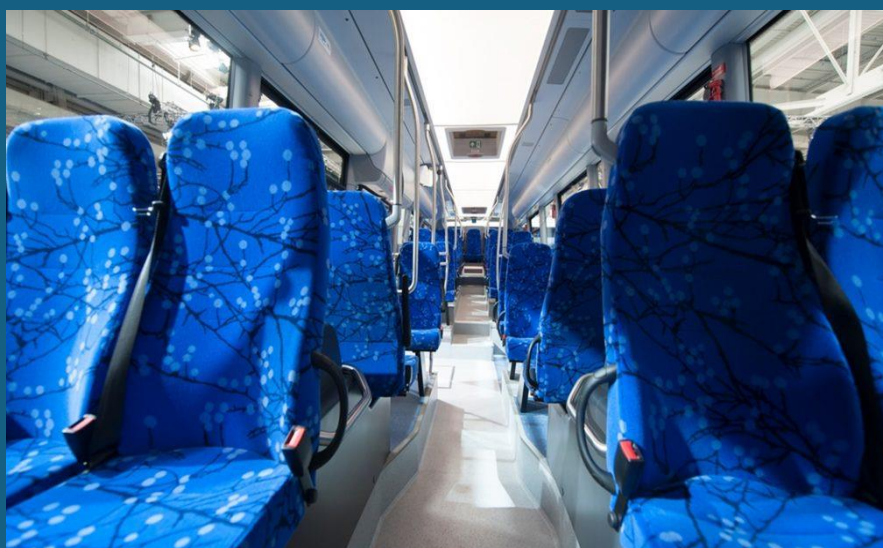


מגשים: אורי  
פרלמוטר ואיתן  
קנטמן

# פרויקט בסיסי נתונים – חברת הסעות – אגף כרטיסים



## אגף הכרטיסים – חברת הסעות

3	שלב 1
3	פירוט על הפרויקט והישויות והיחסים
3	מטרות המערכת:
3	פירוט ישויות ויחסים:
4	תרשים ERD:
4	תרשים DSD:
5	יצירת הטבלאות:
6	סדר מחיקת הטבלאות:
6	INSERT TABLE
7	הצגת הנתונים דרך SELECT:
10	Desc
13	הכנסת הנתונים – כל סכמה עם לפחות 400 שורות.
13	דרך א' – DATA GENERATOR
17	דרך ב' – Mockoro
20	דרך שלישית – הכנסה באמצעות פקודות פייתון
21	גיבוי ושחזור:
22	פתיחת הגיבוי:
23	חלק ב' – שאילתות
23	שאילתות בלי פרמטרים:
23	שאילתות SELECT:
26	delete
27	Update
28	שאילתות פרמטרים:
30	אילוצים
33	שלב 3 – פונקציות ופרוצדורות:
33	פונ' 1: חישוב רווח כולל בטווח תאריכים מסוים:
33	פונ' 2: מספר האנשים ששם המשפחה שלהם מתחיל באות מסוימת.
34	פרוצדורה 1: חישוב המשקל הכולל
35	פרוצדורה 2: חישוב עלות כל סוג כרטיס ועדכון המחיר לכל סוג.
36	Main1:
36	Main2:
39	שלב 4 – אינטגרציה
39	אינטגרציה ברמת העיצוב
41	ERD משותף
42	DSD משותף
43	הסבר שינויים ואינטגרציה

43.....	פירוט המימד הטכני של השינויים.....
49.....	אינטגרציה – רמת התשאול.....
50.....	מבט על האגף החדש.....

## שלב 1

### פירוט על הפרויקט והישויות והיחסים

הפרויקט עליו אנו עובדים הוא מערכת לניהול והזמנה של כרטיסי נסיעה בתחבורה ציבורית. מערכת זו כוללת מספר ישויות עיקריות כמו נוסעים, מוכרי כרטיסים, כרטיסים, כבודה, הזמנות ודוחות תשלום. המערכת מאפשרת למשתמשים לנהל את המידע הנוגע לנוסעים, למכור כרטיסים, לנהל כבודה, לבצע הזמנות ולהפיק דוחות תשלום.

### מטרות המערכת:

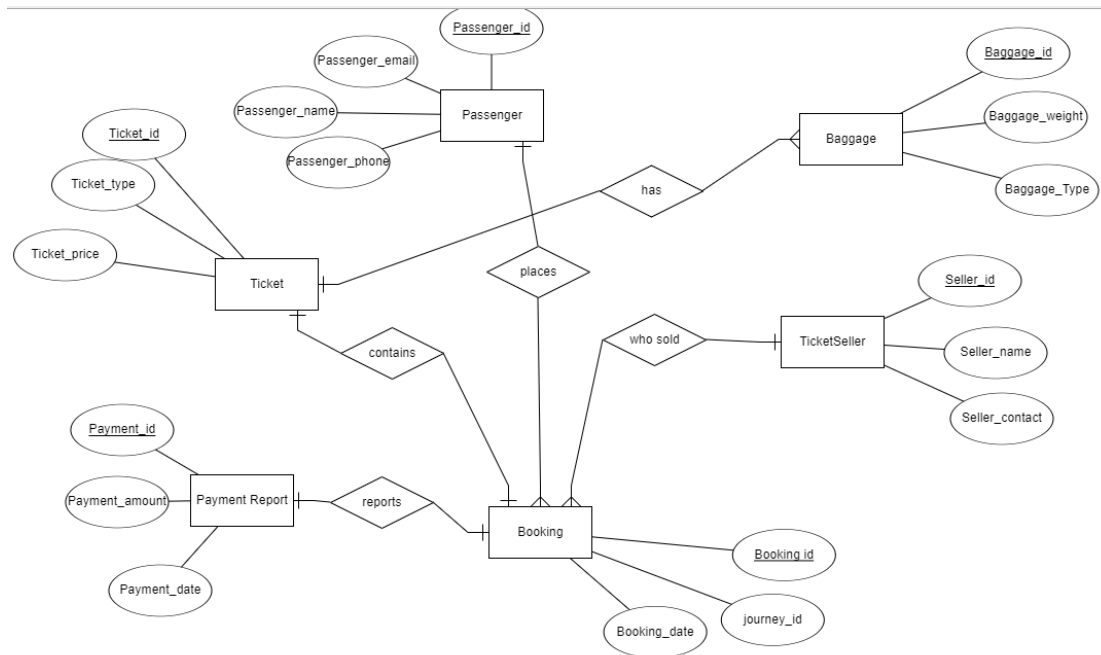
ניהול נוסעים: שמירת מידע אישי של נוסעים הכולל שם, טלפון ודואר אלקטרוני.  
ניהול מוכרי כרטיסים: שמירת מידע אישי של מוכרי כרטיסים כולל שם ופרטי קשר.  
ניהול כרטיסים: שמירת פרטי הכרטיסים כמו סוג הכרטיס ומחירו.  
ניהול כבודה: שמירת מידע אודות סוג הכבודה, משקלה וקשירתה לכרטיס הנסיעה.  
ניהול הזמנות: מעקב אחר פרטי ההזמנות הכוללים תאריך הזמנה, מזהה נוסע, מזהה מוכר ומזהה כרטיס.  
ניהול דוחות תשלום: מעקב אחר פרטי התשלומים הכוללים סכום, תאריך התשלום ומזהה ההזמנה.

### פירוט ישויות ויחסים:

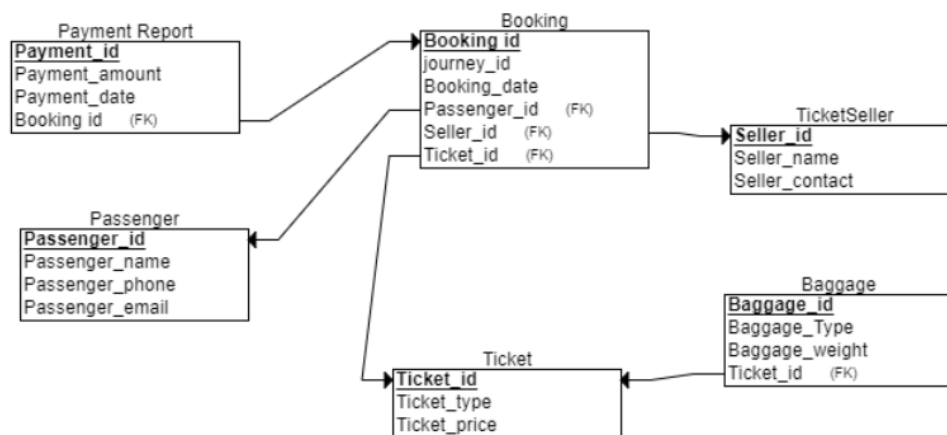
Passenger (נוסע): שומר את פרטי הנוסעים כמו שם, טלפון ודוא"ל.  
TicketSeller (מוכר כרטיסים): שומר את פרטי המוכרים כמו שם ופרטי קשר.  
Ticket (כרטיס): כולל פרטים על סוג הכרטיס ומחירו.  
Baggage (כבודה): כולל פרטים על סוג הכבודה ומשקלה, מקושר לכרטיס נסיעה.  
Booking (הזמנה): כולל פרטים על הזמנות כמו תאריך ההזמנה ומזהי נוסע, מוכר וכרטיס.  
Payment\_Report (דו"ח תשלום): כולל פרטים על תשלומים כמו סכום התשלום, תאריך ומזהה הזמנה.  
דוגמה לפרויקט:

המערכת נועדה לסייע לחברת תחבורה ציבורית בניהול כל המידע הקשור לנוסעים, מכירת כרטיסים, הזמנות ותשלומים בצורה ממוחשבת ויעילה. המערכת תאפשר לנוסעים לרכוש כרטיסים בצורה נוחה ולחברה לנהל את כל המידע בצורה מרוכזת ומסודרת.

## תרשים ERD:



## תרשים DSD:



כל הטבלאות הינן ביחס של 3NF ואין צורך בנרמול, נוכיח זאת:

כל הטבלאות עומדות ביחס של 1NF מפני שכל השדות אטומיים

כל הטבלאות עומדות ב2NF משום שכולן המפתח הוא רק שדה אחד ולא תתכן תלות בחלק מן המפתח אלא בכולו

כל הטבלאות עומדות ב3NF משום שאין קשר בין השדות השונים, אלא רק ע"י שדה המפתח.

## יצירת הטבלאות:

```
CREATE TABLE Passenger
(
    Passenger_id NUMBER (38) NOT NULL,
    Passenger_name VARCHAR2 (255) NOT NULL,
    Passenger_phone VARCHAR2 (15) NOT NULL,
    Passenger_email VARCHAR2 (255) NOT NULL,
    PRIMARY KEY (Passenger_id)
);

CREATE TABLE TicketSeller
(
    Seller_id NUMBER (38) NOT NULL,
    Seller_name VARCHAR2 (255) NOT NULL,
    Seller_contact VARCHAR2 (255) NOT NULL,
    PRIMARY KEY (Seller_id)
);

CREATE TABLE Ticket
(
    Ticket_id NUMBER (38) NOT NULL,
    Ticket_type VARCHAR2 (255) NOT NULL,
    Ticket_price FLOAT NOT NULL,
    PRIMARY KEY (Ticket_id)
);
```

```
CREATE TABLE Baggage
(
    Baggage_id NUMBER (38) NOT NULL,
    Baggage_Type VARCHAR2 (255) NOT NULL,
    Baggage_weight FLOAT NOT NULL,
    Ticket_id NUMBER (38) NOT NULL,
    PRIMARY KEY (Baggage_id),
    FOREIGN KEY (Ticket_id) REFERENCES Ticket (Ticket_id)
);
```

```
CREATE TABLE Booking
(
    Booking_id NUMBER (38) NOT NULL,
    journey_id NUMBER (38) NOT NULL,
    Booking_date DATE NOT NULL,
    Passenger_id NUMBER (38) NOT NULL,
    Seller_id NUMBER (38) NOT NULL,
    Ticket_id NUMBER (38) NOT NULL,
    PRIMARY KEY (Booking_id),
    FOREIGN KEY (Passenger_id) REFERENCES Passenger(Passenger_id),
    FOREIGN KEY (Seller_id) REFERENCES TicketSeller(Seller_id),
    FOREIGN KEY (Ticket_id) REFERENCES Ticket(Ticket_id)
);

CREATE TABLE Payment_Report
(
    Payment_id NUMBER (38) NOT NULL,
    Payment_amount FLOAT NOT NULL,
    Payment_date DATE NOT NULL,
    Booking_id NUMBER (38) NOT NULL,
    PRIMARY KEY (Payment_id),
    FOREIGN KEY (Booking_id) REFERENCES Booking(Booking_id)
);
```

## סדר מחיקת הטבלאות:

SQL	Output	Statistics
<pre>-- Drop tables in reverse order of their dependencies DROP TABLE Payment_Report; DROP TABLE Booking; DROP TABLE Baggage; DROP TABLE Ticket; DROP TABLE TicketSeller; DROP TABLE Passenger;</pre>		

## INSERT TABLE

```
-- Inserting Data
INSERT INTO Passenger (Passenger_id, Passenger_name, Passenger_phone, Passenger_email)
VALUES (1, 'John Doe', '1234567890', 'john.doe@example.com');

INSERT INTO Passenger (Passenger_id, Passenger_name, Passenger_phone, Passenger_email)
VALUES (2, 'Jane Smith', '0987654321', 'jane.smith@example.com');

INSERT INTO TicketSeller (Seller_id, Seller_name, Seller_contact)
VALUES (1, 'Alice Johnson', 'alice.johnson@example.com');

INSERT INTO TicketSeller (Seller_id, Seller_name, Seller_contact)
VALUES (2, 'Bob Brown', 'bob.brown@example.com');
```

```
INSERT INTO Ticket (Ticket_id, Ticket_type, Ticket_price)
VALUES (1, 'Single', 2.50);
```

```
INSERT INTO Ticket (Ticket_id, Ticket_type, Ticket_price)
VALUES (2, 'Return', 4.00);
```

```
INSERT INTO Baggage (Baggage_id, Baggage_Type, Baggage_weight, Ticket_id)
VALUES (1, 'Cabin', 7.5, 1);
```

```
INSERT INTO Baggage (Baggage_id, Baggage_Type, Baggage_weight, Ticket_id)
VALUES (2, 'Checked', 15.0, 2);
```

```
INSERT INTO Booking (Booking_id, journey_id, Booking_date, Passenger_id, Seller_id, Ticket_id)
VALUES (1, 101, TO_DATE('2024-05-01', 'YYYY-MM-DD'), 1, 1, 1);
```

```
INSERT INTO Booking (Booking_id, journey_id, Booking_date, Passenger_id, Seller_id, Ticket_id)
VALUES (2, 102, TO_DATE('2024-05-02', 'YYYY-MM-DD'), 2, 2, 2);
```

```
INSERT INTO Payment_Report (Payment_id, Payment_amount, Payment_date, Booking_id)
VALUES (1, 2.50, TO_DATE('2024-05-01', 'YYYY-MM-DD'), 1);
```

```
INSERT INTO Payment_Report (Payment_id, Payment_amount, Payment_date, Booking_id)
VALUES (2, 4.00, TO_DATE('2024-05-02', 'YYYY-MM-DD'), 2);
```

