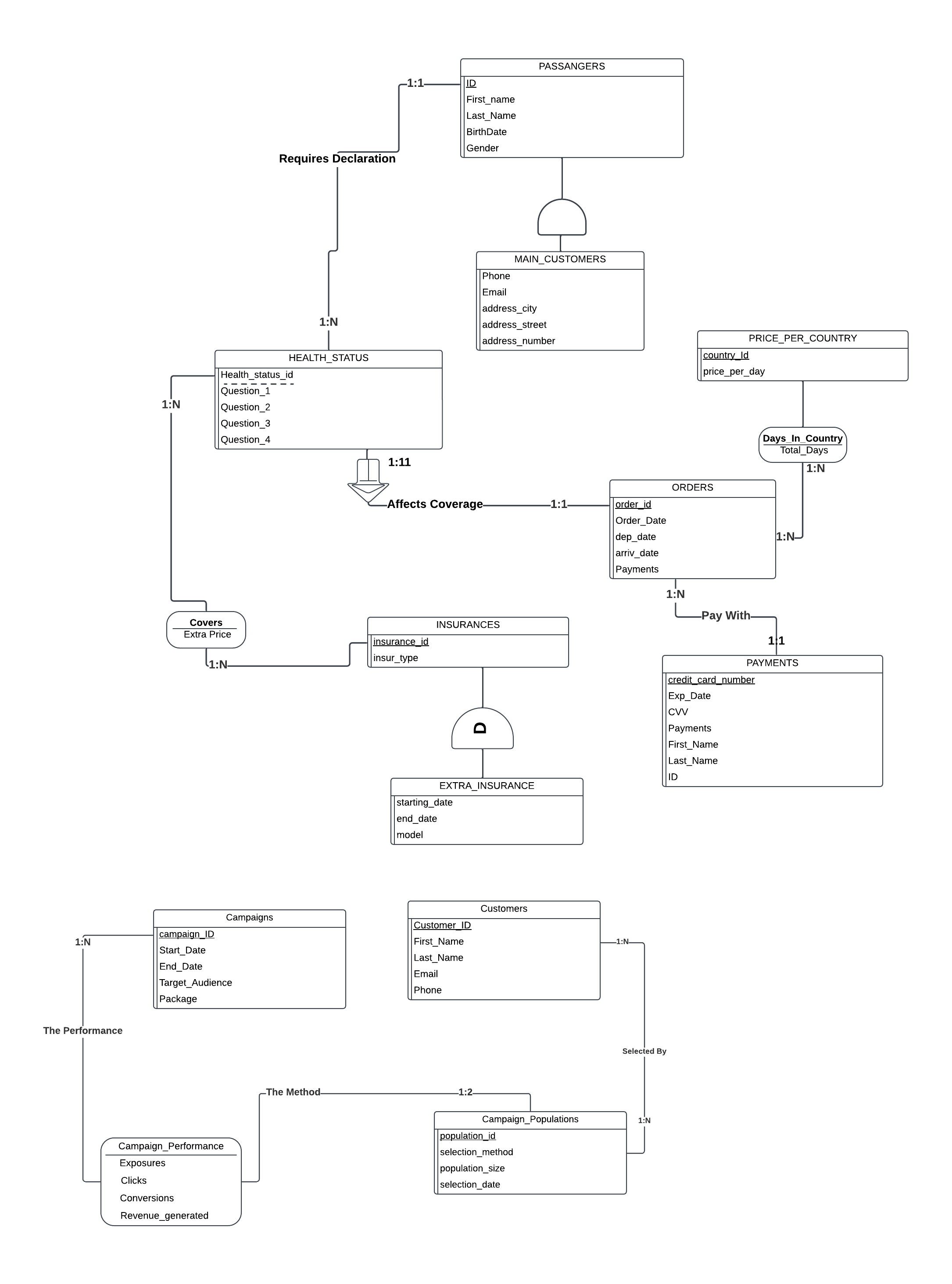
* ב-payments יש לכם שרשרת תלויות של credit\_card\_num -> id -> first\_name, last\_name, וזו הפרת נרמול.
* חסר סימון מפתח ראשי ב – insurance\_prices בשדה insurance\_type.

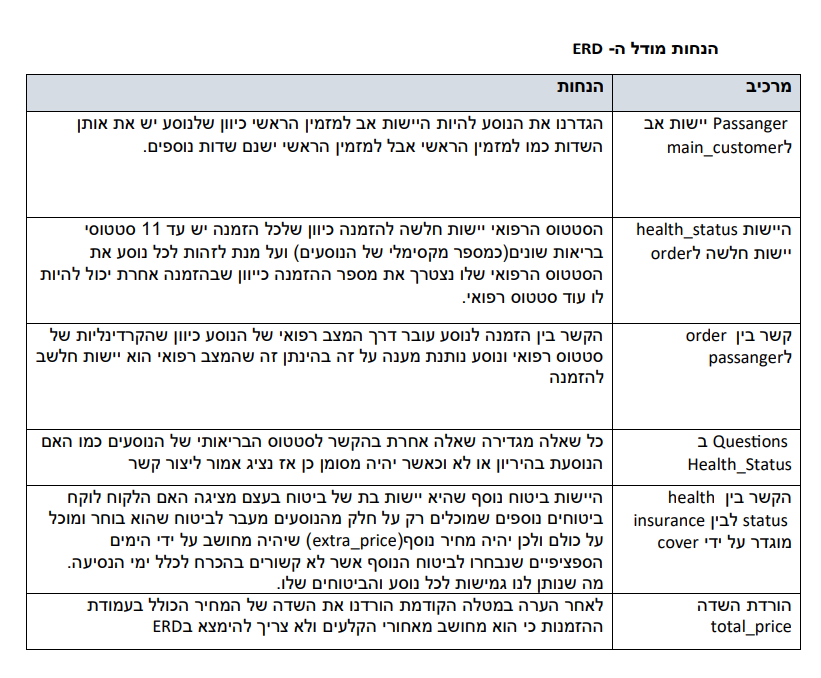
הסברים כלליים על התיקונים שעשינו:

1. שינינו את ישות MAIN\_CUSTOMER כך שיהיה ישות בן של passenger, לפני זה passenger היה ישות חלשה של ישות הmain\_suctomer, ביצענו זאת כדי למנוע חזרתיות של שדות.
2. הורדנו מפתח זר שנרשם בerd שלא לצורך.
3. הוספנו שדות נוספים לטבלת health\_status שיאפיינו בצורה מלאה את הליך הצהרת הבריאות.
4. הורדנו את הקשר country\_price כיוון שהמחיר מחושב מאחורי הקלעים.
5. הוספנו קשר covers בין health\_status לinsurance כך שהוא מקשר בסופו של דבר בין ההזמנה בישות orders לביטוחים הנוספים שהנוסע לוקח.

## פרויקט בסיסי נתונים – חלק ג'

# פרק ראשון - מטלות חובה

**ERD**

****

**מודל טבלאי:**

**Passengers** (ID, gender, first\_name, last\_name, birthday, Gender(Gender Lookup)

**Main\_customers**(ID(passenger), address\_city, address\_street, address\_number, phone, email)

**Orders**(order\_id, order\_date, dep\_date, arriv\_date, Payments, credit\_card\_num(payments))

**Days\_in\_country** (order\_id(orders), country\_id(prices\_per\_country), total\_days)

**Prices\_per\_country**(country\_id, price\_per\_day)

**Payments** (credit\_card\_num, expiration\_date, CVV, Id(payments\_id))

**Payments\_id**(id, first\_name, last\_name)

**Helath\_status**(Order\_id(Orders), Health\_status\_id, ID(Passengers), Question1, Question2, Question3, Question4)

**Insurances**(Insurance\_ID, Insur\_type)

**Covers**(Insurance\_ID(Insurances),{Health\_status\_id, Order\_id}(Health\_statuse), extra price, ID(Passengers))

**Extra\_insurances**(Insurance\_ID (Insurance), starting\_date, end\_date, model)

**הנחות והסברים נוספים:**

|  |  |
| --- | --- |
| מרכיב | הנחות |
| פיצול payments | פיצלנו את הטבלה לשתי טבלאות בכדי לנרמל מסדר שלישי, מכיוון ששם ושם משפחה הוגדרו על ידי תעודת הזהות. |

# פירוט שדות בסיס הנתונים והערכת גודלם

# Database Table Schema

## Main\_Customers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Main\_Customers** | ID (Passengers) | Varchar(50) | 50 | Primary Key, Foreign Key |
| Phone | Varchar(20) | 20 |  |
| Email | Varchar(50) | 50 |  |
| Adress\_City | Varchar(20) | 20 |  |
| Adress\_Street | Varchar(20) | 20 |  |
| Adress\_Number | Varchar(10) | 10 |  |
| **Total Record Size (Bytes)** |  | **170** |  |

## Passengers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Passengers** | ID | Varchar(50) | 50 | Primary Key |
| First\_Name | Varchar(20) | 20 |  |
| Last\_Name | Varchar(20) | 20 |  |
| Birthdate | Date | 3 |  |
| Gender  (Gender\_Lookup) | Varchar(10) | 10 | Foreign Key |
| **Total Record Size (Bytes)** |  | **103** |  |

