FINAL PROJECT DATA BASE

DELIVERY COMPANY

Presented by:
Guy Ben Moshe 318363397
Ariel Yogev 207917501

CONTENTS

The purpose of the system

- **Manage People**: It handles information about clients who order food and the deliverers who bring the food to them.
- **Track Deliveries:** It keeps track of every delivery, like who ordered, which restaurant the food came from, and which deliverer handled it.
- **Handle Restaurants & Dishes**: It stores details about restaurants and their menus, making sure each restaurant's dishes are linked to the right place.
- **Addresses & Time**: It stores where the food is being delivered and tracks when orders are placed and delivered.

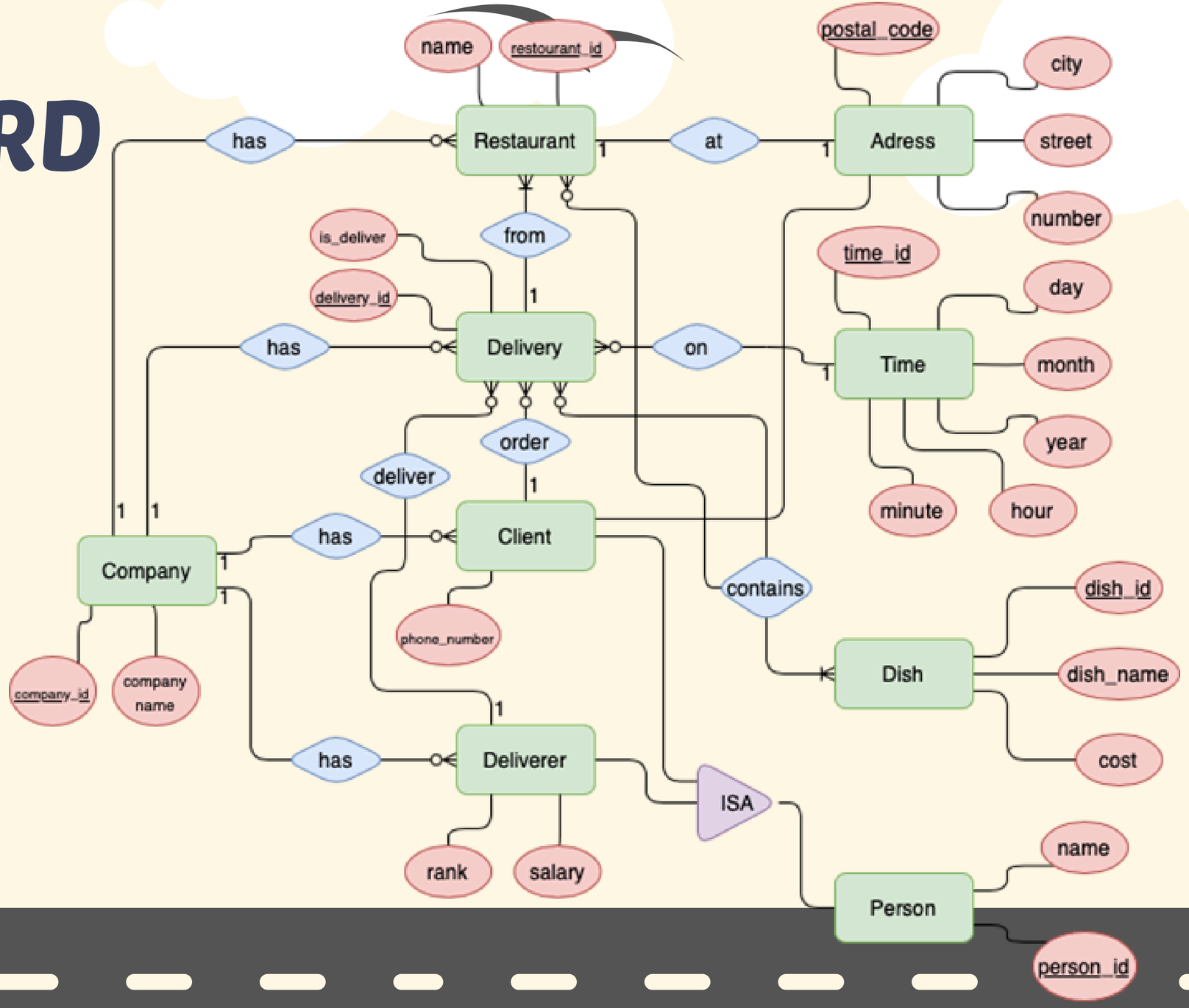
In short, it’s a system to make sure food gets from restaurants to clients, while keeping track of all the important details in between.