## הצגת הנתונים דרך SELECT:

```
-- Select Data
SELECT * FROM Passenger;
SELECT * FROM TicketSeller;
SELECT * FROM Ticket;
SELECT * FROM Baggage;
SELECT * FROM Booking;
SELECT * FROM Payment_Report;
```

Insert payment\_report Insert payment\_report Commit Select passenger Select ticketseller Select ticket Select baggage Select booking Select payment\_report

	PASSENGER_ID	PASSENGER_NAME	PASSENGER_PHONE	PASSENGER_EMAIL
1	1	John Doe	1234567890	john.doe@example.com
2	2	Jane Smith	0987654321	jane.smith@example.com



dropTables.sql sel1.sql createTables.sql

SQL Output Statistics

```
-- Select all records from each table
SELECT * FROM Passenger;

SELECT * FROM TicketSeller;

SELECT * FROM Ticket;

SELECT * FROM Baggage;

SELECT * FROM Booking;

SELECT * FROM Payment_Report;
```

Select passenger Select ticketseller Select ticket Select baggage Select booking Select payment\_report

	SELLER_ID	SELLER_NAME	SELLER_CONTACT
1	1	Alice Johnson	alice.johnson@example.com
2	2	Bob Brown	bob.brown@example.com

dropTables.sql sel1.sql createTables.sql

SQL Output Statistics

```
-- Select all records from each table
SELECT * FROM Passenger;

SELECT * FROM TicketSeller;

SELECT * FROM Ticket;

SELECT * FROM Baggage;

SELECT * FROM Booking;

SELECT * FROM Payment_Report;
```

Select passenger Select ticketseller Select ticket Select baggage Select booking Select payment\_report

	TICKET_ID	TICKET_TYPE	TICKET_PRICE
1	1	Single	2.5
2	2	Return	4

dropTables.sql
 sel1.sql
 
 createTables.sql

SQL
 

Output

Statistics

```

-- Select all records from each table
SELECT * FROM Passenger;

SELECT * FROM TicketSeller;

SELECT * FROM Ticket;

SELECT * FROM Baggage;

SELECT * FROM Booking;

SELECT * FROM Payment_Report;
  
```

Select passenger
 Select ticketseller
 Select ticket
 **Select baggage**
 Select booking
 Select payment\_report

	BAGGAGE_ID	BAGGAGE_TYPE	BAGGAGE_WEIGHT	TICKET_ID
▶ 1	1	Cabin	7.5	1
▶ 2	2	Checked	15	2

dropTables.sql | sel1.sql | createTables.sql

SQL Output Statistics

```
-- Select all records from each table
SELECT * FROM Passenger;

SELECT * FROM TicketSeller;

SELECT * FROM Ticket;

SELECT * FROM Baggage;

SELECT * FROM Booking;

SELECT * FROM Payment_Report;
```

Select passenger | Select ticketseller | Select ticket | Select baggage | **Select booking** | Select payment\_report

	BOOKING_ID	JOURNEY_ID	BOOKING_DATE	PASSENGER_ID	SELLER_ID	TICKET_ID
▶ 1	1	101	01/05/2024	1	1	1
2	2	102	02/05/2024	2	2	2

The screenshot shows a SQL IDE interface with three tabs: dropTables.sql, sel1.sql, and createTables.sql. The 'sel1.sql' tab is active, displaying the following SQL commands:

```
-- Select all records from each table
SELECT * FROM Passenger;

SELECT * FROM TicketSeller;

SELECT * FROM Ticket;

SELECT * FROM Baggage;

SELECT * FROM Booking;

SELECT * FROM Payment_Report;
```

Below the query window, there are tabs for selecting data from different tables: Select passenger, Select ticketseller, Select ticket, Select baggage, Select booking, and Select payment\_report. The 'Select payment\_report' tab is active, showing a results grid with the following data:

	PAYMENT_ID	PAYMENT_AMOUNT	PAYMENT_DATE	BOOKING_ID
1	1	2.5	01/05/2024	1
2	2	4	02/05/2024	2

Desc

The screenshot shows a terminal window with the command `desc Passenger;` and its output. The output is formatted as a table with two columns: Field and Type.

Field	Type
Passenger_id	NUMBER(38)
Passenger_name	VARCHAR2(255)
Passenger_phone	VARCHAR2(15)
Passenger_email	VARCHAR2(255)

```
desc TicketSeller;
```

Field	Type
-----	-----
Seller_id	NUMBER(38)
Seller_name	VARCHAR2(255)
Seller_contact	VARCHAR2(255)

```
desc Ticket;
```

Field	Type
-----	-----
Ticket_id	NUMBER(38)
Ticket_type	VARCHAR2(255)
Ticket_price	FLOAT

```
desc Baggage;
```

Field	Type
-----	-----
Baggage_id	NUMBER(38)
Baggage_Type	VARCHAR2(255)
Baggage_weight	FLOAT
Ticket_id	NUMBER(38)

```
desc Booking;
```

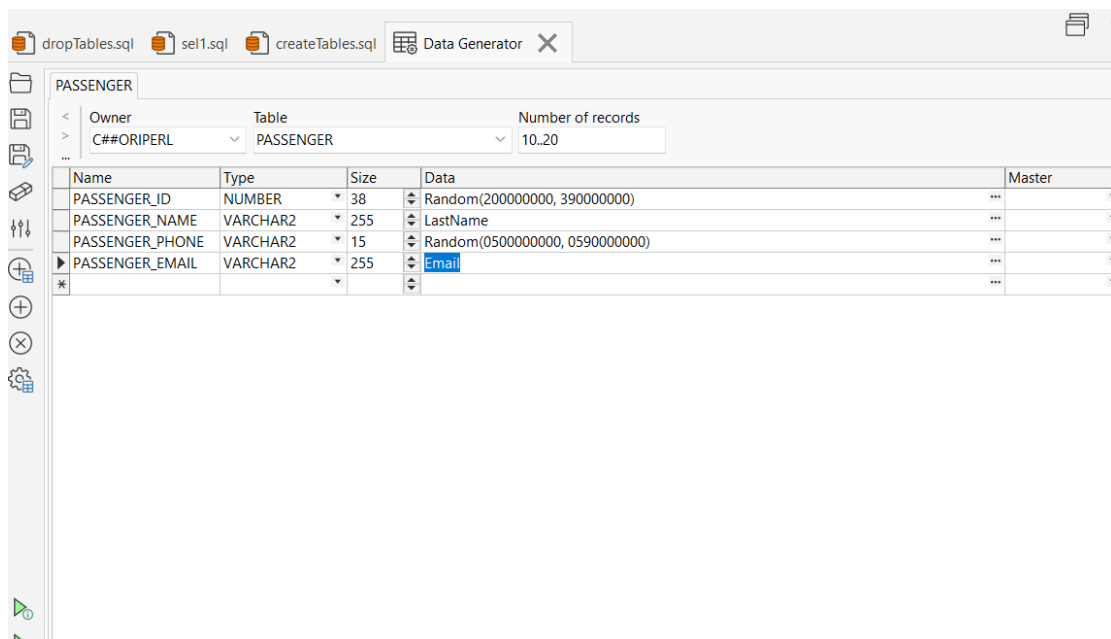
Field	Type
-----	-----
Booking_id	NUMBER(38)
journey_id	NUMBER(38)
Booking_date	DATE
Passenger_id	NUMBER(38)
Seller_id	NUMBER(38)
Ticket_id	NUMBER(38)

```
desc Payment_Report;
```

Field	Type
Payment_id	NUMBER(38)
Payment_amount	FLOAT
Payment_date	DATE
Booking_id	NUMBER(38)

הכנסת הנתונים – כל סכמה עם לפחות 400 שורות.

## DATA GENERATOR – דרך א'



BAGGAGE

< Owner Table Number of records  
> C#ORIPERL BAGGAGE 10..20

Name	Type	Size	Data	Master
BAGGAGE_ID	NUMBER	38	Random(10000,100000)	...
BAGGAGE_TYPE	VARCHAR2	255	List('Cabin', 'Checked', 'Bag')	...
BAGGAGE_WEIGHT	FLOAT	22	Random(0.1, 10)	...
TICKET_ID	NUMBER	38	List(select Ticket_id from Ticket)	...
*				...

dropTables.sql sel1.sql createTables.sql Data Generator X

TICKET

< Owner Table Number of records  
> C#ORIPERL TICKET 300..350

Name	Type	Size	Data	Master
TICKET_ID	NUMBER	38	Random(200000, 390000)	...
TICKET_TYPE	VARCHAR2	255	List('regular', 'handicapped', 'special')	...
TICKET_PRICE	FLOAT	22	List('1', '5', '2.5')	...
*				...

Definition Options Result

dropTables.sql sel1.sql createTables.sql Data Generator X

TICKETSELLER

< Owner Table Number of records  
> C#ORIPERL TICKETSELLER 10..20

Name	Type	Size	Data	Master
SELLER_ID	NUMBER	38	Random(200000000, 3900000000)	...
SELLER_NAME	VARCHAR2	255	LastName	...
✓ SELLER_CONTACT	VARCHAR2	255	Random(0500000000, 05900000000)	...
*				...

Definition Options Result

dropTables.sql sel1.sql createTables.sql Data Generator X

BOOKING

< Owner Table Number of records  
> C#ORIPERL BOOKING 300..400

Name	Type	Size	Data	Master
BOOKING_ID	NUMBER	38	Random(200000, 990000)	...
JOURNEY_ID	NUMBER	38	Random(200000, 990000)	...
BOOKING_DATE	DATE		Random(01.1.2024, 30.12.2024)	...
PASSENGER_ID	NUMBER	38	List(select Passenger_id from Passenger)	...
SELLER_ID	NUMBER	38	Random(200000000, 3900000000)	...
TICKET_ID	NUMBER	38	List(select Ticket_id from Ticket)	...
*				...



dropTables.sql sel1.sql createTables.sql Data Generator X

PAYMENT\_REPORT

< Owner Table Number of records  
> C#ORIPERL PAYMENT\_REPORT 10..20  
...

Name	Type	Size	Data	Master
PAYMENT_ID	NUMBER	38	Random(200000000, 390000000)	...
PAYMENT_DATE	DATE		Random(0.21.2024, 30.12.2024)	...
BOOKING_ID	NUMBER	38	List(select booking_id from Booking)	...
*				...

Definition Options Result

dropTables.sql sel1.sql createTables.sql Data Generator Data Generator X

BOOKING

< Owner Table Number of records  
> C#ORIPERL BOOKING 10..20  
...

Name	Type	Size	Data	Master
BOOKING_ID	NUMBER	38	Random(200000000, 990000000)	...
JOURNEY_ID	NUMBER	38	Random(200000000, 990000000)	...
BOOKING_DATE	DATE		Random(0.21.2024, 30.12.2024)	...
PASSENGER_ID	NUMBER	38	List(select passenger_id from Passenger)	...
SELLER_ID	NUMBER	38	List(select seller_id from Seller)	...
TICKET_ID	NUMBER	38	List(select ticket_id from Ticket)	...
*				...

## Mockoro – ברך ב'

dr1.sql
sel1.sql
createTables.sql
MOCK\_pass.csv

Data from Textfile
Data to Oracle

**General**

Owner
Table
PASSENGER
Initializing Script

Commit every...
100
☒ Overwrite duplicates
☐ Ignore duplicates
☐ Delete records
☐ Truncate table
Finalizing Script

**Fields**

Field1 Passenger\_id ->
Field2 Passenger\_name
Field3 Passenger\_phone
Field4 Passenger\_email
Field
Fieldtype

**Result Preview**

Passenger_id	Passenger_name	Passenger_phone	Passenger_email
300000038	Balmer	311-168-8736	tbalmer12@trellian.com
300000039	Cottem	641-721-4034	tcottem13@eventbrite.com

Import
Import to Script
Close
c##oriperl@XE
[11:13:59] 70 rec
Help

dr1.sql
sel1.sql
createTables.sql
MOCK\_seller.csv

Data from Textfile
Data to Oracle

**General**

Owner
Table
TICKETSELLER
Initializing Script

Commit every...
100
☒ Overwrite duplicates
☐ Ignore duplicates
☐ Delete records
☐ Truncate table
Finalizing Script

**Fields**

Field1 Seller\_id -> SELL
Field2 Seller\_name -> S
Field3 Seller\_contact ->
Field
Fieldtype

**Result Preview**

Seller_id	Seller_name	Seller_contact
310000000	Harroll	406-630-9267
310000001	Aronin	864-698-0298

Import
Import to Script
Close
c##oriperl@XE
MOCK\_seller.csv
Help

dr1.sql sel1.sql createTables.sql MOCK\_payment.csv

Data from Textfile Data to Oracle

### General

Owner:  Table: PAYMENT\_REPORT

Commit every...: 100

☒ Overwrite duplicates ☐ Delete records ☐ Truncate table

Initializing Script:  ...

Finalizing Script:  ...

### Fields

Field1 Payment\_id -> PAYMENT\_ID  
Field2 Payment\_date -> PAYMENT\_DATE  
Field3 Booking\_id -> BOOKING\_ID

Field:   
Fieldtype:

### Result Preview

Payment_id	Payment_date	Booking_id
1042	7/24/2023	1042
1043	7/7/2023	1043

Import Import to Script Close c##oriperl@XE [14:01:14] 404 re Help

dr1.sql sel1.sql createTables.sql MOCK\_booking.csv

Data from Textfile Data to Oracle

### General

Owner:  Table: BOOKING

Commit every...: 100

☒ Overwrite duplicates ☐ Delete records ☐ Truncate table

Initializing Script:  ...