## Orders

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Orders** | Order\_ID | Varchar(50) | 50 | Primary Key |
| Order\_Date | Date | 3 |  |
| Dep\_Date | Date | 3 |  |
| Arriv\_Date | Date | 3 |  |
| Credit\_Card\_Num (Payment) | Varchar(16) | 16 | Foreign Key |
| Payments | Smallint | 2 |  |
| **Total Record Size (Bytes)** |  | **77** |  |

## Price\_Per\_Country

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Price\_Per\_Country** | Country\_ID | Varchar(50) | 50 | Primary Key |
| Price\_Per\_Day | Decimal(10,2) | 9 |  |
| **Total Record Size (Bytes)** |  | **59** |  |

## Payments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Payments** | Credit\_Card\_Number | Varchar(16) | 16 | Primary Key |
| Exp\_Date | Date | 3 |  |
| CVV | Varchar(3) | 3 |  |
| ID (payments\_ID) | Varchar(50) | 50 | Foreign Key |
| **Total Record Size (Bytes)** |  | **72** |  |

## Payments\_ID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Payments\_ID** | ID | Varchar(50) | 50 | Primary Key |
| First\_Name | Varchar(20) | 20 |  |
| Last\_Name | Varchar(20) | 20 |  |
| **Total Record Size (Bytes)** |  | **90** |  |

## Health\_Status

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Health\_Status** | Order\_ID (Orders) | Varchar(50) | 50 | Primary Key, Foreign Key |
| Health\_Status\_ID | Varchar(50) | 50 | Primary Key |
| ID (Passengers) | Varchar(50) | 50 | Foreign Key |
| Question\_1 | Bit | 1 |  |
| Question\_2 | Bit | 1 |  |
| Question\_3 | Bit | 1 |  |
| Question\_4 | Bit | 1 |  |
| **Total Record Size (Bytes)** |  | **154** |  |

## Insurance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Insurance** | insurance\_id | Varchar(50) | 50 | Primary Key |
| Insur\_Type  (Insurance\_Lookup) | Varchar(50) | 50 | Foreign Key |
| **Total Record Size (Bytes)** |  | **100** |  |

## Extra\_Insurances

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Extra\_Ins** | Extra\_Insurance\_ID | Varchar(50) | 50 | Primary Key  Foreign Key |
| Starting\_Date | Date | 3 |  |
| End\_Date | Date | 3 |  |
| model | Varchar(50) | 50 |  |
| **Total Record Size (Bytes)** |  | **106** |  |

## Covers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **covers** | Insurance\_ID  (Insurances) | Varchar(50) | 50 | Foreign Key |
| Order\_ID (Health\_Status) | Varchar(50) | 50 | Foreign Key |
| ID (passengers) | Varchar(50) | 50 | Foreign Key |
| Health\_Status\_ID (Health\_Status) | Varchar(50) | 50 | Foreign Key |
| Extra\_Price | Decimal(10,2) | 9 |  |
| **Total Record Size (Bytes)** |  | **209** |  |

## Days\_In\_Country

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table | Field | Data Type | Size (Bytes) | Key |
| **Country\_Price** | Order\_ID (Orders) | Varchar(50) | 50 | Primary Key, Foreign Key |
| Country\_ID (Price\_Per\_Country) | Varchar(50) | 50 | Primary Key, Foreign Key |
| Total\_Days | SmallInt | 2 |  |
| **Total Record Size (Bytes)** |  | **102** |  |

**הנחות והסברים נוספים:**

## הנחות על שדות בסיסי הנתונים:

* בישות main\_customer הגדרנו את מספר הטלפון כvarchar מכיוון שיכולים להיות תווים שאינם מספרים ובint מספר לא יכול להתחיל באפס.
* בישות payments השדה credit\_card\_number מוגדר כvarchar מאותן סיבות כמו בטבלת main\_customer.
* בישות health\_status השדות של השאלות מוגדרות מסוג bit כיוון שנצפה שיכיל תשובה בינארית בהקשר לשאלה.

**הנחות נוספת מההערות על הגשה מספר 2:**

* כלל המזהים עברו להיות מסוג VARCHAR(50) כיוון ששמירת מזהים בתור INTEGER בעייתית, כי אלו שמתחילות ב-0 יאבדו את הספרה הראשונה.
* כלל התאריכים עברו להיות מסוג DATE לאחר שהוגדרה תחילה כ-DATETIME, כיוון שהתאריך עצמו הוא מה שחשוב ואין חשיבות לשעה המדוייקת.
* מספר התשלומים עבר להיות SMALLINT מ-INT כיוון שאין צורך במספר רב של ספרות כאשר מדובר במספרים קטנים ככה"נ.

# שתי שאילתות SELECT ללא קינון

-- Task 1 - SELECT - Q1

-- Business Logic:

-- The purpose of this query is to identify countries with a high number of risky trips

-- (trips by passengers who marked '1' in at least one health-related question).

-- These countries may pose a higher risk for insurance purposes.

-- The company might consider strengthening collaborations with local healthcare providers

-- in these countries to ensure efficient and quick service for its clients.

SELECT TOP 10

DIC.country\_id AS CountryID,

COUNT(DIC.order\_id) AS RiskyTrips

FROM

DAYS\_IN\_COUNTRY DIC

JOIN

ORDERS O

ON DIC.order\_id = O.order\_id

JOIN

HEALTH\_STATUS HS

ON O.order\_id = HS.order\_id

WHERE

HS.Question\_1 = 1

OR HS.Question\_2 = 1

OR HS.Question\_3 = 1

OR HS.Question\_4 = 1

GROUP BY

DIC.country\_id

ORDER BY

RiskyTrips DESC;

-- Task 1 - SELECT - Q2

-- Business Logic:

-- The goal of this query is to identify passengers who visited multiple countries (more than one) in a single trip.

-- Additionally, the query calculates the trip duration using the DATEDIFF() system function.

-- These insights can help the company identify high-value customers who take complex, long trips and design tailored premium services for them.

SELECT

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

O.order\_id AS OrderID,

O.dep\_date AS DepartureDate,

O.arriv\_date AS ArrivalDate,

DATEDIFF(DAY, O.dep\_date, O.arriv\_date) AS TripDurationDays,

COUNT(DISTINCT DIC.country\_id) AS CountryCount

FROM

PASSENGERS P

JOIN

Health\_Status HS

ON P.ID = HS.ID

JOIN

Orders o

on HS.Order\_ID = O.Order\_ID

JOIN

DAYS\_IN\_COUNTRY DIC

ON O.order\_id = DIC.order\_id

WHERE

O.dep\_date >= '2025-01-01'

GROUP BY

P.ID, P.First\_Name, P.Last\_Name, O.order\_id, O.dep\_date, O.arriv\_date

HAVING

COUNT(DISTINCT DIC.country\_id) > 1

ORDER BY

CountryCount DESC, TripDurationDays DESC;

# שתי שאילתות SELECT מקוננות

-- Task 1 - SELECT - Q3

-- Business Logic:

-- This query identifies the top 5 passengers with the most critical health status (based on the number of health questions marked as "1")

-- who have visited the largest number of countries in a single trip. It provides details about their travel and health risks.

SELECT TOP 5

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

O.order\_id AS OrderID,

O.dep\_date AS DepartureDate,

O.arriv\_date AS ArrivalDate,

H.TotalHealthRisks,

COUNT(DISTINCT DIC.country\_id) AS CountryCount

FROM

PASSENGERS P

JOIN

Health\_Status HS

ON P.ID = HS.ID

JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

JOIN

DAYS\_IN\_COUNTRY DIC

ON O.order\_id = DIC.order\_id

JOIN

(

SELECT

HS.Health\_status\_id AS order\_id,

(

ISNULL(CAST(HS.Question\_1 AS INT), 0) +

ISNULL(CAST(HS.Question\_2 AS INT), 0) +

ISNULL(CAST(HS.Question\_3 AS INT), 0) +

ISNULL(CAST(HS.Question\_4 AS INT), 0)

) AS TotalHealthRisks

FROM

HEALTH\_STATUS HS

) H

ON H.order\_id = O.order\_id

WHERE

O.dep\_date >= '2025-02-09'

GROUP BY

P.ID, P.First\_Name, P.Last\_Name, O.order\_id, O.dep\_date, O.arriv\_date, H.TotalHealthRisks

HAVING

COUNT(DISTINCT DIC.country\_id) > 1

ORDER BY

H.TotalHealthRisks DESC, CountryCount DESC;

-- Task 1 - SELECT - Q4

-- Business Logic:

-- This query identifies the passenger(s) who took the longest trip (in days) in 2025,

-- using a scalar subquery to calculate the maximum trip duration across all trips in that year.

SELECT

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

O.order\_id AS OrderID,

O.dep\_date AS DepartureDate,

O.arriv\_date AS ArrivalDate,

DATEDIFF(DAY, O.dep\_date, O.arriv\_date) AS TripDurationDays

FROM

PASSENGERS P

JOIN

Health\_Status HS

ON P.ID = HS.ID

JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

WHERE

YEAR(O.dep\_date) = 2025

AND DATEDIFF(DAY, O.dep\_date, O.arriv\_date) = (

SELECT MAX(DATEDIFF(DAY, O2.dep\_date, O2.arriv\_date))

FROM ORDERS O2

WHERE YEAR(O2.dep\_date) = 2025

)

ORDER BY

P.ID;

## שאילתות עסקיות המשלבות Window Functions

-- Task 1 - Window Functions - Q5

-- Business Logic:

-- This query identifies orders with the highest total costs of extra insurance (summed for each order),

-- ranks the orders by their extra insurance costs within each year, and compares the cost of each order to the average for that year.