System requirements

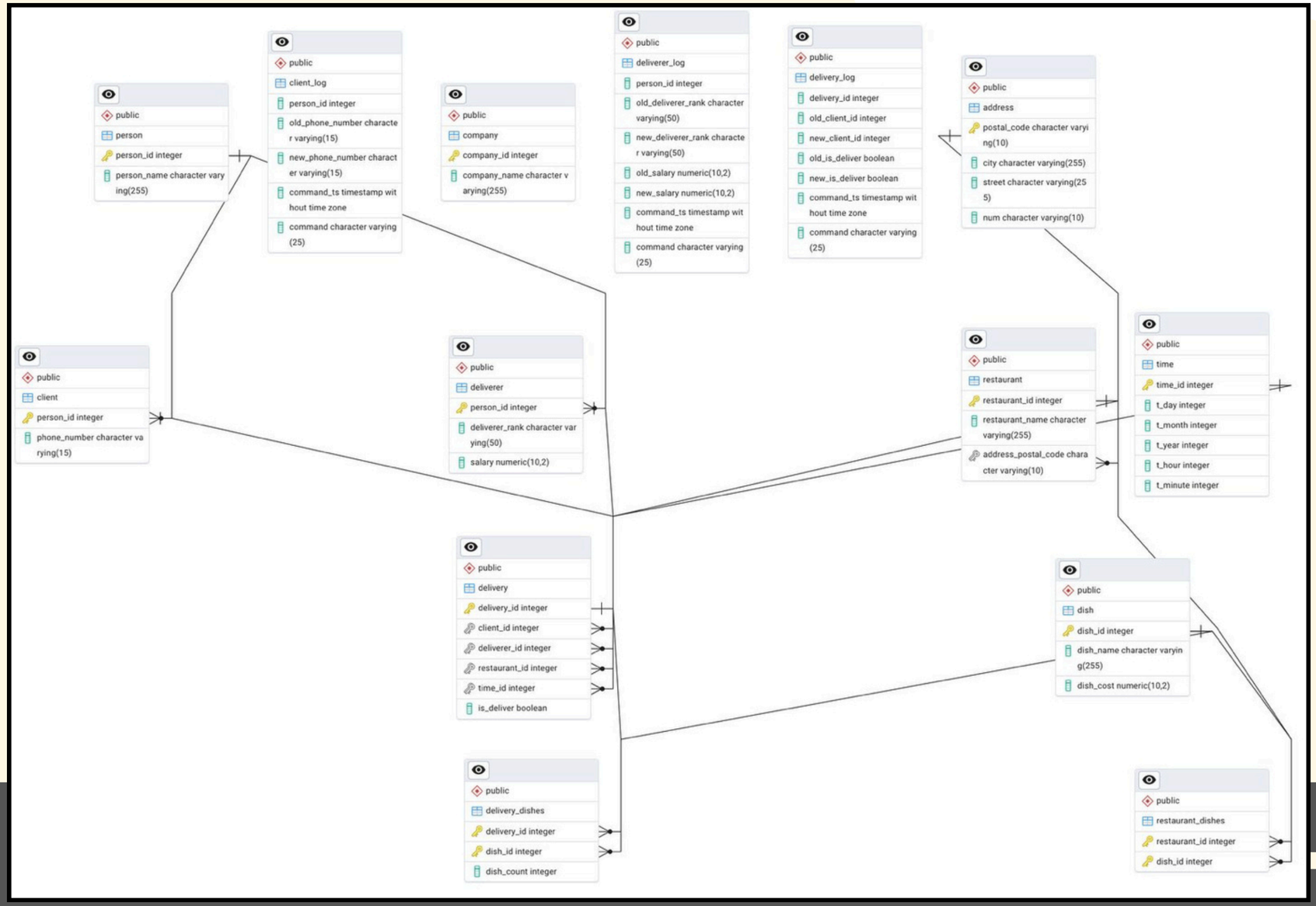
- To optimize the flow of information within a large-scale food delivery service, there is a clear need for a unified system that connects data across various entities.

- The system must be capable of creating, retrieving, updating, deleting, and displaying relationships between different entities involved in the delivery process.