Finalizing Script:  ...

### Fields

Field1 Booking\_id -> BOOKING\_ID  
Field2 Journey\_id -> JOURNEY\_ID  
Field3 Booking\_date -> BOOKING\_DATE  
Field4 Passenger\_id -> PASSENGER\_ID

Field:   
Fieldtype:

### Result Preview

Booking_id	Journey_id	Booking_date	Passenger_id	Seller_id	Ticket_id
1000	1000	06/08/2023	1573871528	234413401	200137
1001	1001	6/24/2023	4139544263	214945813	200361

Import Import to Script Close c##oriperl@XE MOCK\_booking.c Help

dr1.sql sel1.sql createTables.sql MOCK\_ticket.csv

Data from Textfile Data to Oracle

**General**

Owner:  Table: **TICKET**

Commit every...: 100

☒ Overwrite duplicates ☐ Delete records ☐ Truncate table

☐ Ignore duplicates

Initializing Script:  ...

Finalizing Script:  ...

**Fields**

Field1 Ticket\_id -> TICKET\_ID  
Field2 Ticket\_type -> TICKET\_TYPE  
Field3 Ticket\_price -> TICKET\_PRICE

Field:   
Fieldtype:

**Result Preview**

Ticket_id	Ticket_type	Ticket_price
359768	special	10
359769	regular	1

Import Import to Script Close c##oriperl@XE MOCK\_ticket.csv Help

dr1.sql sel1.sql createTables.sql MOCK\_baggage1.csv

Data from Textfile Data to Oracle

**General**

Owner:  Table: **BAGGAGE**

Commit every...: 100

☒ Overwrite duplicates ☐ Delete records ☐ Truncate table

☐ Ignore duplicates

Initializing Script:  ...

Finalizing Script:  ...

**Fields**

Field1 Baggage\_id -> BAGGAGE\_ID  
Field2 Baggage\_Type -> BAGGAGE\_TYPE  
Field3 Baggage\_weight -> BAGGAGE\_WEIGHT  
Field4 Ticket\_id -> TICKET\_ID

Field:   
Fieldtype:

**Result Preview**

Baggage_id	Baggage_Type	Baggage_weight	Ticket_id
1000	Bag	1	359768
1001	Bag	2	359769

Import Import to Script Close c##oriperl@XE MOCK\_baggage1 Help

## דרך שלישית – הכנסה באמצעות פקודות פיתוח

```
import random
from faker import Faker

# Initialize Faker
fake = Faker()

# Define the number of records to generate
num_records = 20

# Open the file to write SQL insert statements
with open("insertData.sql", "w") as f:
    # Write SQL insert statements for Passenger
    f.write("-- Generating SQL insert statements for Passenger table\n")
    for i in range(num_records):
        passenger_id = 100000 + i
        passenger_name = fake.name().replace("'", "'")
        passenger_phone = fake.phone_number().replace("'", "'")
        passenger_email = fake.email().replace("'", "'")
        f.write(f"INSERT INTO Passenger (Passenger_id, Passenger_name, Passenger_phone, Passenger_email) VALUES ({passenger_id}, '{passenger_name}, '{passenger_phone}, '{passenger_email}')
```

```
# Write SQL insert statements for TicketSeller
f.write("\n-- Generating SQL insert statements for TicketSeller table\n")
for i in range(num_records):
    seller_id = 200000 + i
    seller_name = fake.name().replace("'", "'")
    seller_contact = fake.email().replace("'", "'")
    f.write(f"INSERT INTO TicketSeller (Seller_id, Seller_name, Seller_contact) VALUES ({seller_id}, '{seller_name}', '{seller_contact}');\n")

# Write SQL insert statements for Ticket
f.write("\n-- Generating SQL insert statements for Ticket table\n")
for i in range(num_records):
    ticket_id = 300000 + i
    ticket_type = random.choice(['Single', 'Return']).replace("'", "'")
    ticket_price = round(random.uniform(1.0, 50.0), 2)
    f.write(f"INSERT INTO Ticket (Ticket_id, Ticket_type, Ticket_price) VALUES ({ticket_id}, '{ticket_type}', {ticket_price});\n")

# Write SQL insert statements for Baggage
f.write("\n-- Generating SQL insert statements for Baggage table\n")
for i in range(num_records):
    baggage_id = 400000 + i
    baggage_type = random.choice(['Cabin', 'Checked']).replace("'", "'")
    baggage_weight = round(random.uniform(5.0, 30.0), 2)
    ticket_id = random.randint(300000, 300000 + num_records - 1)
    f.write(f"INSERT INTO Baggage (Baggage_id, Baggage_type, Baggage_weight, Ticket_id) VALUES ({baggage_id}, '{baggage_type}', {baggage_weight}, {ticket_id});\n")
```

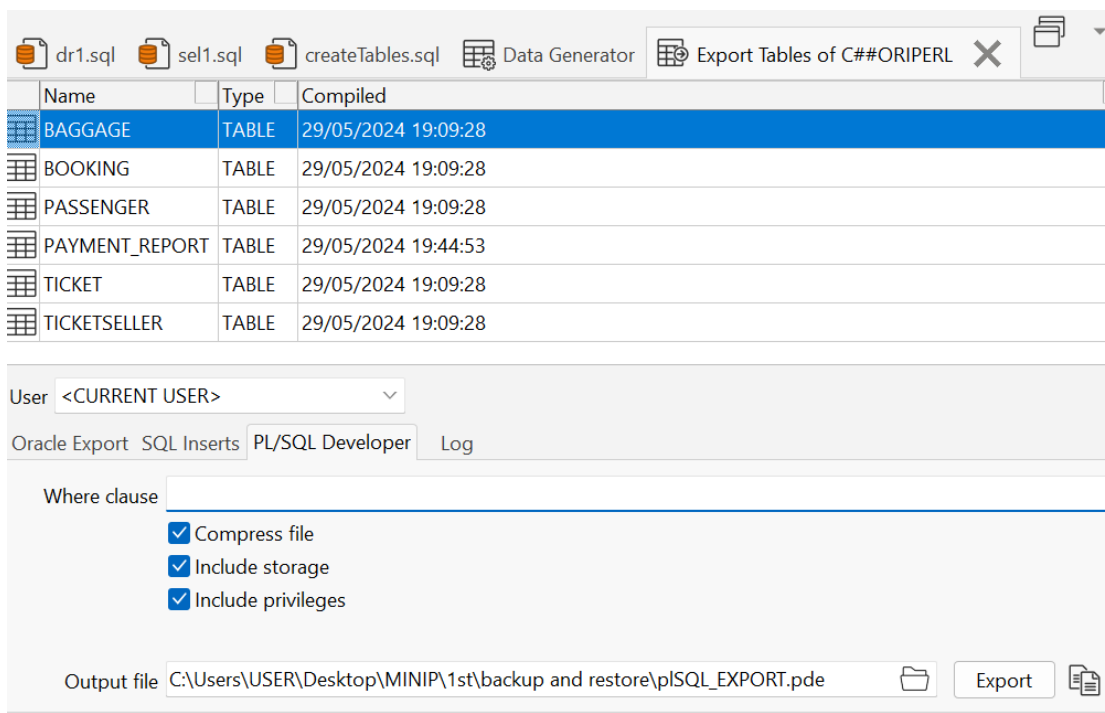
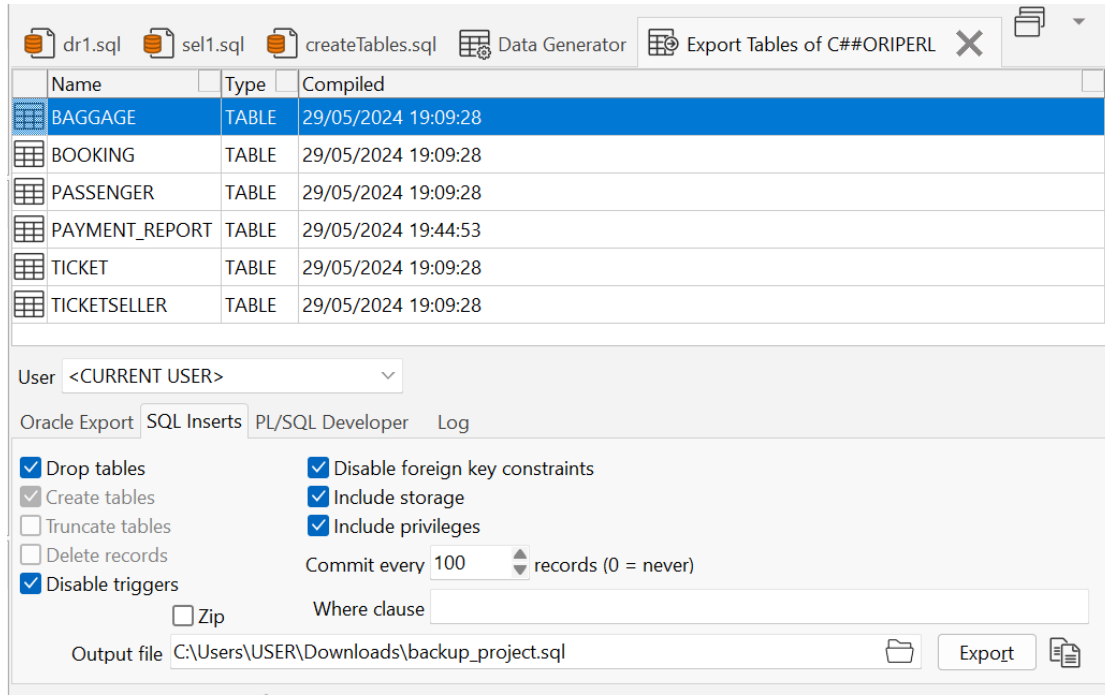
```
# Write SQL insert statements for Booking
f.write("\n-- Generating SQL insert statements for Booking table\n")
for i in range(num_records):
    booking_id = 500000 + i
    journey_id = random.randint(600000, 600000 + num_records - 1)
    booking_date = fake.date()
    passenger_id = random.randint(100000, 100000 + num_records - 1)
    seller_id = random.randint(200000, 200000 + num_records - 1)
    ticket_id = random.randint(300000, 300000 + num_records - 1)
    f.write(f"INSERT INTO Booking (Booking_id, Journey_id, Booking_date, Passenger_id, Seller_id, Ticket_id) VALUES ({booking_id}, {journey_id}, '{booking_date}', {passenger_id}, {seller_id}, {ticket_id});\n")

# Write SQL insert statements for Payment_Report
f.write("\n-- Generating SQL insert statements for Payment_Report table\n")
for i in range(num_records):
    payment_id = 700000 + i
    payment_amount = round(random.uniform(1.0, 100.0), 2)
    payment_date = fake.date()
    booking_id = random.randint(500000, 500000 + num_records - 1)
    f.write(f"INSERT INTO Payment_Report (Payment_id, Payment_amount, Payment_date, Booking_id) VALUES ({payment_id}, {payment_amount}, '{payment_date}', {booking_id});\n")
```

## גיבוי ושחזור:

C:\Users\USER\Downloads\backup\_project.sql

## 2 דרכי הגיבוי



## פתיחת הגיבוי:

```

SQL      Output  Statistics
-- Enabling triggers for Passenger table
ALTER TABLE Passenger ENABLE ALL TRIGGERS;

-- Enabling triggers for TicketSeller table
ALTER TABLE TicketSeller ENABLE ALL TRIGGERS;

-- Enabling triggers for Ticket table
ALTER TABLE Ticket ENABLE ALL TRIGGERS;

-- Enabling triggers for Baggage table
ALTER TABLE Baggage ENABLE ALL TRIGGERS;

-- Enabling triggers for Booking table
ALTER TABLE Booking ENABLE ALL TRIGGERS;

-- Enabling triggers for Payment_Report table
ALTER TABLE Payment_Report ENABLE ALL TRIGGERS;

```

sel1.sql

SQL Output Statistics

```

SELECT * FROM Passenger;
SELECT * FROM TicketSeller;
SELECT * FROM Ticket;
SELECT * FROM Baggage;
SELECT * FROM Booking;
SELECT * FROM Payment_Report;

```

Select passenger | Select ticketseller | Select ticket | Select baggage | Select booking | Select payment\_report

	PASSENGER_ID	PASSENGER_NAME	PASSENGER_PHONE	PASSENGER_EMAIL
1	1573871528	Reid	557345006	rob.reid@hewlettpackard.ch
2	4139544263	Keith	558099995	a.keith@wci.com
3	379982361	Orlando	559214529	rorlando@lms.uk
4	334694516	Loggia	584251872	marty.loggia@epamsystems.ca
5	342850352	Krumholtz	589935367	cole.k@cocacola.com

000 1 of 490 c##eytan12@XE [19:24:40] 490 rows selected in 0.470 seconds

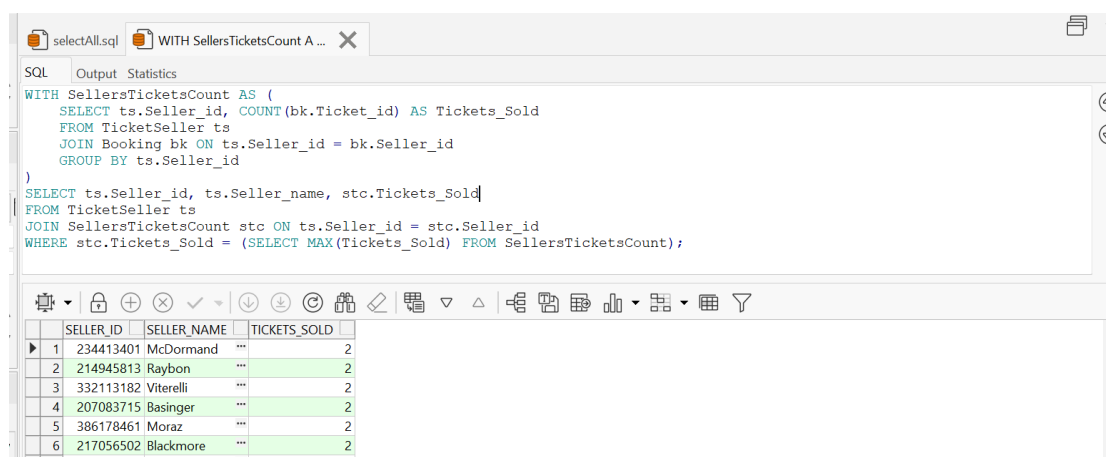
## חלק ב' – שאלות

שאלות בלי פרמטרים:

שאלות SELECT:

שאלת 1:

הצגת המוכרים שמכרו הכי הרבה כרטיסים סך הכול:  
המידע על המוכרים שמכרו הכי הרבה כרטיסים חשוב לחברה כי הוא מאפשר לזהות ולהוקיר את המוכרים המובילים, לנתח את ביצועיהם וליישם אסטרטגיות דומות להגדלת המכירות הכוללת.



```
WITH SellersTicketsCount AS (
    SELECT ts.Seller_id, COUNT(bk.Ticket_id) AS Tickets_Sold
    FROM TicketSeller ts
    JOIN Booking bk ON ts.Seller_id = bk.Seller_id
    GROUP BY ts.Seller_id
)
SELECT ts.Seller_id, ts.Seller_name, stc.Tickets_Sold
FROM TicketSeller ts
JOIN SellersTicketsCount stc ON ts.Seller_id = stc.Seller_id
WHERE stc.Tickets_Sold = (SELECT MAX(Tickets_Sold) FROM SellersTicketsCount);
```

	SELLER_ID	SELLER_NAME	TICKETS_SOLD
1	234413401	McDormand	2
2	214945813	Raybon	2
3	332113182	Viterelli	2
4	207083715	Basinger	2
5	386178461	Moraz	2
6	217056502	Blackmore	2

שאלת 2:

הצגת פרטי הנוסעים שהזמינו כרטיסים ביום שבו הזמינו הכי הרבה כרטיסים:  
הצגת פרטי הנוסעים שהזמינו כרטיסים ביום שבו הזמינו הכי הרבה כרטיסים חשובה לחברה כי היא מאפשרת לזהות מגמות ביקוש, לייעל את תכנון השירותים ולשפר את חוויית הלקוח באמצעות התאמה טובה יותר להעדפותיהם וצרכיהם.



selectAll.sql SELECT p.Passenger\_name, b ...

SQL Output Statistics

```

SELECT p.Passenger_name, bk.Booking_date, bk.ticket_id
FROM Passenger p
JOIN Booking bk ON p.Passenger_id = bk.Passenger_id
WHERE bk.Booking_date = (
    SELECT bk.Booking_date
    FROM Booking bk
    GROUP BY bk.Booking_date
    ORDER BY COUNT(bk.Ticket_id) DESC
    FETCH FIRST 1 ROWS ONLY
)
ORDER BY bk.Booking_date;

```

	PASSENGER_NAME	BOOKING_DATE	TICKET_ID
1	Reid	08/06/2023	200137
2	de Lancie	08/06/2023	359821
3	Hebard	08/06/2023	213289
4	Blest	08/06/2023	217895
5	Tomasutti	08/06/2023	223134
6	Agglio	08/06/2023	231694
7	Keatley	08/06/2023	237657

4:55 #airind@VF 100:56:37 7 rows selected in 0.160 seconds (rows)

שאלתה 3: מציאת האנשים שלא שילמו על כרטיס:  
 מציאת האנשים שלא שילמו על כרטיס חשובה לחברה כדי לטפל בהכנסות אבודות,  
 למנוע הונאה ולוודא שכל הנוסעים משלמים עבור השירותים שהם מקבלים, מה  
 שתורם לניהול פיננסי יעיל יותר.

selectAll.sql SELECT passenger.Passenger ...

SQL Output Statistics

```

SELECT passenger.Passenger_name, passenger.passenger_email, passenger.passenger_phone
FROM passenger
JOIN booking ON passenger.Passenger_id = booking.Passenger_id
WHERE booking.Booking_id NOT IN (
    SELECT Booking_id
    FROM payment_report
)
ORDER BY passenger.Passenger_name;

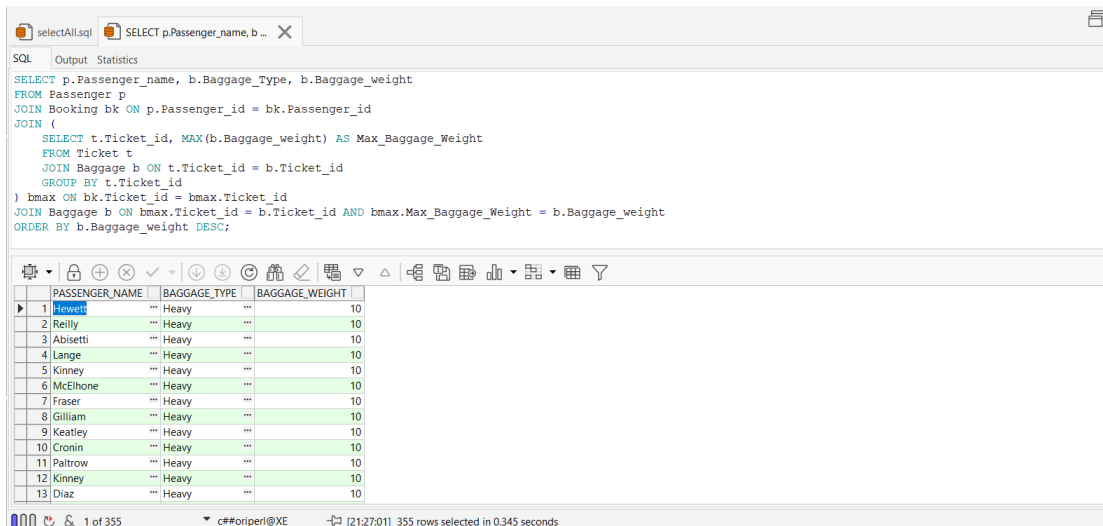
```

	PASSENGER_NAME	PASSENGER_EMAIL	PASSENGER_PHONE
1	Balmer	tbalmer12@trellian.com	311-168-8736
2	Canario	ecanario1g@squarespace.com	424-626-5683
3	Cube	gavinc@perfectorder.br	558508908
4	Dorff	w.dorff@msdw.com	558102429
5	Estick	mestickm@shareasale.com	274-606-6944
6	Garr	aprilg@asa.it	552537916
7	Guzman	jody@otbd.com	557695174

4:34 #airind@VF 100:56:16 7 rows selected in 0.004 seconds (rows)

#### שאלתה 4:

מציאת שמות הנוסעים עם הכבודה הכי כבדה והסוג של הכבודה:  
מציאת שמות הנוסעים עם הכבודה הכי כבדה והסוג של הכבודה חשובה לחברה כדי  
לנהל את משקל הטיסות בצורה יעילה, להבטיח בטיחות ויציבות, ולספק שירותים  
מותאמים אישית לנוסעים עם דרישות מיוחדות.



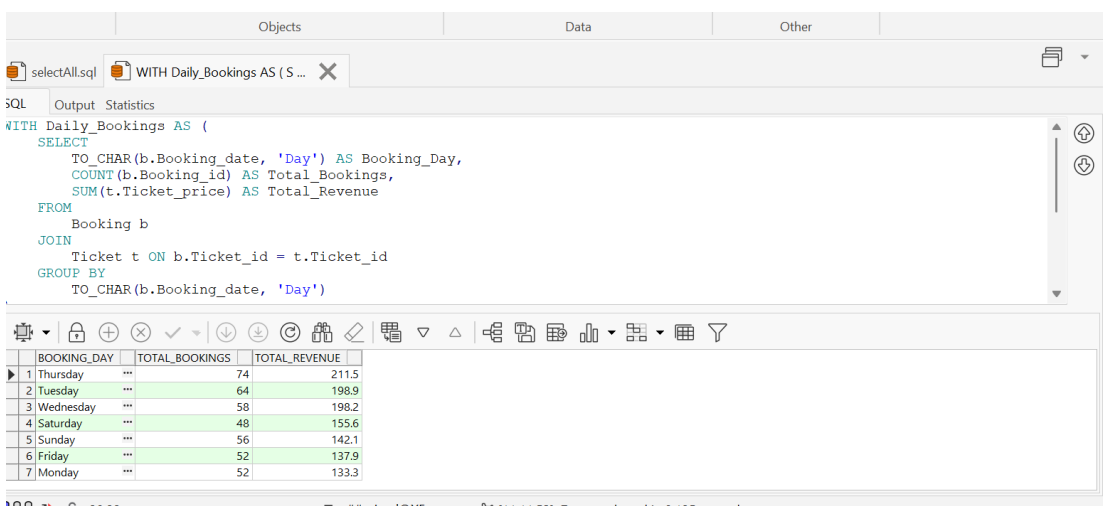
The screenshot shows a SQL query in the 'SQL' tab of SQL Developer. The query is a complex join between Passenger, Booking, Ticket, and Baggage tables to find the heaviest baggage for each passenger. The results are shown in the 'Output' tab as a table with 13 rows.

```
SELECT p.Passenger_name, b.Baggage_Type, b.Baggage_weight
FROM Passenger p
JOIN Booking bk ON p.Passenger_id = bk.Passenger_id
JOIN (
  SELECT t.Ticket_id, MAX(b.Baggage_weight) AS Max_Baggage_Weight
  FROM Ticket t
  JOIN Baggage b ON t.Ticket_id = b.Ticket_id
  GROUP BY t.Ticket_id
) bmax ON bk.Ticket_id = bmax.Ticket_id
JOIN Baggage b ON bmax.Ticket_id = b.Ticket_id AND bmax.Max_Baggage_Weight = b.Baggage_weight
ORDER BY b.Baggage_weight DESC;
```

	PASSENGER_NAME	BAGGAGE_TYPE	BAGGAGE_WEIGHT
1	Hewett	Heavy	10
2	Reilly	Heavy	10
3	Abisetti	Heavy	10
4	Lange	Heavy	10
5	Kinney	Heavy	10
6	McElhone	Heavy	10
7	Fraser	Heavy	10
8	Gilliam	Heavy	10
9	Keatley	Heavy	10
10	Cronin	Heavy	10
11	Paltrow	Heavy	10
12	Kinney	Heavy	10
13	Diaz	Heavy	10

\*בנוס\*: איזה ימים היו הכי רווחיים:

זיהוי הימים שהיו הכי רווחיים חשוב לחברה כי הוא מאפשר לנתח מגמות הכנסות,  
למקד מבצעים ושיווק בתאריכים פוטנציאליים ולהתאים את הקצאת המשאבים  
והתפעול בהתאם כדי למקסם רווחים עתידיים.



The screenshot shows a SQL query in the 'SQL' tab of SQL Developer. The query uses a CTE named 'Daily\_Bookings' to calculate total bookings and revenue by day of the week. The results are shown in the 'Output' tab as a table with 7 rows.

```
WITH Daily_Bookings AS (
  SELECT
    TO_CHAR(b.Booking_date, 'Day') AS Booking_Day,
    COUNT(b.Booking_id) AS Total_Bookings,
    SUM(t.Ticket_price) AS Total_Revenue
  FROM
    Booking b
  JOIN
    Ticket t ON b.Ticket_id = t.Ticket_id
  GROUP BY
    TO_CHAR(b.Booking_date, 'Day')
)
```

	BOOKING_DAY	TOTAL_BOOKINGS	TOTAL_REVENUE
1	Thursday	74	211.5
2	Tuesday	64	198.9
3	Wednesday	58	198.2
4	Saturday	48	155.6
5	Sunday	56	142.1
6	Friday	52	137.9
7	Monday	52	133.3

## delete

מחיקת מוכרים שלא מכרו כרטיסים בשנה האחרונה:  
מחיקת מוכרים שלא מכרו כרטיסים בשנה האחרונה חשובה לחברה כדי לייעל את  
ניהול כוח האדם, לצמצם עלויות הקשורות לניהול עובדים לא פעילים ולהתמקד  
במוכרים הפעילים שמייצרים הכנסות.

The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results grid. The query is a DELETE statement targeting the TicketSeller table based on a condition derived from the Booking table. The results grid displays 19 rows of data for the TicketSeller table.

```
SQL
DELETE FROM TicketSeller
WHERE Seller_id NOT IN (
    SELECT DISTINCT bk.Seller_id
    FROM Booking bk
    WHERE bk.Booking_date >= ADD_MONTHS(SYSDATE, -12)
);
```

SELLER_ID	SELLER_NAME	SELLER_CONTACT
8	325390520 Badalucco	511270153
9	246648879 Warden	549952920
10	324559120 Broderick	505142978
11	297909520 Alexander	550394009
12	280607651 Aiken	561697874
13	228359755 Benoit	586002921
14	355379591 Yankovic	522759611
15	220299024 McBride	506606288
16	256738120 O'Connor	579200664
17	298423242 Wopat	587238122
18	262882544 Cervine	556035975
19	354252310 Vaughn	508978282

שאלתה 2: מחיקת כל הנוסעים שלא ביצעו הזמנה מאז תאריך מסוים, ושאין להם כבודה  
מחיקת כל הנוסעים שלא ביצעו הזמנה מאז תאריך מסוים ושאין להם כבודה חשובה לחברה  
כדי לשמור על מסד נתונים נקי ומדויק, להפחית עלויות אחסון ולמקד את המאמצים  
השיווקיים והמשאבים בנוסעים פעילים בעלי פוטנציאל להכנסות.