-- It also divides the orders into quartiles based on their costs, helping to identify high-value customers

-- and understand spending patterns on extra insurance products.

SELECT

O.order\_id AS OrderID,

YEAR(O.dep\_date) AS OrderYear,

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

SUM(C.Extra\_Price) AS TotalExtraInsuranceCost,

RANK() OVER (PARTITION BY YEAR(O.dep\_date) ORDER BY SUM(C.Extra\_Price) DESC) AS RankByYear,

AVG(SUM(C.Extra\_Price)) OVER (PARTITION BY YEAR(O.dep\_date)) AS AvgExtraInsuranceCostPerYear,

NTILE(4) OVER (PARTITION BY YEAR(O.dep\_date) ORDER BY SUM(C.Extra\_Price) DESC) AS Quartile,

CASE

WHEN SUM(C.Extra\_Price) > AVG(SUM(C.Extra\_Price)) OVER (PARTITION BY YEAR(O.dep\_date)) THEN 'Above Average'

WHEN SUM(C.Extra\_Price) = AVG(SUM(C.Extra\_Price)) OVER (PARTITION BY YEAR(O.dep\_date)) THEN 'Average'

ELSE 'Below Average'

END AS PerformanceCategory

FROM

PASSENGERS P

JOIN

Health\_Status HS

ON P.ID = HS.ID

JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

JOIN

COVERS C

ON O.order\_id = C.order\_id

WHERE

C.Extra\_Price > 0

GROUP BY

O.order\_id, YEAR(O.dep\_date), P.ID, P.First\_Name, P.Last\_Name

ORDER BY

OrderYear, RankByYear;

-- Task 1 - Window Functions - Q6

-- Business Logic:

-- This query identifies customers with the highest divide payments for a single order,

-- ranks orders by total payment amount within each year, and compares the payment amount to the annual average.

-- It helps identify high-value customers and analyze payment patterns.

SELECT

O.order\_id AS OrderID,

YEAR(O.dep\_date) AS OrderYear,

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

O.payments AS TotalPayment,

RANK() OVER (PARTITION BY YEAR(O.dep\_date) ORDER BY O.payments DESC) AS RankByYear,

AVG(O.payments) OVER (PARTITION BY YEAR(O.dep\_date)) AS AvgPaymentPerYear,

NTILE(4) OVER (PARTITION BY YEAR(O.dep\_date) ORDER BY O.payments DESC) AS Quartile,

CASE

WHEN O.payments > AVG(O.payments) OVER (PARTITION BY YEAR(O.dep\_date)) THEN 'Above Average'

WHEN O.payments = AVG(O.payments) OVER (PARTITION BY YEAR(O.dep\_date)) THEN 'Average'

ELSE 'Below Average'

END AS PerformanceCategory

FROM

PASSENGERS P

JOIN

Health\_Status HS

ON P.ID = HS.ID

JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

JOIN

PAYMENTS PY

ON O.Credit\_Card\_number = PY.Credit\_Card\_number

WHERE

O.Payments > 0

ORDER BY

OrderYear, RankByYear;

## שאילתה מקוננת תוך שימוש ב-CTE (פסקת WITH)

-- Task 1 - With - Q7

-- business logic – the goal of this query is to find the top 5 customers with the highest insurance price.

WITH

TripDays AS (

SELECT

DIC.order\_id AS OrderID,

DIC.country\_id AS CountryID,

DIC.Total\_Days AS TotalDaysInCountry,

PPC.price\_per\_day \* DIC.Total\_Days AS CountryInsuranceCost

FROM

DAYS\_IN\_COUNTRY DIC

JOIN

PRICE\_PER\_COUNTRY PPC

ON

DIC.country\_id = PPC.country\_id

),

BaseInsurance AS (

SELECT

O.order\_id AS OrderID,

O.dep\_date AS DepartureDate,

O.arriv\_date AS ArrivalDate,

DATEDIFF(DAY, O.dep\_date, O.arriv\_date) AS TripDays,

SUM(IL.price\_per\_day \* DATEDIFF(DAY, O.dep\_date, O.arriv\_date)) AS TotalBaseInsuranceCost,

SUM(C.Extra\_Price) AS ExtraPriceBaseInsurance

FROM

ORDERS O

JOIN

COVERS C

ON

O.order\_id = C.order\_id

JOIN

INSURANCE I

ON

C.insurance\_id = I.insurance\_id

JOIN

INSURANCE\_LOOKUP IL

ON

I.insur\_type = IL.insur\_type

GROUP BY

O.order\_id, O.dep\_date, O.arriv\_date

),

ExtraInsurance AS (

SELECT

C.order\_id AS OrderID,

EI.Starting\_Date AS ExtraStartDate,

EI.end\_date AS ExtraEndDate,

DATEDIFF(DAY, EI.Starting\_Date, EI.end\_date) AS ExtraTripDays,

SUM(IL.price\_per\_day \* DATEDIFF(DAY, EI.Starting\_Date, EI.end\_date)) AS TotalExtraInsuranceBaseCost,

SUM(C.Extra\_Price) AS ExtraPriceExtraInsurance

FROM

ORDERS O

JOIN

COVERS C

ON

O.order\_id = C.order\_id

JOIN

INSURANCE I

ON

C.Insurance\_id = I.Insurance\_id

join

Extra\_Insurances EI

on

EI.insurance\_id = i.insurance\_id

JOIN

INSURANCE\_LOOKUP IL

ON

I.Insur\_Type = IL.insur\_type

GROUP BY

C.order\_id, EI.Starting\_Date, EI.end\_date

),

TotalInsurance AS (

SELECT

BI.OrderID,

BI.DepartureDate,

BI.ArrivalDate,

BI.TripDays,

COALESCE(SUM(TD.CountryInsuranceCost), 0) AS TotalCountryInsuranceCost,

COALESCE(BI.TotalBaseInsuranceCost, 0) AS TotalBaseInsuranceCost,

COALESCE(SUM(EI.TotalExtraInsuranceBaseCost), 0) AS TotalExtraInsuranceCost,

COALESCE(BI.ExtraPriceBaseInsurance, 0) + COALESCE(SUM(EI.ExtraPriceExtraInsurance), 0) AS TotalExtraPrice,

COALESCE(SUM(TD.CountryInsuranceCost), 0) +

COALESCE(BI.TotalBaseInsuranceCost, 0) +

COALESCE(SUM(EI.TotalExtraInsuranceBaseCost), 0) +

COALESCE(BI.ExtraPriceBaseInsurance, 0) +

COALESCE(SUM(EI.ExtraPriceExtraInsurance), 0) AS TotalInsuranceCost

FROM

BaseInsurance BI

LEFT JOIN

ExtraInsurance EI

ON

BI.OrderID = EI.OrderID

LEFT JOIN

TripDays TD

ON

BI.OrderID = TD.OrderID

GROUP BY

BI.OrderID, BI.DepartureDate, BI.ArrivalDate, BI.TripDays, BI.TotalBaseInsuranceCost, BI.ExtraPriceBaseInsurance

),

CustomerInsuranceSummary AS (

SELECT

P.ID AS PassengerID,

P.First\_Name + ' ' + P.Last\_Name AS PassengerName,

MIN(TI.DepartureDate) AS FirstDepartureDate,

MAX(TI.ArrivalDate) AS LastArrivalDate,

SUM(TI.TripDays) AS TotalTripDays,

SUM(TI.TotalInsuranceCost) AS TotalInsurancePaid

FROM

HEALTH\_STATUS HS

JOIN

PASSENGERS P

ON

HS.ID = P.ID

JOIN

ORDERS O

ON

HS.order\_id = O.order\_id

JOIN

TotalInsurance TI

ON

O.order\_id = TI.OrderID

GROUP BY

P.ID, P.First\_Name, P.Last\_Name

),

TopCustomers AS (

SELECT

CIS.PassengerID,

CIS.PassengerName,

CIS.FirstDepartureDate,

CIS.LastArrivalDate,

CIS.TotalTripDays,

CIS.TotalInsurancePaid,

RANK() OVER (ORDER BY CIS.TotalInsurancePaid DESC) AS RankByInsurance

FROM

CustomerInsuranceSummary CIS

)

SELECT

TC.PassengerID,

TC.PassengerName,

TC.FirstDepartureDate,

TC.LastArrivalDate,

TC.TotalTripDays,

TC.TotalInsurancePaid,

TC.RankByInsurance

FROM

TopCustomers TC

WHERE

TC.RankByInsurance <= 5

ORDER BY

TC.RankByInsurance;

## מטלה 2 – יישומי כלים מתקדמים

# View

-- Task 2 - View - Q1

-- the goal of this view function is to show the highest risk customer along side their trip duration