Logical ERD

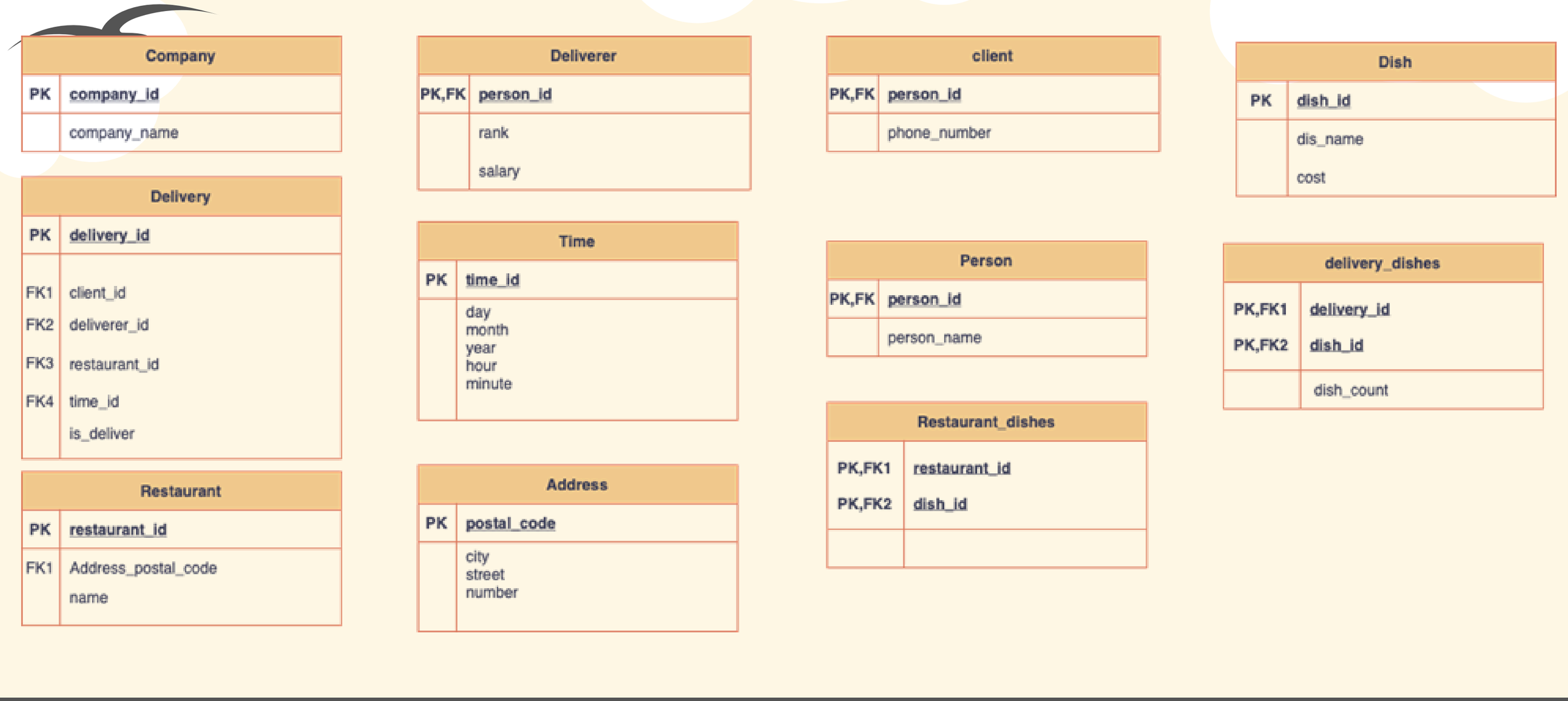

name
restourant_id
postal_code
city
street
number
Restaurant
has
at
Adress
from
is_deliver
delivery_id
time_id
day
Delivery
has
on
Time
month
year
minute
hour
order
deliver
Client
has
Company
contains
phone_number
company_id
company name
dish_id
dish_name
cost
Dish
has
Deliverer
ISA
rank
salary
name
Person
person_id

Physical ERD



Entities

- Company
- Delivery
- Restaurant
- Address
- Deliverer
- Person
- Client
- Time
- Dish
- Delivery_Dish
- Restaurant_dishes



Company	
PK	company_id
	company_name

Delivery	
PK	delivery_id
FK1	client_id
FK2	deliverer_id
FK3	restaurant_id
FK4	time_id
	is_deliver

Restaurant	
PK	restaurant_id
FK1	Address_postal_code
	name

Deliverer	
PK,FK	person_id
	rank
	salary

Time	
PK	time_id
	day month year hour minute

Address	
PK	postal_code
	city street number

client	
PK,FK	person_id
	phone_number

Person	
PK,FK	person_id
	person_name

Restaurant_dishes	
PK,FK1	restaurant_id
PK,FK2	dish_id

Dish	
PK	dish_id
	dis_name
	cost

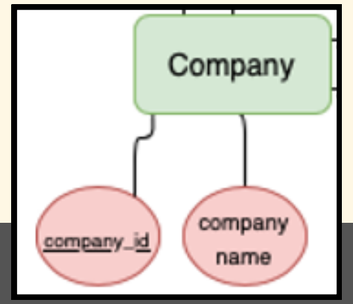
delivery_dishes	
PK,FK1	delivery_id
PK,FK2	dish_id
	dish_count

Company

Represents the organization responsible for overseeing the delivery operations.