The screenshot shows a SQL Server Enterprise Manager window with a query editor and a results grid. The query is a DELETE statement targeting the Passenger table based on a condition derived from the Booking table and a self-join on the Passenger table. The results grid displays 13 rows of data for the Passenger table.

```
SQL
DELETE FROM Passenger
WHERE Passenger_id NOT IN (
    SELECT DISTINCT bk.Passenger_id
    FROM Booking bk
    WHERE bk.Booking_date >= TO_DATE('2023-07-12', 'YYYY-MM-DD')
)
AND Passenger_id NOT IN (
    SELECT DISTINCT bk.Passenger_id
    FROM Booking bk
    JOIN Baggage b ON bk.Ticket_id = b.Ticket_id
);
```

PASSENGER_ID	PASSENGER_NAME	PASSENGER_PHONE	PASSENGER_EMAIL
1	4139544263 Keith	558099995	a.keith@wci.com
2	334694516 Loggia	584251872	marty.loggia@epamsystems.ca
3	296014467 Davis	567218634	merilee.davis@mathis.com
4	270448006 Moraz	515607433	wendy.moraz@mls.com
5	317148159 Edmunds	517677942	b.edmunds@pragmatechsoftware.uk
6	223177804 Andrews	575879623	guy.andrews@ait.com
7	300000003 Blayney	195-917-7571	eblayney3@mediafire.com
8	300000004 Cruxton	874-537-4250	lcruxton4@wix.com
9	300000006 Trimmell	927-618-7899	dtrimmell6@devhub.com
10	300000007 Broschke	342-502-1817	pbroschke7@pen.io
11	300000010 Hebard	205-237-0029	mhebarda@discuz.net
12	300000012 Venditto	415-120-3226	lvenditto@ucuz.com
13	300000013 Quittonden	444-919-5544	fquittondend@typepad.com

## Update

עדכון מחיר הכרטיסים לסוג כרטיס מסויים (Ticket\_type) שהוזמנו אחרי תאריך מסויים: (עודכנו 75 שורות)  
עדכון מחיר הכרטיסים לסוג כרטיס מסויים שהוזמנו אחרי תאריך מסויים חשוב לחברה כדי לשקף שינויים בשוק, להגדיל רווחים, ולהתאים את התמחור לתנאי ביקוש והיצע משתנים.



```
UPDATE Ticket
SET Ticket_price = Ticket_price * 1.10
WHERE Ticket_id IN (
  SELECT DISTINCT t.Ticket_id
  FROM Ticket t
  JOIN Booking b ON t.Ticket_id = b.Ticket_id
  WHERE b.Booking_date > TO_DATE('2023-06-12', 'YYYY-MM-DD')
  AND t.Ticket_type = 'regular'
);
```

75 rows updated in 0.013 seconds

עדכון מספר הטלפון של המוכרים שמכרו הכי הרבה כרטיסים: (עודכנה שורה)  
עדכון מספר הטלפון של המוכרים שמכרו הכי הרבה כרטיסים חשוב לחברה כדי להבטיח תקשורת יעילה ומתמשכת עם המוכרים המובילים, לתמוך בהם טוב יותר ולהבטיח שהם זמינים להמשיך מאמצי המכירה והקידום.



```
UPDATE TicketSeller
SET Seller_contact = '999999999'
WHERE Seller_id IN (
  SELECT Seller_id
  FROM (
    SELECT bk.Seller_id, COUNT(*) AS num_tickets
    FROM Booking bk
    GROUP BY bk.Seller_id
    ORDER BY num_tickets DESC
  )
  WHERE ROWNUM = 1
);
```

1 row updated in 0.004 seconds

## שאלות פרמטרים:

(1)

מציאת המוכרים שמכרו הכי הרבה כרטיסים לאחר תאריך מסוים יחד עם סך הכרטיסים שמכרו וסכומם המשקולל (פרמטר תאריך):

מציאת המוכרים שמכרו הכי הרבה כרטיסים לאחר תאריך מסוים יחד עם סך הכרטיסים שמכרו וסכומם המשקולל חשובה לחברה כדי לזהות את המוכרים המצטיינים בתקופה נתונה, להעריך את ביצועיהם הכספיים ולתכנן תגמולים או תמריצים שיניעו מכירות נוספות.

The screenshot shows a SQL query in the 'SQL' tab of SQL Developer. The query is a CTE named 'Seller\_Info' that selects seller name, total sales price, and tickets sold from the TicketSeller, Booking, and Ticket tables. The results are displayed in a table below the query editor.

	SELLER_NAME	TICKETS_SOLD	TOTAL_SALES_PRICE
1	Nicks	2	9
2	Raybon	2	6.5
3	Cazale	2	6

At the bottom of the window, the status bar indicates: c##oriperl@XE, [10:12:09], 3 rows selected in 0.104 seconds.

(2) משקל ומחיר ממוצע של רשימה של סוגי מזוודות (פרמטר רשימה):

מידע על משקל ומחיר הממוצע של סוגי מזוודות חשוב לחברה לתכנון וניהול מלאי יעיל.

The screenshot shows a SQL query in the 'SQL' tab of SQL Developer. The query selects ticket type, average baggage weight, and average ticket price from the Ticket and Baggage tables, filtered by ticket type 'regular' or 'special'. The results are displayed in a table below the query editor.

	TICKET_TYPE	AVGBAGGAGEWEIGHT	AVGTICKETPRICE
1	regular	5.05882352941176	2.76470588235294
2	special	3.96	3.6496

At the bottom of the window, the status bar indicates: c##oriperl@XE, [09:27:57], 2 rows selected in 0.080 seconds.

- 3) שמות ות"ז המוכרים שהכניסו הכי הרבה כסף ממכירת כרטיסים מסוג 'מיוחד' וכמה שהרוויחו (פרמטר שם)
- שמות ות"ז של המוכרים שהכניסו הכי הרבה כסף ממכירת כרטיסים מסוג 'מיוחד' והסכום שהם הרוויחו מסייעים לחברה לזהות את המוכרים המצטיינים ולפתח אסטרטגיות לגידול המכירות בקטגוריה זו.

The screenshot shows a SQL query in the 'SQL' tab of SQL Developer. The query is a JOIN between 'TicketSeller ts', 'Booking bk', and 'Ticket t' to calculate total sales for each seller. The 'Output' tab shows the results in a table with columns: SELLER\_ID, SELLER\_NAME, and TOTAL\_SALES. The results are sorted by total sales in descending order.

SELLER_ID	SELLER_NAME	TOTAL_SALES
1	310000005 Klaus	8
2	310000036 Withey	8
3	310000004 Pechard	8
4	310000031 Thorold	8
5	310000001 Aronin	8
6	310000010 Ayrton	8
7	255356533 Hingle	8
8	210000004 Mousak	8

1 of 138 rows selected in 0.202 seconds

- 4) נתוני מכירות על חודש יוני 2023 :
- נתוני המכירות על חודש יוני 2023 חשובים לחברה כדי להעריך את ביצועי השוק בתקופה זו, לנתח מגמות ביקוש ולתכנן אסטרטגיות עתידיות להגדלת המכירות ולשיפור תגובת השוק.

The screenshot shows a SQL query in the 'SQL' tab of SQL Developer. The query uses a CTE named 'Monthly\_Sales AS' to calculate the count of bookings and total revenue for each month-year combination. The query filters for the month of June 2023. The 'Output' tab shows the results in a table with columns: MONTH\_YEAR, TOTAL\_BOOKINGS, and TOTAL\_REVENUE.

MONTH_YEAR	TOTAL_BOOKINGS	TOTAL_REVENUE
1 06-2023	379	1091.9

1 row selected in 0.150 seconds

## אילוצים

### Passenger: not null

נדרוש לא להשאיר את שם הנוסע ללא שם

```
SQL Output Statistics
-- NOT NULL constraint and DEFAULT constraint for a field in the Passenger table
ALTER TABLE Passenger
MODIFY Passenger_name VARCHAR2(255) NOT NULL,
```

ננסה להכניס נוסע ללא שם:

The screenshot shows a SQL Developer window with a query that attempts to insert a NULL value for the Passenger\_name field. An error dialog box is displayed, stating: "ORA-01400: cannot insert NULL into ('C##ORIPERL', 'PASSENGER', 'PASSENGER\_NAME')". The error message is in English, and the dialog box has "OK", "Cancel", and "Help" buttons.

### TicketSeller: check

נדרוש שאורך המס' טלפון יהיה 20 תווים ומטה

```
-- CHECK constraint for a field in the TicketSeller table
ALTER TABLE TicketSeller
ADD CONSTRAINT check_seller_contact
CHECK (LENGTH(Seller_contact) <= 20);
```

ננסה להכניס עם מספר ארוך מ20 תווים:

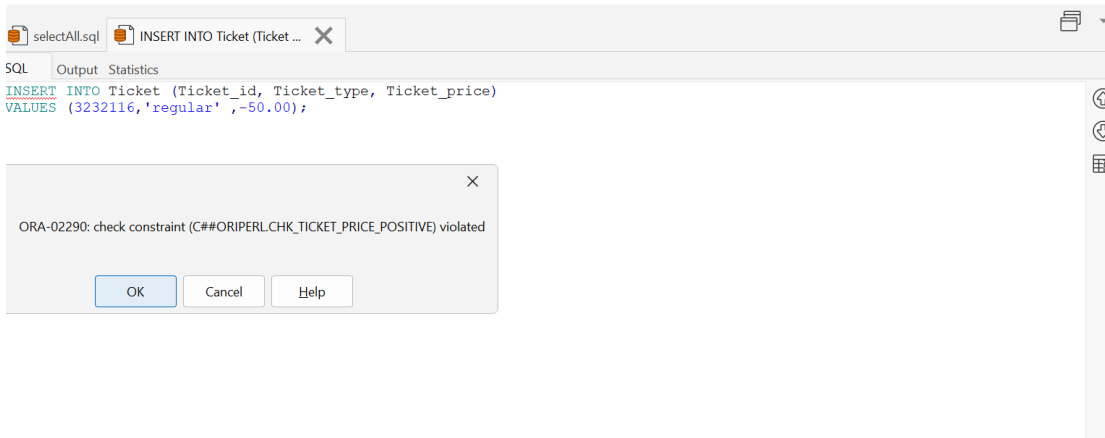
The screenshot shows a SQL Developer window with a query that attempts to insert a long contact number for the TicketSeller table. An error dialog box is displayed, stating: "ORA-02290: check constraint (C##ORIPERL.CHECK\_SELLER\_CONTACT) violated". The error message is in English, and the dialog box has "OK", "Cancel", and "Help" buttons.

### Ticket: check

נדרוש מחיר חיובי לכרטיס

```
ALTER TABLE Ticket
ADD CONSTRAINT CHK_Ticket_Price_Positive CHECK (Ticket_price > 0);
```

ננסה להכניס עם סכום שלילי

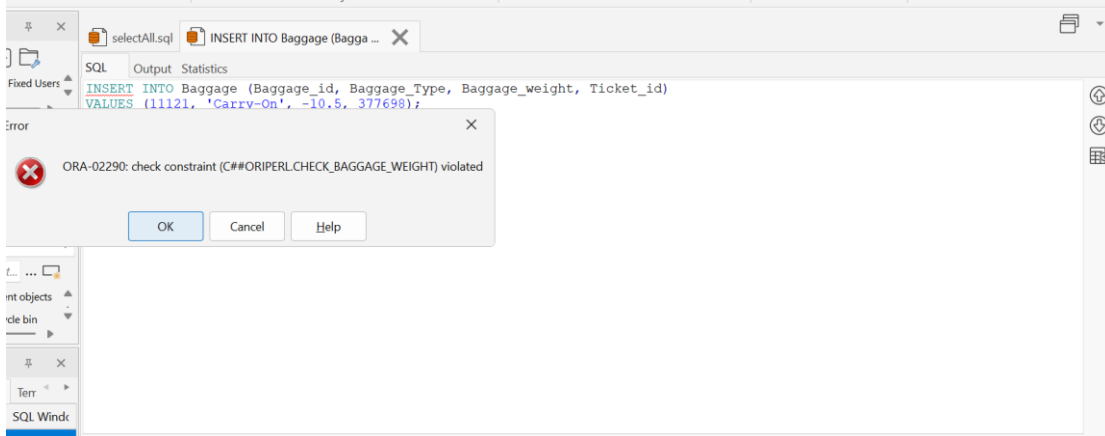


## Baggage: check

נדרוש שמשקל המזוודה אינו שלילי

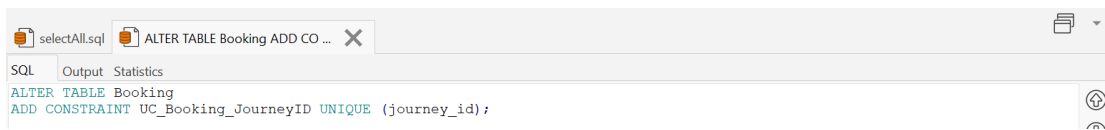
```
-- CHECK constraint for a field in the Baggage table
ALTER TABLE Baggage
ADD CONSTRAINT check_baggage_weight
CHECK (Baggage_weight >= 0);
```

ננסה להכניס מזוודה עם משקל שלילי:

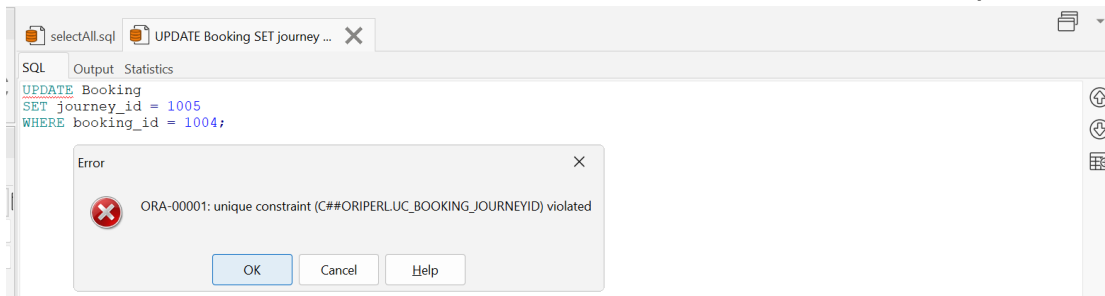


## Booking: unique

נדרוש שלכל הזמנה יהיה מספר מסע מיוחד



ננסה לעדכן למספר מסע דומה:





## Payment report: check

נדרוש שמספר התשלום/ת"ז התשלום יהיה חיובי

```
SQL Output Statistics
-- CHECK constraint for a field in the table
ALTER TABLE Payment_Report
ADD CONSTRAINT CHK_P_Positive CHECK (p_id > 0);
```


ננסה לעדכן לת"ז תשלום שלילי:

selectAll.sql UPDATE Payment\_Report SET ...