CREATE VIEW vw\_HighRiskOrders AS

SELECT

O.order\_id AS OrderID,

P.ID AS PassengerID,

P.First\_Name + ' ' + P.Last\_Name AS PassengerName,

HS.Health\_status\_id AS HealthStatusID,

(ISNULL(CAST(HS.Question\_1 AS INT), 0) +

ISNULL(CAST(HS.Question\_2 AS INT), 0) +

ISNULL(CAST(HS.Question\_3 AS INT), 0) +

ISNULL(CAST(HS.Question\_4 AS INT), 0)) AS TotalHealthRisks,

O.dep\_date AS DepartureDate,

O.arriv\_date AS ArrivalDate,

DATEDIFF(DAY, O.dep\_date, O.arriv\_date) AS TripDuration

FROM

PASSENGERS P

JOIN

HEALTH\_STATUS HS

ON P.ID = HS.ID

JOIN

ORDERS O

ON HS.order\_id = O.order\_id

WHERE

(ISNULL(CAST(HS.Question\_1 AS INT), 0) +

ISNULL(CAST(HS.Question\_2 AS INT), 0) +

ISNULL(CAST(HS.Question\_3 AS INT), 0) +

ISNULL(CAST(HS.Question\_4 AS INT), 0)) > 2;

-- Operate the view function

SELECT \* FROM vw\_HighRiskOrders ORDER BY TotalHealthRisks DESC;

--delete table

DROP VIEW IF EXISTS vw\_HighRiskOrders;

# פונקציות (Functions)

-- Task 2 - Function - Q2.1

-- the goal of this function is to find the risk score for a specific customer

CREATE FUNCTION dbo.fn\_GetHighRiskOrders

(

@PassengerID varchar(50)

)

RETURNS TABLE

AS

RETURN

(

SELECT

OrderID,

PassengerName,

TotalHealthRisks,

DepartureDate,

ArrivalDate,

TripDuration

FROM

vw\_HighRiskOrders

WHERE

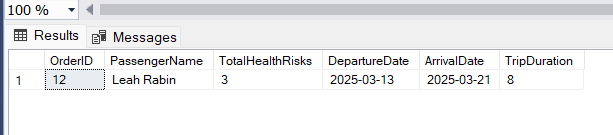
PassengerID = @PassengerID

);

-- Operate the fn\_GetHighRiskOrders function

SELECT \*

FROM dbo.fn\_GetHighRiskOrders('1000000171');



-- Task 2 - Function - Q2.2

-- the goal of this function is to find the number of payments that the company expect in a specific month duration

CREATE FUNCTION dbo.fn\_ExpectedPaymentsByMonth

(

@MonthOffset INT

)

RETURNS INT

AS

BEGIN

DECLARE @ExpectedPayments INT;

DECLARE @TargetDate DATE;

SET @TargetDate = DATEADD(MONTH, @MonthOffset, GETDATE());

SELECT

@ExpectedPayments = COUNT(\*)

FROM

PASSENGERS P

JOIN

HEALTH\_STATUS HS

ON

P.ID = HS.ID

JOIN

ORDERS O

ON

HS.ORDER\_ID = O.ORDER\_ID

JOIN

PAYMENTS PMT

ON

O.Credit\_Card\_Number = PMT.Credit\_Card\_Number

WHERE

DATEADD(MONTH, O.Payments - 1, O.Order\_Date) >= EOMONTH(@TargetDate, -1)

AND O.Order\_Date <= @TargetDate

AND DATEDIFF(MONTH, O.Order\_Date, @TargetDate) + 1 <= O.Payments;

RETURN @ExpectedPayments;

END;

-- Operate the fn\_GetHighRiskOrders

-- Calculate expected payments for two months ahead

SELECT dbo.fn\_ExpectedPaymentsByMonth(2) AS PaymentsInTwoMonths;

# 

# Trigger

-- Task 2 - TRIGGER - Q3

-- the goal of the trigger is to check that the dep\_date is before the arriv\_date and that the order date is before the dep\_date

CREATE TRIGGER trg\_ValidateDates

ON Orders

AFTER INSERT, UPDATE

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted

WHERE Dep\_Date <= Order\_Date

OR Arriv\_Date <= Dep\_Date

)

BEGIN

ROLLBACK TRANSACTION;

THROW 50000, 'Validation Error: Dep\_Date must be after Order\_Date and Arriv\_Date must be after Dep\_Date.', 1;

END

END;

# פרוצדורה שמורה ((Stored Procedure

-- Task 2 - Stored Procedure - Q4

-- the goal of this procedure is to automated system to identify high-risk trips and notify passengers, enhancing safety and customer trust.

CREATE TABLE dbo.Notifications

(

NotificationID INT IDENTITY(1,1) PRIMARY KEY,

NotificationDate DATETIME NOT NULL,

PassengerID INT NOT NULL,

OrderID INT NOT NULL,

NotificationMessage NVARCHAR(255) NOT NULL

);

CREATE PROCEDURE dbo.sp\_GenerateHighRiskTripNotifications

(

@PassengerID INT

)

AS

BEGIN

-- Step 1: Create a temporary table to store high-risk orders for the passenger

CREATE TABLE #HighRiskOrders

(

OrderID INT,

PassengerName NVARCHAR(100),

TotalHealthRisks INT,

DepartureDate DATE,

ArrivalDate DATE,

TripDuration INT);

-- Step 2: Insert high-risk orders for the given passenger into the temporary table

INSERT INTO #HighRiskOrders

SELECT

OrderID,

PassengerName,

TotalHealthRisks,

DepartureDate,

ArrivalDate,

TripDuration

FROM

dbo.fn\_GetHighRiskOrders(@PassengerID);

-- Step 3: Insert high-risk trip notifications into the Notifications table

INSERT INTO dbo.Notifications

(

NotificationDate,

PassengerID,

OrderID,

NotificationMessage)

SELECT

GETDATE() AS NotificationDate,

@PassengerID AS PassengerID,

HRO.OrderID AS OrderID,

CONCAT('High-risk trip detected. Total health risks: ', HRO.TotalHealthRisks,

'. Trip duration: ', HRO.TripDuration, ' days.') AS NotificationMessage

FROM

#HighRiskOrders HRO;

-- Step 4: Return a summary of the notifications inserted

SELECT

N.NotificationDate,

N.PassengerID,

N.OrderID,

N.NotificationMessage

FROM

dbo.Notifications N

WHERE

N.PassengerID = @PassengerID;

-- Step 5: Cleanup temporary table

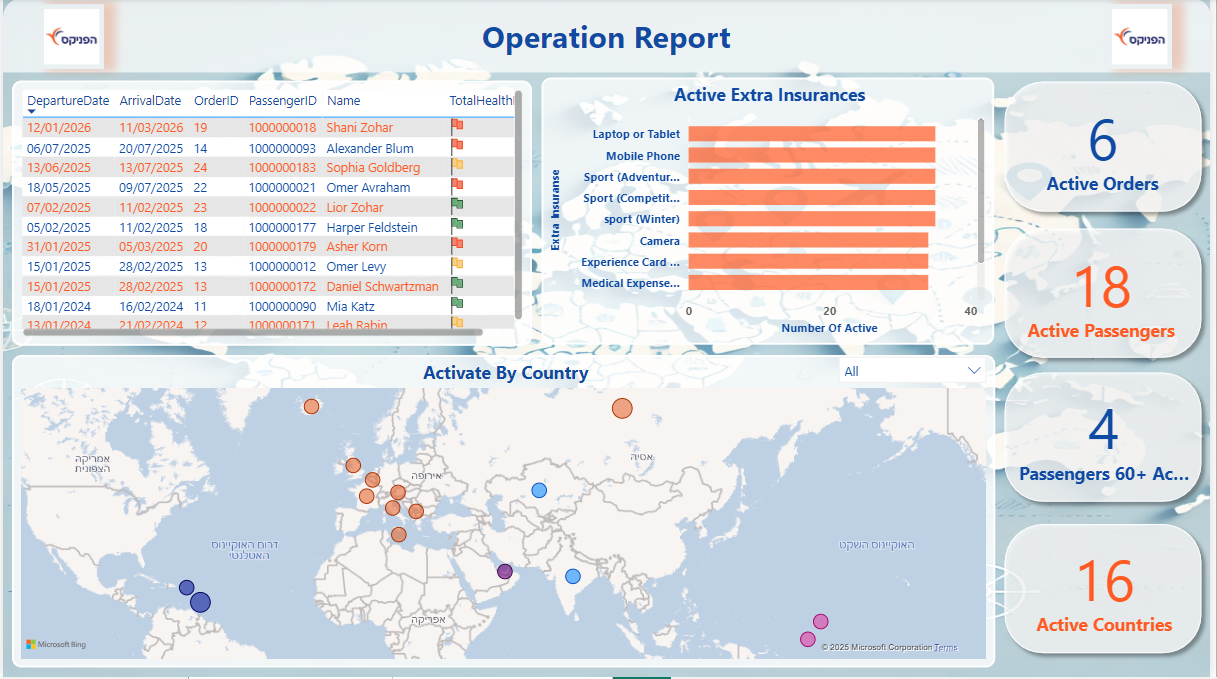
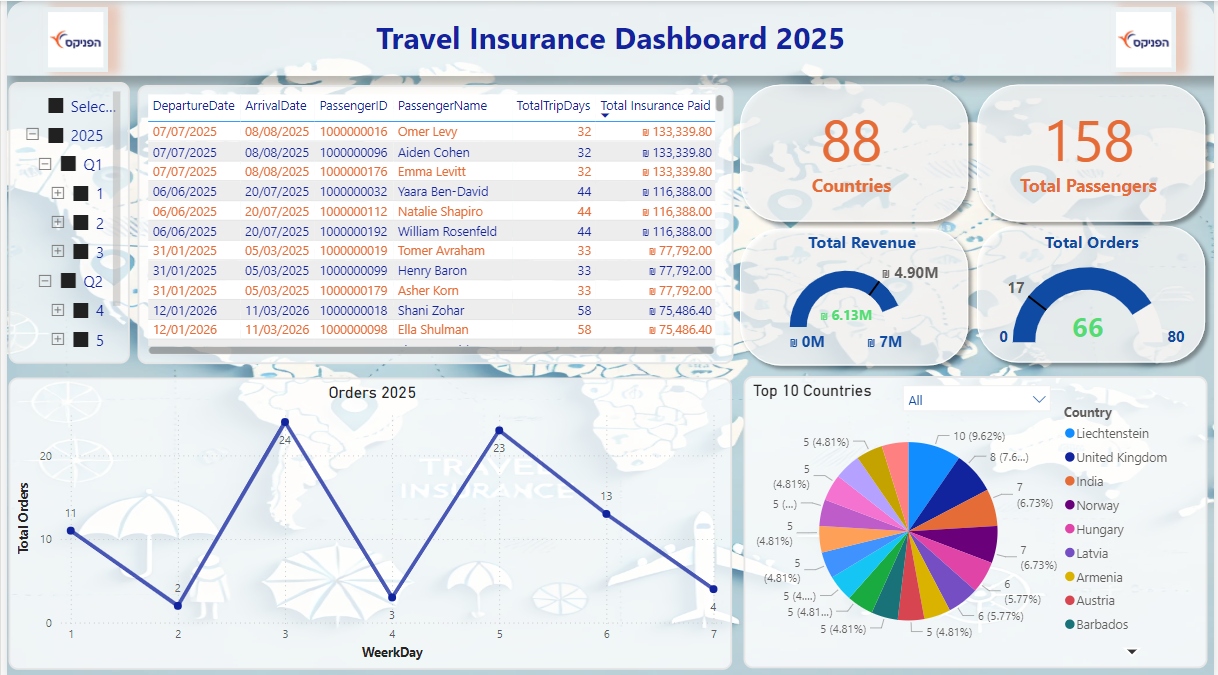
DROP TABLE #HighRiskOrders;