SQL Output Statistics

```
UPDATE Payment_Report
SET payment_id = -11
WHERE payment_id = 1001;
```

Error

 ORA-02290: check constraint (C##ORIPERL.CHK\_P\_POSITIVE) violated

OK

Cancel

Help

## שלב 3 – פונקציות ופרוצדורות:

פונ' 1: חישוב רווח כולל בטווח תאריכים מסוים:

**מנהל חברת הסעות, יוסי, עמד מול ערימת מסמכים מבולגנת במשרדו הקטן. המתח היה ניכר על פניו; הוא ידע שפטור ממס יכול לחסוך לחברתו סכומי כסף משמעותיים. הוא החליט להיעזר ברואה חשבון מנוסה כדי להבין את סך הרווחים של החברה בין תאריכים מסוימים. לאחר מספר פגישות ודיונים, רואה החשבון הצליח לסדר את כל הנתונים ולחשב את הרווחים במדויק. יוסי הרגיש הקלה עצומה, וכשקיבל את האישור לפטור ממס, ידע שהמאמץ היה משתלם. החברה שלו תוכל עכשיו להשקיע את החסכונות בפרויקטים חדשים ולהמשיך לצמוח.**

```
tion  Cursor  Select
CREATE OR REPLACE FUNCTION calculate_total_revenue(start_date IN DATE, end_date IN DATE)
RETURN NUMBER
IS
    total_revenue NUMBER := 0;
    CURSOR cur_revenue IS
        SELECT SUM(t.Ticket_price) AS TotalRevenue
        FROM Booking b
        JOIN Ticket t ON b.Ticket_id = t.Ticket_id
        WHERE b.Booking_date BETWEEN start_date AND end_date;
BEGIN
    OPEN cur_revenue;
    FETCH cur_revenue INTO total_revenue;
    CLOSE cur_revenue;

    RETURN total_revenue;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        RETURN 0;
    WHEN OTHERS THEN
        RAISE;
END;
```

פונ' 2: מספר האנשים ששם המשפחה שלהם מתחיל באות מסוימת

**מנהל תחנת תחבורה ציבורית, אבי, קיבל משימה מעניינת מהעירייה: לבדוק כמה נוסעים ששם משפחתם מתחיל באות מסוימת משתמשים בתחבורה הציבורית בעיר. המידע הזה היה נחוץ לצורך קמפיין שיווקי ממוקד. אבי ידע שהאתגר גדול, אך היה נחוש לעמוד בו. הוא פנה למערכת הניהול הדיגיטלית של התחנה וביקש מהצוות הטכני להפיק דוח על שמות הנוסעים. לאחר כמה ימים של עבודה מאומצת ואיסוף נתונים, הצליחו להפיק את הדוח המבוקש. אבי הציג את הממצאים לעירייה, שהייתה מרוצה מאוד והשתמשה במידע לתכנון הקמפיין הממוקד. התחנה זכתה לשבחים רבים על העבודה הקפדנית, ואבי ידע שהוא וצוותו תרמו לשיפור השירות לתושבים.**

```
le section  Loop  Code section  Statement  < >
1 CREATE OR REPLACE FUNCTION Count_Passengers_With_Condition(initial IN CHAR)
2 RETURN NUMBER IS
3     passengerCount NUMBER := 0;
4     passengerRec Passenger%ROWTYPE;
5     CURSOR passengerCursor IS
6         SELECT * FROM Passenger WHERE SUBSTR(Passenger_name, 1, 1) = initial;
7 BEGIN
8     OPEN passengerCursor;
9     LOOP
10         FETCH passengerCursor INTO passengerRec;
11         EXIT WHEN passengerCursor%NOTFOUND;
12         passengerCount := passengerCount + 1;
13     END LOOP;
14     CLOSE passengerCursor;
15
16     RETURN passengerCount;
17 EXCEPTION
18     WHEN OTHERS THEN
19         RETURN -1;
20 END;
```

## פרוצדורה 1: חישוב המשקל הכולל

**מנהל תחנת תחבורה ציבורית, רמי, קיבל משימה מורכבת מהעירייה: לחשב את המשקל הכולל של הכבודות בכל הנסיעות במהלך החודש האחרון. המידע היה חשוב לצורך תכנון ושיפור מערך התחבורה הציבורית בעיר.**

**רמי פנה לצוות הטכני וביקש להוציא נתונים מכל הנסיעות של החודש האחרון. הם עבדו ימים כלילות, בודקים כל כבודה, רושמים את משקלה ומחשבים את הסך הכולל. לאחר איסוף הנתונים ועיבודם, הצליחו להגיע למספר המדויק.**

**כשרמי הציג את הממצאים לעירייה, הם הופתעו מהתוצאה שהראתה את העומס על התחבורה הציבורית. המידע עזר להם לקבל החלטות חשובות לשיפור השירות, כמו תוספת כלי רכב במועדים עמוסים. רמי ידע שהעבודה הקשה שלו ושל הצוות תורמת לשיפור חוויית הנסיעה עבור כל התושבים בעיר.**

```
CREATE OR REPLACE PROCEDURE calculate_total_baggage_weight(ticket_type IN VARCHAR2, total_weight OUT NUMBER) IS
    CURSOR cur_baggage IS
        SELECT b.Baggage_weight
        FROM Baggage b
        JOIN Ticket t ON b.Ticket_id = t.Ticket_id
        WHERE t.Ticket_type = ticket_type;
    rec_baggage cur_baggage%ROWTYPE;
BEGIN
    total_weight := 0;
    OPEN cur_baggage;
    LOOP
        FETCH cur_baggage INTO rec_baggage;
        EXIT WHEN cur_baggage%NOTFOUND;
        total_weight := total_weight + rec_baggage.Baggage_weight;
    END LOOP;
    CLOSE cur_baggage;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        total_weight := 0;
```

## פרוצדורה 2: חישוב עלות כל סוג כרטיס ועדכון המחיר לכל סוג

**מנהל תחנת תחבורה ציבורית, גדי, קיבל הנחיה מהעירייה לבצע חישוב מחודש של עלות כל סוג כרטיס ועדכון המחירים בהתאם. המשימה הייתה קריטית לשמירה על תקציב התחנה ולשיפור השירות לנוסעים.**

**גדי כינס את הצוות הכלכלי והטכני והחל במבצע איסוף נתונים. הם בחנו את כל סוגי הכרטיסים: יומיים, שבועיים, חודשיים וכרטיסי סטודנטים. הם לקחו בחשבון עלויות תפעול, תחזוקה, ושינויים בביקוש.**

**לאחר חישוב מעמיק ועדכני, הצוות גילה שיש צורך בהתאמת מחירים כדי לכסות את העלויות ולהשאיר מרווח סביר לרווחים. הם עדכנו את המחירים במערכת והודיעו לנוסעים על השינויים הקרובים.**

**כשהשינויים נכנסו לתוקף, גדי עקב מקרוב אחרי התגובות מהנוסעים והשפעת השינויים על השימוש בתחבורה הציבורית. המהלך הוביל לאיזון טוב יותר בין ההכנסות להוצאות, ושיפר את היכולת של התחנה להעניק שירות איכותי לנוסעים.**

```
CREATE OR REPLACE PROCEDURE Calculate_Total_Sales IS
    CURSOR cur_ticket IS
        SELECT Ticket_type, SUM(Ticket_price) AS Total_Sales
        FROM Ticket
        GROUP BY Ticket_type;
    rec_ticket cur_ticket%ROWTYPE;
    new_price NUMBER;
BEGIN
    OPEN cur_ticket;
    LOOP
        FETCH cur_ticket INTO rec_ticket;
        EXIT WHEN cur_ticket%NOTFOUND;
        -- Set new price based on ticket type
        CASE rec_ticket.Ticket_type
            WHEN 'regular' THEN
                new_price := 10;
            WHEN 'special' THEN
                new_price := 15;
            WHEN 'handicapped' THEN
                new_price := 5;
            ELSE
                new_price := rec_ticket.Total_Sales / 100; -- Default price if type doesn't match
        END CASE;
        -- Update ticket prices
        UPDATE Ticket
        SET Ticket_price = new_price
        WHERE Ticket_type = rec_ticket.Ticket_type;
        DBMS_OUTPUT.PUT_LINE('Ticket Type: ' || rec_ticket.Ticket_type ||
            ', Total Sales: ' || rec_ticket.Total_Sales ||
            ', New Price: ' || new_price);
    END LOOP;
    CLOSE cur_ticket;
    DBMS_OUTPUT.PUT_LINE('Total sales calculated and ticket prices updated successfully.');
```

## Main1:

```
DECLARE
  start_date DATE := TO_DATE('2023-06-06', 'YYYY-MM-DD');
  end_date DATE := TO_DATE('2023-12-31', 'YYYY-MM-DD');
  total_revenue NUMBER;
  total_weight NUMBER;
BEGIN
  -- קריאה לפונקציה calculate_total_revenue
  total_revenue := calculate_total_revenue(start_date, end_date);
  DBMS_OUTPUT.PUT_LINE('Total Revenue: ' || total_revenue);

  -- קריאה לפונקציה calculate_total_baggage_weight
  calculate_total_baggage_weight('regular', total_weight);
  DBMS_OUTPUT.PUT_LINE('Total Baggage Weight: ' || total_weight);
END;
/
```

## Main2:

```
DECLARE
  initial CHAR := 'Z' -- אות התחלתית לשם הנוסע
  passengerCount NUMBER
BEGIN
  -- קריאה לפונקציה Count_Passengers_With_Condition
  passengerCount := Count_Passengers_With_Condition(initial)

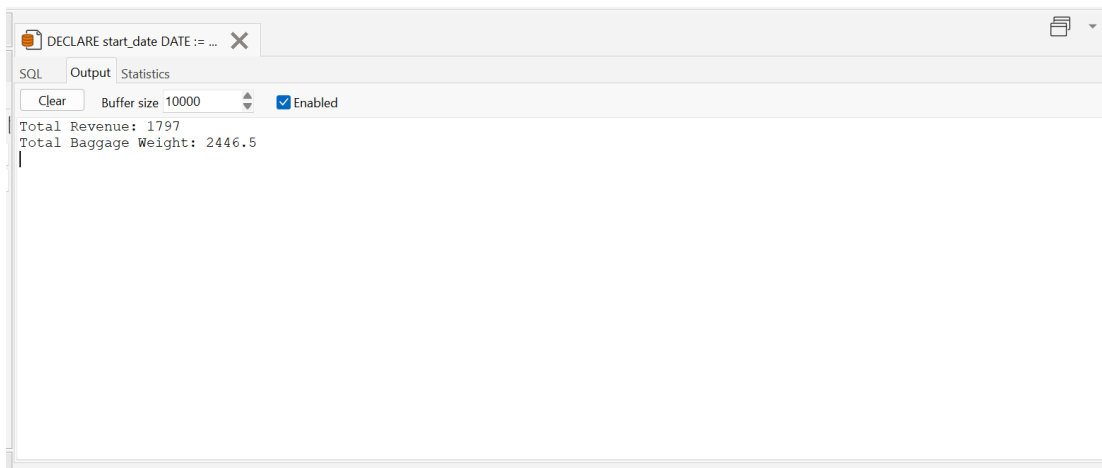
  -- הדפסת התוצאה
  DBMS_OUTPUT.PUT_LINE('Number of passengers with names starting with ' || initial || ': ' || passengerCount)

  -- Call the Calculate_Total_Sales procedure
  Calculate_Total_Sales

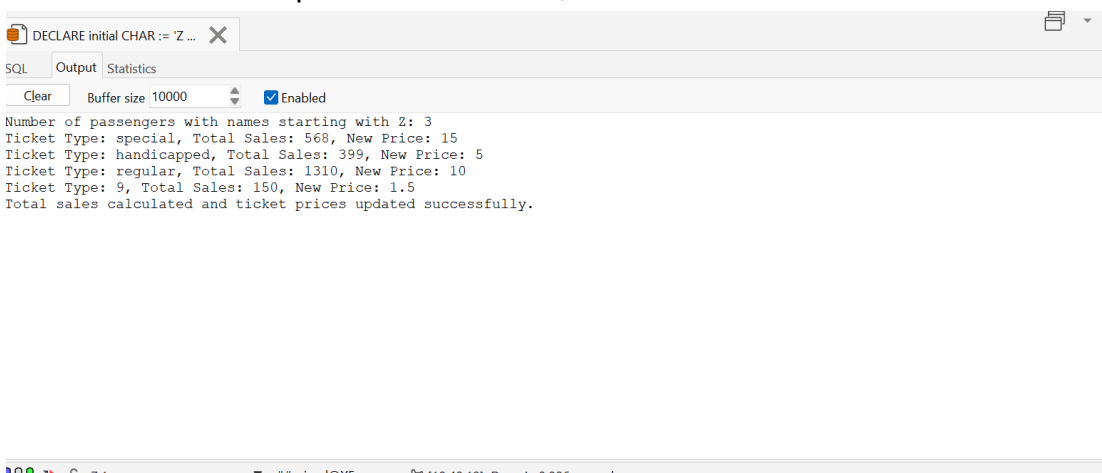
;END
/
```

הוכחת נכונות ריצה: עבור פונ' 1 ופרוצדורה 1, הרצנו את הMAIN וקיבלנו:

- בלמ"ס -



הוכחת נכונות ריצה: עבור פונ' 2 ופרוצדורה 2, הרצנו את ה-MAIN וקיבלנו:



אכן בוצע העדכון:

```
SELECT * FROM Booking;
```

```
SELECT * FROM Payment_Report;
```

Select passenger






Select ticketseller






Select ticket






Select baggage

Select booking

Select payment

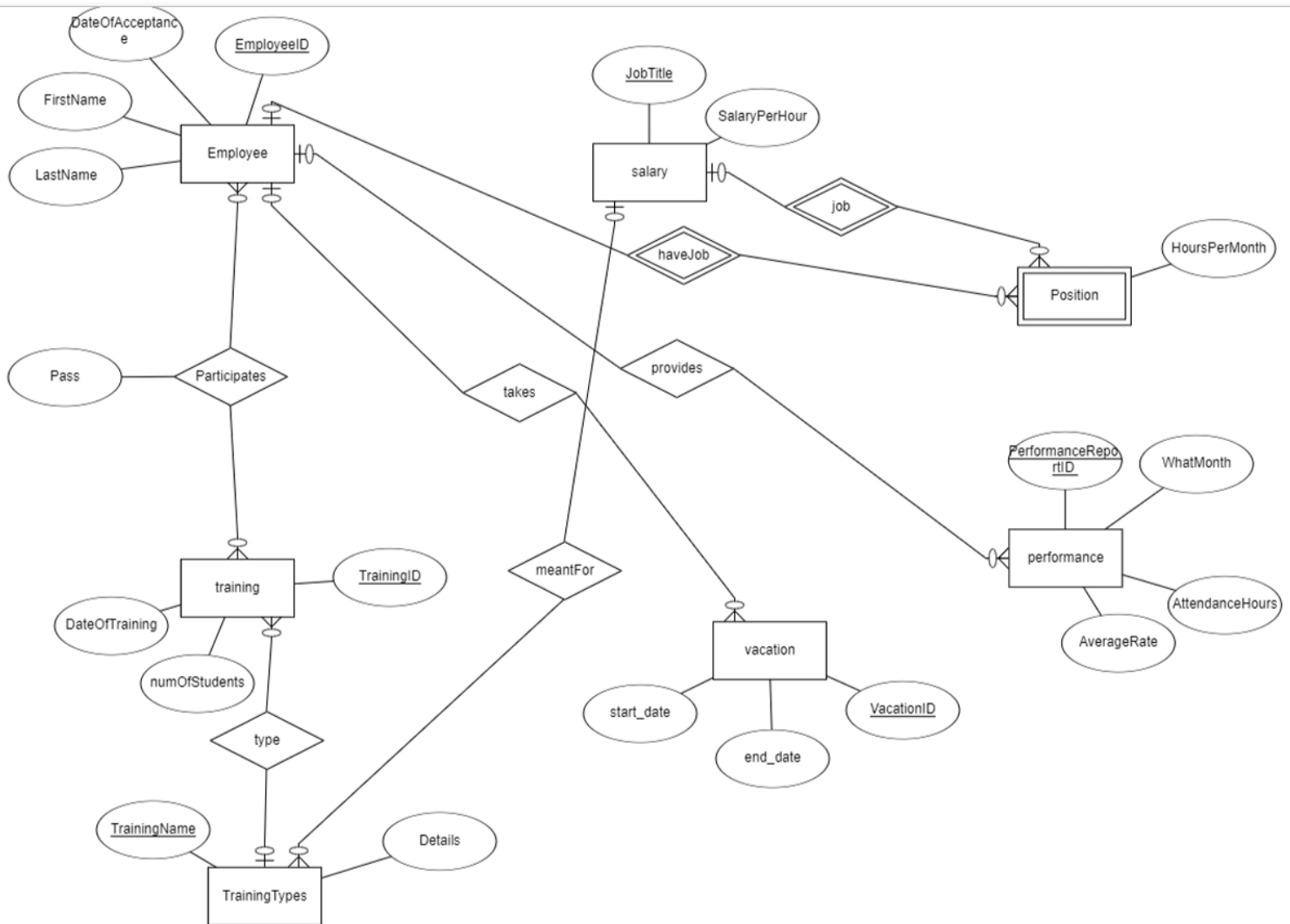
	TICKET_ID	TICKET_TYPE	TICKET_PRICE
46	203421	regular	10
47	290598	handicapped	5
48	200137	special	15
49	236715	handicapped	5
50	380933	special	15

	TICKET_ID	TICKET_TYPE	TICKET_PRICE
403	359844	handicapped	5
404	359845	special	15
405	359846	regular	10
406	359847	regular	10
407	3232116	9	1.5

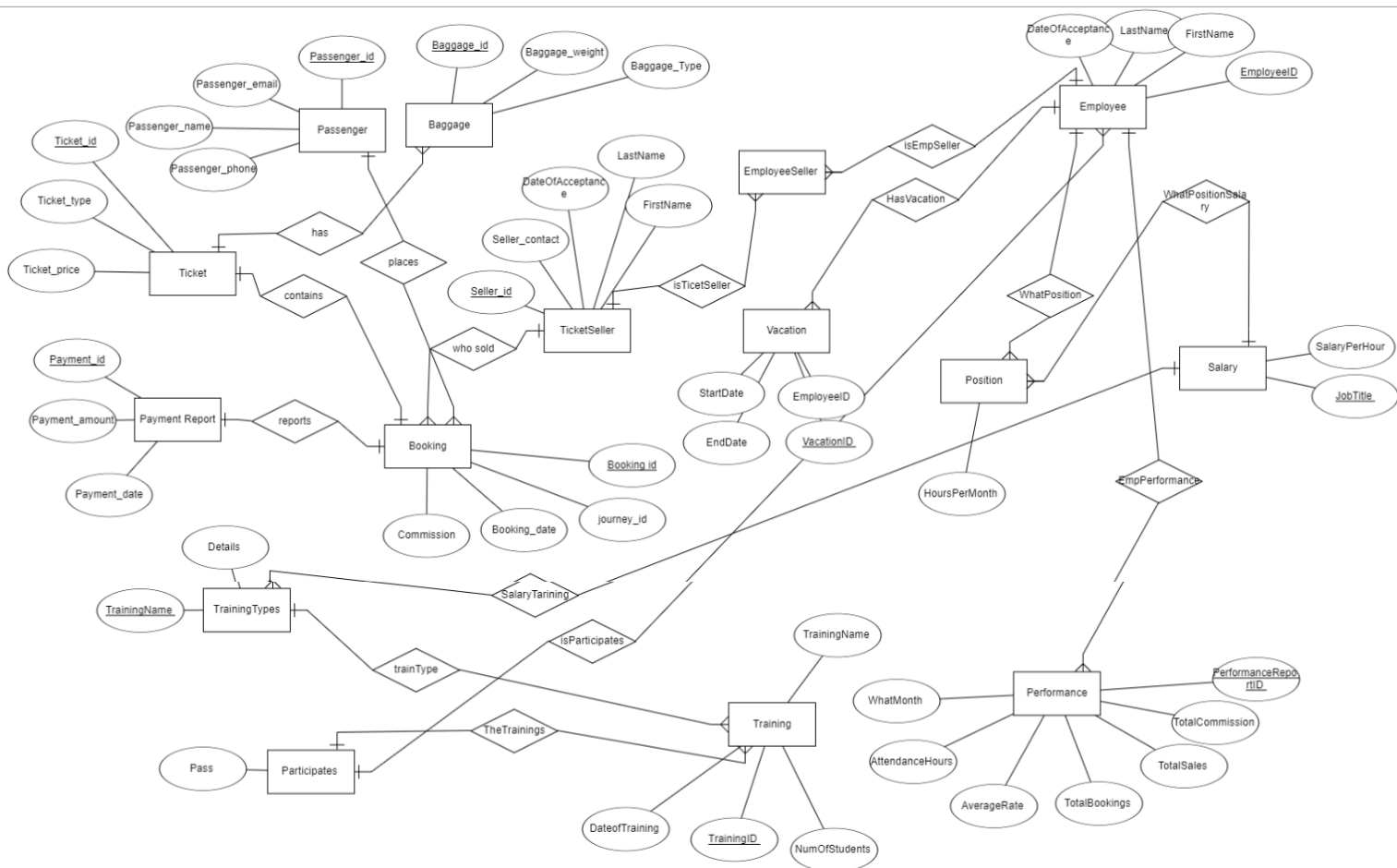




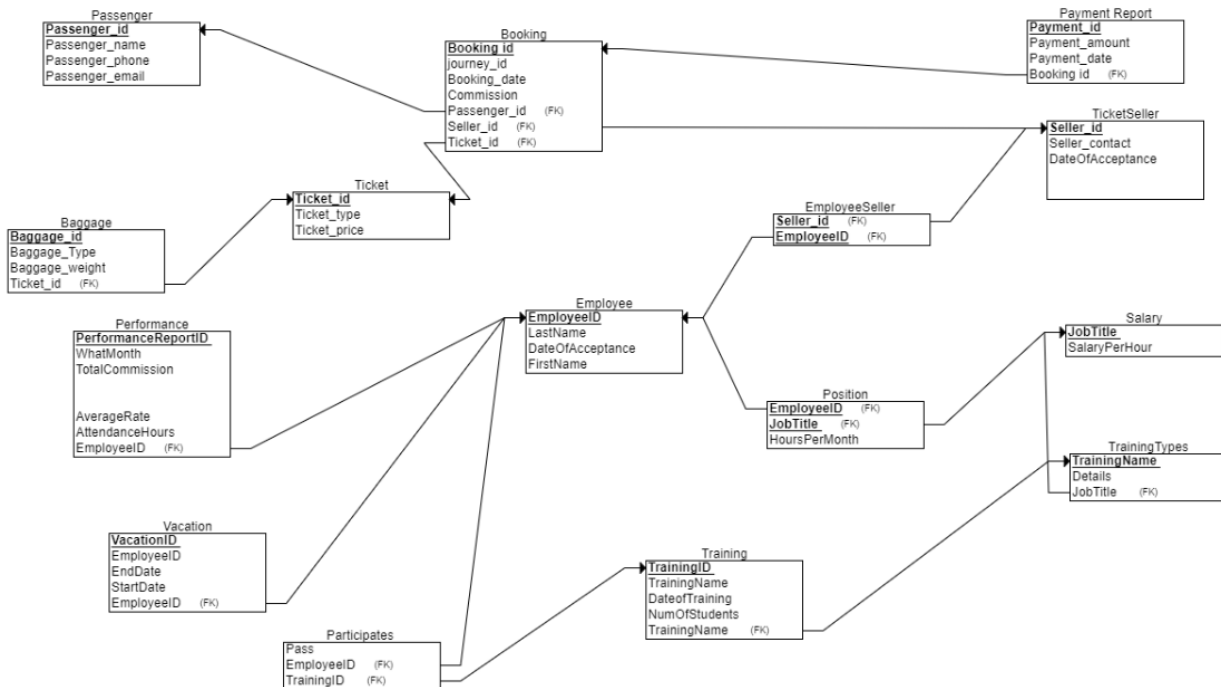
:ERD



## ERD משותף



## DSD משותף



## הסבר שינויים ואינטגרציה

טבלת העובדים ממסד הנתונים שקיבלנו (החדשה) נותרה ללא שינוי ברובה, אך כעת יש לה קשר עקיף לטבלת TicketSeller דרך טבלת EmployeeSeller חדשה. זה מאפשר לעובדים לקחת על עצמם תפקידים כמוכרי כרטיסים מבלי לשנות את מבנה הנתונים המרכזי של העובדים. טבלת הביצועים הורחבה כך שתכלול את סך העמלות TotalCommission, המאפשרת תצוגה מקיפה יותר העמלות המגיעות רק למוכרי כרטיסים, ששכרם הכי נמוך בSALARY משום שהוא מבוסס גם על העמלות.

נוצרו קשרים חדשים בין שתי המערכות המקוריות. טבלת EmployeeSeller משמשת כגשר, ומאפשרת לעובדים להיות משויכים לתפקידי מוכר כרטיסים מבלי לשנות מהותית את טבלת העובד או TicketSeller. כלומר, יהיו מוכרי כרטיסים חיצוניים ופנימיים. נניח שמכירת כרטיסים ניתן לעשות כעובד חברה וגם כעובד מבחוץ. לצורך כך, נשנה את הטבלה של מוכר כרטיסים ונשנה את השם שלו לNICKNAME ואם הוא עובד בחברה, אז הפרטים שלו יישמרו בEMPLOYEE וEmployeeSeller זה שומר על שלמות שתי המערכות המקוריות תוך הפעלת פונקציות חדשות. טבלת הביצועים המורחבת כעת יכולה לשקף הן מדדי עובדים מסורתיים והן ביצועי מכירות, ומספקת ראייה הוליסטית יותר של תרומות העובדים.

מסד הנתונים המשולב שומר על כל הפונקציונליות המקורית של שתי המערכות תוך הפעלת יכולות חדשות. זה מאפשר מעקב אחר ביצועי עובדים בתפקידים שונים, כולל מכירות. המערכת יכולה כעת לשייך הזמנות ומכירות לעובדים ספציפיים, לעקוב אחר עמלות ולספק מבט מקיף יותר הן על ניהול העובדים והן על פעולות מכירת הכרטיסים. אינטגרציה זו מאפשרת דיווח וניתוח מתוחכמים יותר, כגון זיהוי עובדים בעלי ביצועים גבוהים על פני מדדים שונים, מעקב אחר התקדמות הקריירה מעובד רגיל למוכר כרטיסים, וניתוח השפעת ההדרכה על ביצועי המכירות.

## פירוט המימד הטכני של השינויים

### 1. טבלאות ששוננו:

א. עובד:

- ללא שינויים מבניים, אך כעת מקושר בעקיפין ל-TicketSeller דרך טבלת EmployeeSeller

ב. ביצועים:

- עמודות שנוספו: TotalCommission (FLOAT)

ג. מוכר כרטיסים:

- עמודות ששוננו: שם לכינוי.

ד. הזמנה:

- נוספה עמודה: עמלה (FLOAT)

### 2. טבלאות חדשות:

EmployeeSeller .

- עמודות: EmployeeID (INT), Seller\_id (NUMBER(38))

- מפתח ראשי: (EmployeeID, Seller\_id)

- מפתחות זרים:

- הפנייה ל EmployeeID Employee(EmployeeID)

- Seller\_id הפניות TicketSeller(Seller\_id)

### 3. חיבורים חדשים:

א. עובד למוכר כרטיסים:

- חיבור עקיף דרך טבלת EmployeeSeller

- מאפשר לעובדים להיות משויכים לתפקידי מוכר כרטיסים ללא שינוי ישיר של אף אחת מהטבלאות

ב. עובד לביצועים:

- החיבור הקיים כולל כעת מדדי ביצועי מכירות

ג. מוכר כרטיסים להזמנה:

- החיבור הקיים כולל כעת מידע על עמלות

ד. הזמנה לביצועים:

- קשר עקיף דרך EmployeeSeller ו- Employee

- מאפשר להזמנות לתרום למדדי ביצועי עובדי

### אינטגרציה ברמת הבנייה ואכלוס

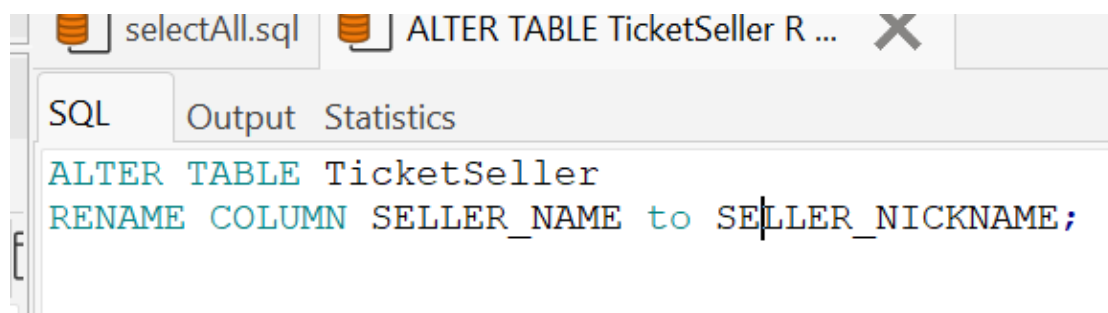
עלינו לבצע את השינויים הנדרשים בDB השונים על מנת להגיע לאינטגרציה של DB המשותף, כפי שהוא משתקף מהERD:

שינוי בטבלת Performance:

```
ALTER TABLE performance  
ADD column TotalCommisions;
```

הוספנו את השדה החדש, המחשב את העמלה לכל העובדים שמגיע להם עמלה כלשהי.