END;

-- Operate the sp\_GenerateHighRiskTripNotifications Stored Procedure

EXEC dbo.sp\_GenerateHighRiskTripNotifications @PassengerID = '1000000171'

מטלה 3 – כלים להצגת נתונים



## מטלה 4 – אופטימיזציה של שאילתות באמצעות Generative AI

-- Task 4 - Generative AI - Improved Q1

CREATE INDEX IDX\_DaysInCountry\_OrderID\_CountryID ON Days\_In\_Country (Order\_ID, Country\_ID);

CREATE INDEX IDX\_Orders\_DepDate ON Orders (Dep\_Date);

CREATE INDEX IDX\_Orders\_OrderID ON Orders (Order\_ID);

CREATE INDEX IDX\_Passengers\_ID ON Passengers (ID);

CREATE INDEX IDX\_HealthStatus\_OrderID\_ID ON Health\_Status (Order\_ID, ID);

SELECT

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

O.Order\_ID AS OrderID,

O.Dep\_Date AS DepartureDate,

O.Arriv\_Date AS ArrivalDate,

DATEDIFF(DAY, O.Dep\_Date, O.Arriv\_Date) AS TripDurationDays,

CountryStats.CountryCount

FROM

PASSENGERS P

INNER JOIN

Health\_Status HS

ON P.ID = HS.ID

INNER JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

INNER JOIN (

SELECT

DIC.Order\_ID,

COUNT(DISTINCT DIC.Country\_ID) AS CountryCount

FROM

Days\_In\_Country DIC

GROUP BY

DIC.Order\_ID

HAVING

COUNT(DISTINCT DIC.Country\_ID) > 1

) AS CountryStats

ON O.Order\_ID = CountryStats.Order\_ID

WHERE

O.Dep\_Date >= '2025-01-01'

ORDER BY

CountryStats.CountryCount DESC,

TripDurationDays DESC;

-- Task 4 - Generative AI - Improved Q5

CREATE INDEX IDX\_Covers\_OrderID\_ExtraPrice ON Covers (Order\_ID, Extra\_Price);

WITH ExtraInsuranceCosts AS (

SELECT

C.Order\_ID,

YEAR(O.Dep\_Date) AS OrderYear,

SUM(C.Extra\_Price) AS TotalExtraInsuranceCost

FROM

Covers C

JOIN

Orders O

ON C.Order\_ID = O.Order\_ID

WHERE

C.Extra\_Price > 0

GROUP BY

C.Order\_ID, YEAR(O.Dep\_Date)

),

RankedOrders AS (

SELECT

EIC.Order\_ID,

EIC.OrderYear,

EIC.TotalExtraInsuranceCost,

RANK() OVER (PARTITION BY EIC.OrderYear ORDER BY EIC.TotalExtraInsuranceCost DESC) AS RankByYear,

AVG(EIC.TotalExtraInsuranceCost) OVER (PARTITION BY EIC.OrderYear) AS AvgExtraInsuranceCostPerYear,

NTILE(4) OVER (PARTITION BY EIC.OrderYear ORDER BY EIC.TotalExtraInsuranceCost DESC) AS Quartile

FROM

ExtraInsuranceCosts EIC

)

SELECT

R.Order\_ID AS OrderID,

R.OrderYear,

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

R.TotalExtraInsuranceCost,

R.RankByYear,

R.AvgExtraInsuranceCostPerYear,

R.Quartile,

CASE

WHEN R.TotalExtraInsuranceCost > R.AvgExtraInsuranceCostPerYear THEN 'Above Average'

WHEN R.TotalExtraInsuranceCost = R.AvgExtraInsuranceCostPerYear THEN 'Average'

ELSE 'Below Average'

END AS PerformanceCategory

FROM

RankedOrders R

JOIN

Health\_Status HS

ON R.Order\_ID = HS.Order\_ID

JOIN

Passengers P

ON HS.ID = P.ID

ORDER BY

R.OrderYear, R.RankByYear;

**קישור לשיחה עם הצאט:**

-- Step 1: Create Independent Tables

CREATE TABLE payments\_ID (

ID VARCHAR(50) PRIMARY KEY NOT NULL,

First\_Name VARCHAR(20) NOT NULL,

Last\_Name VARCHAR(20) NOT NULL

);

CREATE TABLE Payments (

Credit\_Card\_Number VARCHAR(16) PRIMARY KEY NOT NULL CHECK (LEN(Credit\_Card\_Number) = 16),

Exp\_Date DATE NOT NULL,

CVV VARCHAR(3) NOT NULL CHECK (ISNUMERIC(CVV) = 1 AND LEN(CVV) = 3),

ID VARCHAR(50) NOT NULL,

FOREIGN KEY (ID) REFERENCES payments\_ID(ID)

);

-- Step 2: Lookup Tables

CREATE TABLE Insurance\_Lookup (

insur\_type varchar(50) NOT NULL UNIQUE,

price\_per\_day decimal(10,2) NOT NULL

);

INSERT INTO Insurance\_Lookup (insur\_type, price\_per\_day)

VALUES

('Search and Rescue', 0.2),

('Trip Cancellation/Shortening', 0.7),

('Baggage', 0.5),

('Mobile Phone', 1),

('Laptop or Tablet', 0.8),

('Sport (Adventurous)', 0.5),

('sport (Winter)', 10),

('Sport (Competitive)', 25), -- This price is for the whole period

('Experience Card or Event', 5),-- This price is for the whole period

('Rental Car (Private)', 8),

('Rental Car (Jeep/Van)', 15),

('Camera', 1),

('Medical Expenses in Israel', 0.5);

CREATE TABLE Gender\_Lookup (

Gender\_ID INT PRIMARY KEY IDENTITY(1,1),

Gender\_Value VARCHAR(10) NOT NULL UNIQUE

);

INSERT INTO Gender\_Lookup (Gender\_Value)

VALUES ('Male'), ('Female'), ('Other');

-- Step 3: Dependent Tables

CREATE TABLE Passengers (

ID VARCHAR(50) PRIMARY KEY NOT NULL,

First\_Name VARCHAR(20) NOT NULL,

Last\_Name VARCHAR(20) NOT NULL,

Birthdate DATE NOT NULL CHECK (Birthdate <= GETDATE()),

Gender VARCHAR(10) NOT NULL,

FOREIGN KEY (Gender) REFERENCES Gender\_Lookup(Gender\_Value)

);

CREATE TABLE Main\_Customer (

ID VARCHAR(50) NOT NULL,

Phone VARCHAR(20) NOT NULL CHECK ((Phone LIKE '+%' OR Phone LIKE '[0-9]%') AND LEN(Phone) BETWEEN 5 AND 20),

Email VARCHAR(50) NOT NULL CHECK (Email LIKE '%@%.%'),

Adress\_City VARCHAR(20) NOT NULL,

Adress\_Street VARCHAR(20) NOT NULL,

Adress\_Number VARCHAR(10) NOT NULL,

First\_Name VARCHAR(20) NOT NULL,

Last\_Name VARCHAR(20) NOT NULL,

Birthdate DATE NOT NULL CHECK (Birthdate <= GETDATE()),

Gender VARCHAR(10) NOT NULL,

FOREIGN KEY (Gender) REFERENCES Gender\_Lookup(Gender\_Value),

FOREIGN KEY (ID) REFERENCES Passengers(ID)