### שינוי שדות



### הוספת שדה בBOOKING

```
ALTER TABLE Booking  
ADD Commission FLOAT;
```

### יצירת טבלה EMPLOYEESELLER

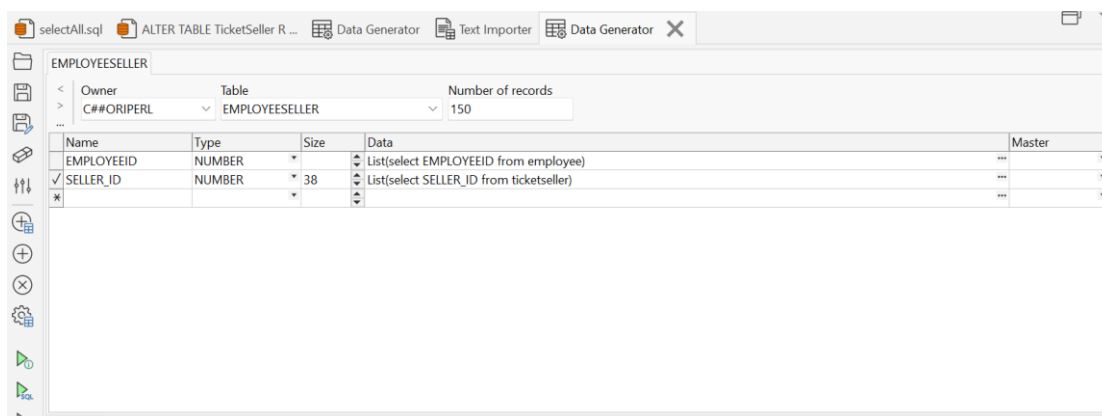
```
SQL Output Statistics  
CREATE TABLE EmployeeSeller (  
    EmployeeID INT,  
    Seller_id NUMBER(38),  
    PRIMARY KEY (EmployeeID, Seller_id),  
    FOREIGN KEY (EmployeeID) REFERENCES Employee(EmployeeID),  
    FOREIGN KEY (Seller_id) REFERENCES TicketSeller(Seller_id)  
);
```

### עדכון ואכלוס הנתונים

נרצה לאכלס את הטבלה SELLEREMPLOYEE, בעצם לעדכן שחלק מהעובדים יהיו מוכרי כרטיסים ושהמוכרי כרטיסים יהיו עובדים

אכלוס הטבלה SELLEREMPLOYEE:

נבחר 150 מכל סוג ונכניס דרך data generator:



נוסיף SALARY של מוכר כרטיסים ונעדכן את העובדים הללו

```

SQL      Output  Statistics
UPDATE POSITION
SET JOBTITLE = 'ticketseller'
WHERE EMPLOYEEID IN (SELECT EMPLOYEEID FROM EMPLOYEESELLER);

```

טבלת POSITION לאחר השינוי:

	EMPLOYEEID	JOBTITLE	HOURSPERMONTH
36	148892712	bus driver	195
37	269615388	bus driver	216
38	927785924	bus driver	84
39	595135204	bus driver	213
40	839974770	accountant	215
41	662599026	ticketseller	203
42	646866861	bus driver	130
43	822559017	ticketseller	125

נעדכן גם את עמודת COMMISSION של BOOKING שתהיה עשירית ממחיר הכרטיס הנמכר

SQLOutputStatistics

```

UPDATE BOOKING b
SET b.COMMISSION = (
    SELECT t.Ticket_price / 10
    FROM TICKET t
    WHERE t.Ticket_id = b.Ticket_id
);

```

	BOOKING_ID	BOOKING_DATE	PASSENGER_ID	SELLER_ID	TICKET_ID	COMMISSION
1	1101	11/06/2023	3554560914	325237214	256940	1.5
2	1102	30/05/2023	1241083848	297909520	258498	1.5
3	1000	08/06/2023	1573871528	234413401	200137	1.5
4	1001	24/06/2023	4139544263	214945813	200361	0.5
5	1002	27/06/2023	379982361	332113182	200561	1
6	1003	06/06/2023	334694516	207083715	200579	1.5
7	1004	18/06/2023	342850352	386178461	201113	1.5
8	1005	08/06/2023	211756555	217056502	201163	0.5
9	1006	14/06/2023	277761157	281154307	201894	1

נוסיף את סך העמלות לכל עובד ב: PERFORMANCE

SQLOutputStatistics

```

UPDATE performance p
SET p.TOTALCOMMISSION = (
    SELECT COALESCE(SUM(b.COMMISSION), 0)
    FROM booking b
    JOIN employeeseller es ON b.SELLER_ID = es.Seller_id
    WHERE es.employeeID = p.EMPLOYEEID
)
WHERE p.EMPLOYEEID IN (
    SELECT DISTINCT employeeID
    FROM employeeseller
);

```

נוסיף סוג אימון חדש עבור מוכר כרטיסים ונשייך אליו את כל המוכרי כרטיסים:

הוספת סוג אימון חדש:

```

INSERT INTO TRAININGTYPES (TRAININGNAME, DETAILS, JOBTITLE)
VALUES ('Sales Training', ' training on sales techniques and customer interaction for ticketsellers.', 'ticketseller');

```

נוסיף אותו ל: TRAINING



```
selectall.sql  INSERT INTO TRAINING (TRAININGID, TRAININGNAME, NUMOFSTUDENTS, DATEOFTRAINING)
SQL  Output  Statistics
INSERT INTO TRAINING (TRAININGID, TRAININGNAME, NUMOFSTUDENTS, DATEOFTRAINING)
VALUES ('trng103001', 'Sales Training', 50, TO_DATE('2024-07-20', 'YYYY-MM-DD'));
```

כעת נשנה את כל העובדים שעובדים כמוכרי כרטיסים שעשו השתלמות אחרת, שיעשו השתלמות של עובדי כרטיסים המתאימה:

```
UPDATE participates
SET TRAININGID = 'trng103001'
WHERE EMPLOYEEID IN (SELECT EMPLOYEEID FROM EmployeeSeller);
|
```

## אינטגרציה – רמת התשאול

### מבט על האגף המקורי

מבט זה משלב מידע על המוכרים (TicketSeller) ועל המכירות (Booking) שבוצעו על ידם. המבט מציג את הפרטים על המוכרים יחד עם המידע על המכירות שהם ביצעו, כולל מחירי הכרטיסים והעמלות שנגבו.

```
CREATE OR REPLACE VIEW TicketSales_View AS
SELECT
    ts.Seller_id,
    ts.SELLER_NICKNAME,
    b.Booking_id,
    t.Ticket_id,
    t.Ticket_price,
    t.Ticket_type,
    b.Booking_date
FROM
    TicketSeller ts
JOIN
    Booking b ON ts.Seller_id = b.Seller_id
JOIN
    Ticket t ON b.Ticket_id = t.Ticket_id;
```

### שאלתה למציאת כל מכירות הכרטיסים של מוכר מסוים:

The screenshot shows a SQL query editor with the following query:

```
SELECT
    SELLER_NICKNAME,
    Booking_id,
    Ticket_id,
    Ticket_price,
    Booking_date
FROM
    TicketSales_View
WHERE
    SELLER_NICKNAME = 'Moraz';
```

Below the query editor, there is a toolbar with various icons for query execution and formatting. At the bottom, a table displays the results of the query:

	SELLER_NICKNAME	BOOKING_ID	TICKET_ID	TICKET_PRICE	BOOKING_DATE
1	Moraz	1004	201113	15	18/06/2023
2	Moraz	1385	378711	15	28/06/2023

שאלתה למציאת סך מכירות הכרטיסים של כל המוכרים:

```
SELECT
    SELLER_NICKNAME,
    COUNT(Ticket_id) AS Total_Tickets_Sold,
    SUM(Ticket_price) AS Total_Revenue
FROM
    TicketSales_View
GROUP BY
    SELLER_NICKNAME
ORDER BY
    Total_Revenue DESC;
```

מבט על האגף החדש

מבט זה משלב מידע מטבלאות שונות על תפקידים, שכר (Position) וסוגי הכשרות (TrainingTypes) המבט מציג את התפקידים השונים במערכת יחד עם השכר המשוך לכל תפקיד וההכשרות הנדרשות לכל תפקיד.

```
CREATE OR REPLACE VIEW Position_Salary_Training_View AS
SELECT
    p.EMPLOYEEID,
    p.JOBTITLE,
    s.JOBTITLE AS Salary_JobTitle,
    s.SALARYPERHOUR AS Salary,
    tt.TRAININGNAME,
    tt.DETAILS AS Training_Details
FROM
    POSITION p
LEFT JOIN
    SALARY s ON p.JOBTITLE = s.JOBTITLE
LEFT JOIN
    TRAININGTYPES tt ON p.JOBTITLE = tt.JOBTITLE;
```

## שאלתה 1: מציאת כל העובדים עם שכר וסוגי הכשרות כולל ממוצע השכר ותיאורי הכשרות מסוימות

SQL Output Statistics

```

SELECT
  EMPLOYEEID,
  JOBTITLE,
  Salary,
  TRAININGNAME,
  Training_Details,
  (SELECT AVG(Salary) FROM Position_Salary_Training_View) AS Average_Salary,
  (SELECT COUNT(*) FROM Position_Salary_Training_View WHERE Training_Details IS NOT NULL) AS Training_Count
FROM
  Position_Salary_Training_View;

```

	EMPLOYEEID	JOBTITLE	SALARY	TRAININGNAME	TRAINING_DETAILS	AVERAGE_SALARY	TRAINING_Count
1	111502154	ticketseller	35	Sales Training	training on sales techniques and customer interaction for ticketseller	46.2265415549598	1
2	113147010	bus driver	45	Bus Driver Safety Training2	Covers regulations, defensive driving techniques, and emergency pro	46.2265415549598	1
3	113147010	bus driver	45	Bus Driver Customer Service Training	Focuses on communication skills, conflict resolution, and passenger s	46.2265415549598	1
4	113147010	bus driver	45	Bus Driver Route Planning and Navigation	Teaches efficient route planning, map reading, and GPS usage.	46.2265415549598	1
5	113147010	bus driver	45	Bus Driver Maintenance and Inspection	Provides knowledge on basic bus maintenance and pre-trip inspectio	46.2265415549598	1
6	113147010	bus driver	45	Bus Driver First Aid and CPR	Trains in emergency response procedures for injuries and medical en	46.2265415549598	1
7	113996804	bus driver	45	Bus Driver Safety Training2	Covers regulations, defensive driving techniques, and emergency pro	46.2265415549598	1
8	113996804	bus driver	45	Bus Driver Customer Service Training	Focuses on communication skills, conflict resolution, and passenger s	46.2265415549598	1
9	113996804	bus driver	45	Bus Driver Route Planning and Navigation	Teaches efficient route planning, map reading, and GPS usage.	46.2265415549598	1
10	113996804	bus driver	45	Bus Driver Maintenance and Inspection	Provides knowledge on basic bus maintenance and pre-trip inspectio	46.2265415549598	1
11	113996804	bus driver	45	Bus Driver First Aid and CPR	Trains in emergency response procedures for injuries and medical en	46.2265415549598	1
12	120969869	ticketseller	35	Sales Training	training on sales techniques and customer interaction for ticketseller	46.2265415549598	1
13	121463098	bus driver	45	Bus Driver Safety Training2	Covers regulations, defensive driving techniques, and emergency pro	46.2265415549598	1

## שאלתה 2: מציאת כל העובדים בתפקיד מסוים עם פרטי השכר וההכשרות שלהם, ממוינים לפי השכר הגבוה ביותר (כולם אותו שכר אבל ניחא)

SQL Output Statistics

```

SELECT
  EMPLOYEEID,
  JOBTITLE,
  Salary,
  TRAININGNAME,
  Training_Details,
  RANK() OVER (ORDER BY Salary DESC) AS Salary_Rank
FROM
  Position_Salary_Training_View
WHERE
  JOBTITLE = 'bus driver'
ORDER BY
  Salary DESC;

```

	EMPLOYEEID	JOBTITLE	SALARY	TRAININGNAME	TRAINING_DETAILS	SALARY_RANK
1	113147010	bus driver	45	Bus Driver Safety Training2	Covers regulations, defensive driving techniques, and emergency pro	1
2	113147010	bus driver	45	Bus Driver Customer Service Training	Focuses on communication skills, conflict resolution, and passenger s	1
3	113147010	bus driver	45	Bus Driver Route Planning and Navigation	Teaches efficient route planning, map reading, and GPS usage.	1
4	113147010	bus driver	45	Bus Driver Maintenance and Inspection	Provides knowledge on basic bus maintenance and pre-trip inspectio	1
5	113147010	bus driver	45	Bus Driver First Aid and CPR	Trains in emergency response procedures for injuries and medical en	1
6	113996804	bus driver	45	Bus Driver Safety Training2	Covers regulations, defensive driving techniques, and emergency pro	1
7	113996804	bus driver	45	Bus Driver Customer Service Training	Focuses on communication skills, conflict resolution, and passenger s	1
8	113996804	bus driver	45	Bus Driver Route Planning and Navigation	Teaches efficient route planning, map reading, and GPS usage.	1
9	113996804	bus driver	45	Bus Driver Maintenance and Inspection	Provides knowledge on basic bus maintenance and pre-trip inspectio	1
10	113996804	bus driver	45	Bus Driver First Aid and CPR	Trains in emergency response procedures for injuries and medical en	1