);

CREATE TABLE Orders (

Order\_ID VARCHAR(50) PRIMARY KEY NOT NULL,

Order\_Date DATE NOT NULL,

Dep\_Date DATE NOT NULL CHECK (Dep\_Date >= GETDATE()),

Arriv\_Date DATE NOT NULL,

Credit\_Card\_number VARCHAR(16) NOT NULL,

Payments SmallINT NOT NULL CHECK (Payments >= 0),

FOREIGN KEY (Credit\_Card\_number) REFERENCES Payments(Credit\_Card\_Number)

);

CREATE TABLE Health\_Status (

Order\_ID VARCHAR(50) NOT NULL,

HEALTH\_STATUS\_ID VARCHAR(50) NOT NULL,

ID VARCHAR(50) NOT NULL,

Question\_1 BIT NOT NULL CHECK (Question\_1 IN (0, 1)),

Question\_2 BIT NOT NULL CHECK (Question\_2 IN (0, 1)),

Question\_3 BIT NOT NULL CHECK (Question\_3 IN (0, 1)),

Question\_4 BIT NOT NULL CHECK (Question\_4 IN (0, 1)),

PRIMARY KEY (Order\_ID, HEALTH\_STATUS\_ID),

FOREIGN KEY (Order\_ID) REFERENCES Orders(Order\_ID),

FOREIGN KEY (ID) REFERENCES Passengers(ID)

);

CREATE TABLE Insurance (

insurance\_id VARCHAR(50) PRIMARY KEY NOT NULL,

insur\_type VARCHAR(50) NOT NULL,

FOREIGN KEY (insur\_type) REFERENCES Insurance\_Lookup(insur\_type)

);

CREATE TABLE Extra\_Insurances (

Extra\_Insurance\_ID VARCHAR(50) PRIMARY KEY,

Extra\_Insur\_Type VARCHAR(50) NOT NULL,

Starting\_Date DATE NOT NULL,

End\_Date DATE NOT NULL,

Model VARCHAR(50) NULL,

FOREIGN KEY (Extra\_Insur\_Type) REFERENCES Insurance\_Lookup(insur\_type)

);

CREATE TABLE Covers (

Insurance\_ID VARCHAR(50) NULL,

Extra\_Insurance\_ID VARCHAR(50) NULL,

Order\_ID VARCHAR(50) NULL,

ID VARCHAR(50) NULL,

HEALTH\_STATUS\_ID VARCHAR(50) NOT NULL,

Extra\_Price DECIMAL(10,2) NULL CHECK (Extra\_Price > 0),

FOREIGN KEY (Insurance\_ID) REFERENCES Insurance(Insurance\_ID),

FOREIGN KEY (Extra\_Insurance\_ID) REFERENCES Extra\_Insurances(Extra\_Insurance\_ID),

FOREIGN KEY (ID) REFERENCES Passengers(ID),

FOREIGN KEY (Order\_ID, HEALTH\_STATUS\_ID) REFERENCES Health\_Status(Order\_ID, HEALTH\_STATUS\_ID)

);

CREATE TABLE Price\_Per\_Country (

Country\_ID VARCHAR(50) PRIMARY KEY NOT NULL,

Price\_Per\_Day DECIMAL(10,2) NOT NULL CHECK (Price\_Per\_Day > 0)

);

CREATE TABLE Days\_In\_Country (

Order\_ID VARCHAR(50) NOT NULL,

Country\_ID VARCHAR(50) NOT NULL,

Total\_Days SMALLINT NOT NULL,

PRIMARY KEY (Order\_ID, Country\_ID),

FOREIGN KEY (Order\_ID) REFERENCES Orders(Order\_ID),

FOREIGN KEY (Country\_ID) REFERENCES Price\_Per\_Country(Country\_ID)

);

-- Step 4: Drop Tables (Cleanup)

-- Drop tables if they exist in the correct order

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Covers') AND type = 'U') DROP TABLE Covers;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Days\_In\_Country') AND type = 'U') DROP TABLE Days\_In\_Country;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Price\_Per\_Country') AND type = 'U') DROP TABLE Price\_Per\_Country;

-- Foreign key dependencies

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Health\_Status') AND type = 'U') DROP TABLE Health\_Status;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Extra\_Ins') AND type = 'U') DROP TABLE Extra\_Ins;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Orders') AND type = 'U') DROP TABLE Orders;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Payments') AND type = 'U') DROP TABLE Payments;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('payments\_ID') AND type = 'U') DROP TABLE payments\_ID;

-- Dependent tables

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Main\_Customer') AND type = 'U') DROP TABLE Main\_Customer;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Passengers') AND type = 'U') DROP TABLE Passengers;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Insurance') AND type = 'U') DROP TABLE Insurance;

-- Lookup tables

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Insurance\_Lookup') AND type = 'U') DROP TABLE Insurance\_Lookup;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Gender\_Lookup') AND type = 'U') DROP TABLE Gender\_Lookup;

-- Remove unused lookup tables

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Country\_Lookup') AND type = 'U') DROP TABLE Country\_Lookup;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Extra\_Ins\_Type\_Lookup') AND type = 'U') DROP TABLE Extra\_Ins\_Type\_Lookup;

-- Remove the Extra\_Insurances table

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Extra\_Insurances') AND type = 'U') DROP TABLE Extra\_Insurances;

-- Step 1: Drop the constraints referencing Insurance\_Lookup

ALTER TABLE Extra\_Insurances DROP CONSTRAINT FK\_Extra\_Insurances\_Insurance\_Lookup;

ALTER TABLE Covers DROP CONSTRAINT FK\_Covers\_Insurance\_Lookup;

-- Step 2: Drop the dependent tables (if necessary)

DROP TABLE Extra\_Insurances;

DROP TABLE Covers;

-- Step 3: Drop the Insurance\_Lookup table

DROP TABLE Insurance\_Lookup;

SELECT \*

FROM INFORMATION\_SCHEMA.TABLES

WHERE TABLE\_NAME = 'Extra\_Insurances';

SELECT TABLE\_NAME

FROM INFORMATION\_SCHEMA.TABLES;

SELECT \*

FROM fn\_my\_permissions(NULL, 'DATABASE')

WHERE permission\_name LIKE '%ALTER%' OR permission\_name LIKE '%SELECT%';

ALTER TABLE Covers DROP CONSTRAINT FK\_Covers\_Insurance\_Lookup;

-- Step 4: Drop Tables (Cleanup)

-- Drop tables if they exist in the correct order

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Covers') AND type = 'U') DROP TABLE Covers;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Days\_In\_Country') AND type = 'U') DROP TABLE Days\_In\_Country;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Price\_Per\_Country') AND type = 'U') DROP TABLE Price\_Per\_Country;

-- Foreign key dependencies

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Health\_Status') AND type = 'U') DROP TABLE Health\_Status;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Extra\_Ins') AND type = 'U') DROP TABLE Extra\_Ins;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Orders') AND type = 'U') DROP TABLE Orders;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Payments') AND type = 'U') DROP TABLE Payments;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('payments\_ID') AND type = 'U') DROP TABLE payments\_ID;

-- Dependent tables

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Main\_Customer') AND type = 'U') DROP TABLE Main\_Customer;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Passengers') AND type = 'U') DROP TABLE Passengers;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Insurance') AND type = 'U') DROP TABLE Insurance;

-- Lookup tables

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Insurance\_Lookup') AND type = 'U') DROP TABLE Insurance\_Lookup;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Gender\_Lookup') AND type = 'U') DROP TABLE Gender\_Lookup;

-- Remove unused lookup tables

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Country\_Lookup') AND type = 'U') DROP TABLE Country\_Lookup;

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Extra\_Ins\_Type\_Lookup') AND type = 'U') DROP TABLE Extra\_Ins\_Type\_Lookup;

-- Remove the Extra\_Insurances table

IF EXISTS (SELECT 1 FROM sys.objects WHERE object\_id = OBJECT\_ID('Extra\_Insurances') AND type = 'U') DROP TABLE Extra\_Insurances;

i have this data base that is related to foregion insurence company and now my task is to improve the running time as described in the picture i attached. i want you to take this query and write an more efficent code for it:

-- Task 1 - SELECT - Q2

-- Business Logic:

-- The goal of this query is to identify passengers who visited multiple countries (more than one) in a single trip.

-- Additionally, the query calculates the trip duration using the DATEDIFF() system function.

-- These insights can help the company identify high-value customers who take complex, long trips and design tailored premium services for them.

SELECT

P.ID AS PassengerID, -- Passenger ID

P.First\_Name AS FirstName, -- Passenger first name

P.Last\_Name AS LastName, -- Passenger last name

O.order\_id AS OrderID, -- Order ID

O.dep\_date AS DepartureDate, -- Departure date of the trip

O.arriv\_date AS ArrivalDate, -- Arrival date of the trip

DATEDIFF(DAY, O.dep\_date, O.arriv\_date) AS TripDurationDays, -- Trip duration in days

COUNT(DISTINCT DIC.country\_id) AS CountryCount -- Number of unique countries in the trip

FROM

PASSENGERS P

JOIN

Health\_Status HS

ON P.ID = HS.ID

JOIN

Orders o

on HS.Order\_ID = O.Order\_ID

JOIN

DAYS\_IN\_COUNTRY DIC

ON O.order\_id = DIC.order\_id

WHERE

O.dep\_date >= '2025-01-01' -- Include only trips that start on or after January 1, 2025

GROUP BY

P.ID, P.First\_Name, P.Last\_Name, O.order\_id, O.dep\_date, O.arriv\_date

HAVING

COUNT(DISTINCT DIC.country\_id) > 1 -- Ensure the trip includes more than one country

ORDER BY

CountryCount DESC, TripDurationDays DESC; -- Sort by the highest number of countries and longest trips

ChatGPT said:

To improve the efficiency of the query you've provided, we can optimize both the structure of the SQL query and potentially the underlying database structure, assuming modifications are allowed. Let's tackle the optimization step by step.

Query Optimization

Key Improvements:

Minimize Joins:

The JOIN operations are costly. If there are unnecessary joins or if certain fields can be queried from fewer tables, this should be addressed.

Indexes:

Adding appropriate indexes to frequently filtered columns (O.dep\_date, DIC.country\_id) and join keys (P.ID, O.Order\_ID) can speed up the query execution.

Avoid DISTINCT in COUNT:

Replace COUNT(DISTINCT column) with an optimized aggregation method if possible.

Reorganize Filters:

Move WHERE conditions as early as possible to reduce the data scanned.

Here's an optimized version of your query:

sql

Copy

Edit

-- Optimized Query for identifying high-value passengers with complex trips

SELECT

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

O.Order\_ID AS OrderID,

O.Dep\_Date AS DepartureDate,

O.Arriv\_Date AS ArrivalDate,

DATEDIFF(DAY, O.Dep\_Date, O.Arriv\_Date) AS TripDurationDays,

CountryStats.CountryCount -- Pre-computed country count

FROM

PASSENGERS P

INNER JOIN

Health\_Status HS

ON P.ID = HS.ID

INNER JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

-- Subquery to precompute country counts for each order

INNER JOIN (

SELECT

DIC.Order\_ID,

COUNT(DISTINCT DIC.Country\_ID) AS CountryCount

FROM

Days\_In\_Country DIC

GROUP BY

DIC.Order\_ID

HAVING

COUNT(DISTINCT DIC.Country\_ID) > 1

) AS CountryStats

ON O.Order\_ID = CountryStats.Order\_ID

WHERE

O.Dep\_Date >= '2025-01-01'

ORDER BY

CountryStats.CountryCount DESC,

TripDurationDays DESC;

Explanation of Improvements:

Subquery for Country Count:

The Days\_In\_Country table is aggregated first to calculate the country count for each Order\_ID. This reduces the amount of data processed in subsequent joins.

Reduced DISTINCT Usage:

The DISTINCT is now applied only in the subquery. This minimizes the computational overhead in the main query.

Indexing:

Ensure indexes exist on the following columns:

Days\_In\_Country.Order\_ID

Days\_In\_Country.Country\_ID

Orders.Order\_ID

Orders.Dep\_Date

Passengers.ID

Early Filtering:

The WHERE clause ensures only recent orders are considered, reducing the rows processed in subsequent joins.

Simplified HAVING Clause:

The HAVING condition is pushed to the subquery, reducing the rows that are returned to the main query.

Database-Level Optimization

Index Recommendations:

Composite index on Days\_In\_Country(Order\_ID, Country\_ID).

Index on Orders(Dep\_Date) to filter efficiently.

Partitioning:

If Days\_In\_Country or Orders contains a significant amount of data, consider partitioning these tables based on logical divisions like Dep\_Date (for Orders) or Order\_ID (for Days\_In\_Country).

Caching:

If this query or its result is frequently used, caching the results in a materialized view or temporary table can further improve performance.

Let me know if you’d like assistance in implementing any of these database changes!

You said:

add the creation of the indexes you relly on this code and send the whole code again

ChatGPT said:

Here's the optimized query along with the creation of the necessary indexes to improve performance. I've included the index creation statements before the query:

Index Creation

sql

Copy

Edit

-- Create indexes to optimize the query

-- Index on Days\_In\_Country for faster aggregation and filtering

CREATE INDEX IDX\_DaysInCountry\_OrderID\_CountryID ON Days\_In\_Country (Order\_ID, Country\_ID);

-- Index on Orders for filtering by Dep\_Date

CREATE INDEX IDX\_Orders\_DepDate ON Orders (Dep\_Date);

-- Index on Orders for joining with Days\_In\_Country

CREATE INDEX IDX\_Orders\_OrderID ON Orders (Order\_ID);

-- Index on Passengers for joining with Health\_Status

CREATE INDEX IDX\_Passengers\_ID ON Passengers (ID);

-- Index on Health\_Status for joining with Orders and Passengers

CREATE INDEX IDX\_HealthStatus\_OrderID\_ID ON Health\_Status (Order\_ID, ID);

Optimized Query

sql

Copy

Edit

-- Optimized Query for identifying high-value passengers with complex trips

SELECT

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

O.Order\_ID AS OrderID,

O.Dep\_Date AS DepartureDate,

O.Arriv\_Date AS ArrivalDate,

DATEDIFF(DAY, O.Dep\_Date, O.Arriv\_Date) AS TripDurationDays,

CountryStats.CountryCount -- Pre-computed country count

FROM

PASSENGERS P

INNER JOIN

Health\_Status HS

ON P.ID = HS.ID

INNER JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

-- Subquery to precompute country counts for each order

INNER JOIN (

SELECT

DIC.Order\_ID,

COUNT(DISTINCT DIC.Country\_ID) AS CountryCount

FROM

Days\_In\_Country DIC

GROUP BY

DIC.Order\_ID

HAVING

COUNT(DISTINCT DIC.Country\_ID) > 1

) AS CountryStats

ON O.Order\_ID = CountryStats.Order\_ID

WHERE

O.Dep\_Date >= '2025-01-01'

ORDER BY

CountryStats.CountryCount DESC,

TripDurationDays DESC;

Explanation of Index Usage

IDX\_DaysInCountry\_OrderID\_CountryID:

Speeds up the aggregation query in the subquery, where Order\_ID and Country\_ID are grouped and counted.

IDX\_Orders\_DepDate:

Optimizes filtering of orders based on Dep\_Date.

IDX\_Orders\_OrderID:

Ensures faster joins between Orders and the subquery.

IDX\_Passengers\_ID:

Speeds up the join between Passengers and Health\_Status.

IDX\_HealthStatus\_OrderID\_ID:

Optimizes the join between Health\_Status, Orders, and Passengers.

Benefits of Indexes

By applying these indexes, the query will process fewer rows and use efficient lookup operations, significantly improving its execution time.

Let me know if you need additional details or further optimizations!

You said:

now i want you to do the same improve in the time running efficient as you did above on this query:

-- Task 1 - Window Functions - Q5

-- Business Logic:

-- This query identifies orders with the highest total costs of extra insurance (summed for each order),

-- ranks the orders by their extra insurance costs within each year, and compares the cost of each order to the average for that year.

-- It also divides the orders into quartiles based on their costs, helping to identify high-value customers

-- and understand spending patterns on extra insurance products.

SELECT

O.order\_id AS OrderID, -- Order ID

YEAR(O.dep\_date) AS OrderYear, -- Order year

P.ID AS PassengerID, -- Passenger ID

P.First\_Name AS FirstName, -- Passenger first name

P.Last\_Name AS LastName, -- Passenger last name

SUM(C.Extra\_Price) AS TotalExtraInsuranceCost, -- Total extra insurance cost for the order

RANK() OVER (PARTITION BY YEAR(O.dep\_date) ORDER BY SUM(C.Extra\_Price) DESC) AS RankByYear, -- Rank orders by total extra insurance cost per year

AVG(SUM(C.Extra\_Price)) OVER (PARTITION BY YEAR(O.dep\_date)) AS AvgExtraInsuranceCostPerYear, -- Average extra insurance cost per year

NTILE(4) OVER (PARTITION BY YEAR(O.dep\_date) ORDER BY SUM(C.Extra\_Price) DESC) AS Quartile, -- Divide orders into quartiles based on total extra insurance cost

CASE

WHEN SUM(C.Extra\_Price) > AVG(SUM(C.Extra\_Price)) OVER (PARTITION BY YEAR(O.dep\_date)) THEN 'Above Average'

WHEN SUM(C.Extra\_Price) = AVG(SUM(C.Extra\_Price)) OVER (PARTITION BY YEAR(O.dep\_date)) THEN 'Average'

ELSE 'Below Average'

END AS PerformanceCategory -- Compare cost against the average

FROM

PASSENGERS P

JOIN

Health\_Status HS

ON P.ID = HS.ID

JOIN

Orders O

ON HS.Order\_ID = O.Order\_ID

JOIN

COVERS C

ON O.order\_id = C.order\_id -- Link extra insurance costs to the order

WHERE

C.Extra\_Price > 0 -- Include only orders with extra insurance costs

GROUP BY

O.order\_id, YEAR(O.dep\_date), P.ID, P.First\_Name, P.Last\_Name -- Group by order and year

ORDER BY

OrderYear, RankByYear;

ChatGPT said:

To optimize the running time of the second query while retaining its functionality, we focus on minimizing the computation involved in the window functions and reducing redundant calculations. Below is the improved query, along with the creation of the indexes used for optimization.

Index Creation

sql

Copy

Edit

-- Create indexes to optimize the query

-- Index on Orders for filtering and grouping by dep\_date

CREATE INDEX IDX\_Orders\_DepDate ON Orders (Dep\_Date);

-- Index on Orders for joining with Covers

CREATE INDEX IDX\_Orders\_OrderID ON Orders (Order\_ID);

-- Index on Covers for filtering and joining by Order\_ID and Extra\_Price

CREATE INDEX IDX\_Covers\_OrderID\_ExtraPrice ON Covers (Order\_ID, Extra\_Price);

-- Index on Passengers for joining with Health\_Status

CREATE INDEX IDX\_Passengers\_ID ON Passengers (ID);

-- Index on Health\_Status for joining with Orders and Passengers

CREATE INDEX IDX\_HealthStatus\_OrderID\_ID ON Health\_Status (Order\_ID, ID);

Optimized Query

sql

Copy

Edit

-- Optimized Query for identifying orders with the highest extra insurance costs

WITH ExtraInsuranceCosts AS (

-- Pre-compute total extra insurance costs per order

SELECT

C.Order\_ID,

YEAR(O.Dep\_Date) AS OrderYear,

SUM(C.Extra\_Price) AS TotalExtraInsuranceCost

FROM

Covers C

JOIN

Orders O

ON C.Order\_ID = O.Order\_ID

WHERE

C.Extra\_Price > 0 -- Include only orders with extra insurance costs

GROUP BY

C.Order\_ID, YEAR(O.Dep\_Date)

),

RankedOrders AS (

-- Rank orders by total extra insurance cost within each year

SELECT

EIC.Order\_ID,

EIC.OrderYear,

EIC.TotalExtraInsuranceCost,

RANK() OVER (PARTITION BY EIC.OrderYear ORDER BY EIC.TotalExtraInsuranceCost DESC) AS RankByYear,

AVG(EIC.TotalExtraInsuranceCost) OVER (PARTITION BY EIC.OrderYear) AS AvgExtraInsuranceCostPerYear,

NTILE(4) OVER (PARTITION BY EIC.OrderYear ORDER BY EIC.TotalExtraInsuranceCost DESC) AS Quartile

FROM

ExtraInsuranceCosts EIC

)

SELECT

R.Order\_ID AS OrderID,

R.OrderYear,

P.ID AS PassengerID,

P.First\_Name AS FirstName,

P.Last\_Name AS LastName,

R.TotalExtraInsuranceCost,

R.RankByYear,

R.AvgExtraInsuranceCostPerYear,

R.Quartile,

CASE

WHEN R.TotalExtraInsuranceCost > R.AvgExtraInsuranceCostPerYear THEN 'Above Average'

WHEN R.TotalExtraInsuranceCost = R.AvgExtraInsuranceCostPerYear THEN 'Average'

ELSE 'Below Average'

END AS PerformanceCategory

FROM

RankedOrders R

JOIN

Health\_Status HS

ON R.Order\_ID = HS.Order\_ID

JOIN

Passengers P

ON HS.ID = P.ID

ORDER BY

R.OrderYear, R.RankByYear;

Key Optimizations

Use of WITH Clauses:

Pre-compute TotalExtraInsuranceCost in the ExtraInsuranceCosts Common Table Expression (CTE) to minimize redundant calculations across the query.

Ranking and calculating averages/quartiles are handled in the RankedOrders CTE, reducing the load on the main query.

Minimized Joins:

Joins with Passengers and Health\_Status are performed only after the window functions and aggregations, reducing the number of rows processed.

Reduced Use of Window Functions:

Window functions like RANK, AVG, and NTILE operate on the pre-aggregated data in the RankedOrders CTE instead of recalculating aggregates multiple times.

Indexes:

Indexes on Orders.Dep\_Date, Covers.Order\_ID, Covers.Extra\_Price, and Passengers.ID ensure that filtering, joining, and grouping operations are efficient.

Early Filtering:

The WHERE C.Extra\_Price > 0 condition is applied before grouping in the ExtraInsuranceCosts CTE, reducing the rows processed in subsequent steps.

Explanation of Index Usage

IDX\_Orders\_DepDate:

Helps filter and group orders by YEAR(O.Dep\_Date) efficiently.

IDX\_Orders\_OrderID:

Optimizes joins between Orders and Covers.

IDX\_Covers\_OrderID\_ExtraPrice:

Speeds up filtering for Extra\_Price > 0 and the grouping of Order\_ID.

IDX\_Passengers\_ID:

Ensures efficient joining between Passengers and Health\_Status.

IDX\_HealthStatus\_OrderID\_ID:

Facilitates joins between Health\_Status, Orders, and Passengers.

By applying these optimizations and indexes, the query execution time will be significantly improved, especially for large datasets. Let me know if you’d like help implementing or testing this!

## פרק שני - בונוס

## מטלה 5 – יישום כלים נוספים בלימוד עצמי

יישום שאילתה עם טבלה זמנית

--The business idea is to enable the manager to assign employees to upcoming shifts in a convenient, organized, and responsible manner.

CREATE PROCEDURE dbo.sp\_CalculateTravelersAndWorkersByDate

(

@InputDate DATE

)

AS

BEGIN

CREATE TABLE #TravelersCount

(

TravelersInAir INT

);

INSERT INTO #TravelersCount

SELECT

COUNT(DISTINCT P.ID) AS TravelersInAir

FROM

PASSENGERS P

JOIN HEALTH\_STATUS HS

ON P.ID = HS.ID

JOIN ORDERS O

ON HS.order\_id = O.order\_id

WHERE

@InputDate BETWEEN O.dep\_date AND O.arriv\_date;

DECLARE @ActiveTravelers INT;

SELECT

@ActiveTravelers = TravelersInAir

FROM

#TravelersCount;

DECLARE @RequiredWorkers INT;

SET @RequiredWorkers = CASE

WHEN @ActiveTravelers <= 50 THEN 2

WHEN @ActiveTravelers BETWEEN 51 AND 100 THEN 4

ELSE 6

END;

SELECT

@ActiveTravelers AS ActiveTravelers,

@RequiredWorkers AS RequiredWorkers;

END;

--input

EXEC dbo.sp\_CalculateTravelersAndWorkersByDate @InputDate = '2025-03-20';

יישום של [**Error Handling with TRY...CATCH**](https://www.sqlservertutorial.net/sql-server-stored-procedures/sql-server-try-catch/)

-- second q with Error Handling with TRY...CATCH

-- this code is meant to verify that the departure date is before the arrival date

IF NOT EXISTS (SELECT 1 FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = 'ErrorLogs')

BEGIN

CREATE TABLE ErrorLogs (

LogID INT IDENTITY(1,1) PRIMARY KEY,

ErrorMessage NVARCHAR(4000) NOT NULL,

ErrorSeverity INT NOT NULL,

ErrorState INT NOT NULL,

LogDate DATETIME NOT NULL DEFAULT GETDATE()

);

END;

CREATE PROCEDURE dbo.sp\_InsertValidatedOrder

(

@OrderID VARCHAR(50),

@PassengerID VARCHAR(50),

@DepDate DATE,

@ArrivDate DATE,

@CreditCardNumber VARCHAR(16),

@Payments SMALLINT

)

AS

BEGIN

BEGIN TRY

-- בדיקות לוגיות נוספות

IF @DepDate > @ArrivDate

THROW 50001, 'Departure date cannot be later than arrival date.', 1;

IF LEN(@CreditCardNumber) <> 16 OR ISNUMERIC(@CreditCardNumber) = 0

THROW 50002, 'Invalid credit card number format.', 1;

-- הוספת נתונים לטבלאות

INSERT INTO Orders (Order\_ID, Order\_Date, Dep\_Date, Arriv\_Date, Credit\_Card\_Number, Payments)

VALUES (@OrderID, GETDATE(), @DepDate, @ArrivDate, @CreditCardNumber, @Payments);

PRINT 'Order inserted successfully.';

END TRY

BEGIN CATCH

-- טיפול בשגיאה

DECLARE @ErrorMessage NVARCHAR(4000) = ERROR\_MESSAGE();

DECLARE @ErrorSeverity INT = ERROR\_SEVERITY();

DECLARE @ErrorState INT = ERROR\_STATE();

-- שמירת השגיאה בטבלת לוגים

INSERT INTO ErrorLogs (ErrorMessage, ErrorSeverity, ErrorState, LogDate)

VALUES (@ErrorMessage, @ErrorSeverity, @ErrorState, GETDATE());

-- הצגת הודעה

PRINT 'Error occurred: ' + @ErrorMessage;

END CATCH;

END;

--good check

EXEC dbo.sp\_InsertValidatedOrder

@OrderID = '81',

@PassengerID = '1000000789',

@DepDate = '2025-02-21',

@ArrivDate = '2025-02-23',

@CreditCardNumber = '5410000000000000',

@Payments = 2;

-- bad check

EXEC dbo.sp\_InsertValidatedOrder

@OrderID = '2',

@PassengerID = '1000000002',

@DepDate = '2025-01-20',

@ArrivDate = '2025-01-15',

@CreditCardNumber = '1234567812345678',

@Payments = 2;