- **company_id:** Unique identifier for the company.
- **name**

```sql
-- Create the Company table
CREATE TABLE Company (
    company_id INT NOT NULL,
    company_name VARCHAR(255),
    PRIMARY KEY (company_id)
);
```




```sql
INSERT INTO Company (company_id, company_name) VALUES
(1, 'Tolo');
```

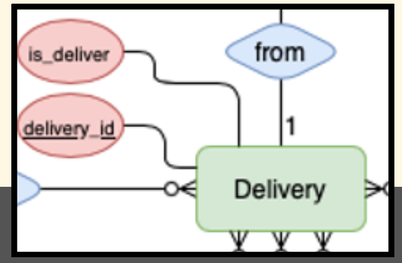
	company_id [PK] integer	company_name character varying (255)
1	1	Tolo

Delivery

Tracks the individual deliveries made to clients.

- **delivery_id**: Unique identifier for each delivery.
- **time_id**
- **client_id**
- **deliverer_id**
- **restaurant_id**
- **is_deliver**

```sql
-- Create the Delivery table (FKs referencing Client, Deliverer, Restaurant, Time)
CREATE TABLE Delivery (
    delivery_id INT NOT NULL,
    client_id INT,
    deliverer_id INT,
    restaurant_id INT,
    time_id INT,
    is_deliver BOOLEAN,  -- Added boolean field to indicate delivery status
    PRIMARY KEY (delivery_id),
    FOREIGN KEY (client_id) REFERENCES Client(person_id),
    FOREIGN KEY (deliverer_id) REFERENCES Deliverer(person_id),
    FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id),
    FOREIGN KEY (time_id) REFERENCES Time(time_id)
);
```




```sql
-- Insert Deliveries
INSERT INTO Delivery (delivery_id, client_id, deliverer_id, restaurant_id, time_id, is_de
(1, 6, 1, 1, 1, FALSE),
(2, 7, 2, 2, 2, TRUE),
(3, 8, 3, 3, 3, TRUE),
(4, 9, 4, 1, 4, FALSE),
(5, 10, 5, 2, 5, TRUE),
(6, 11, 6, 4, 6, TRUE),
(7, 12, 7, 5, 7, TRUE),
(8, 13, 8, 6, 8, FALSE),
(9, 14, 9, 7, 9, TRUE),
(10, 15, 10, 8, 10, FALSE),
(11, 6, 1, 9, 1, TRUE),
(12, 7, 2, 10, 2, TRUE),
(13, 8, 3, 1, 3, TRUE),
(14, 9, 4, 2, 4, TRUE),
(15, 10, 5, 3, 5, TRUE),
```

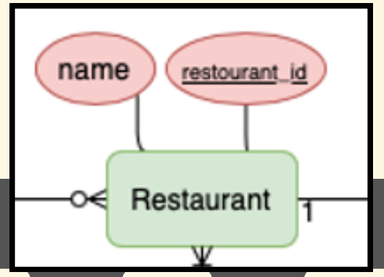
	delivery_id [PK] integer	client_id integer	deliverer_id integer	restaurant_id integer	time_id integer	is_deliver boolean
1	1	6	1	1	1	false
2	2	7	2	2	2	true
3	3	8	3	3	3	true
4	4	9	4	1	4	false
5	5	10	5	2	5	true
6	6	11	6	4	6	true
7	7	12	7	5	7	true
8	8	13	8	6	8	false
9	9	14	9	7	9	true
10	10	15	10	8	10	false

Restaurant

Represents the restaurant from which the delivery is sourced.

- **restaurant_id**: Unique identifier for the restaurant.
- **restaurant_name**
- **address_postal_code**

```sql
-- Create the Restaurant table (FK referencing Address)
CREATE TABLE Restaurant (
    restaurant_id INT NOT NULL,
    name VARCHAR(255),
    address_postal_code VARCHAR(10),
    PRIMARY KEY (restaurant_id),
    FOREIGN KEY (address_postal_code) REFERENCES Address(postal_code)
);
```




```sql
-- Insert Restaurants
INSERT INTO Restaurant (restaurant_id, restaurent_name, address_pos
(1, 'Sabich Shel Oved', '12345'),
(2, 'Pizza Haim', '67890'),
(3, 'Falafel Golani', '54321'),
(4, 'Sushi Express', '11223'),
(5, 'Pasta La Vista', '33445'),
(6, 'Grill House', '55667'),
(7, 'Taco Bell', '77889'),
(8, 'Chicken Delight', '99887'),
(9, 'Hummus & More', '11111'),
(10, 'Vegan Vibes', '22222');
```

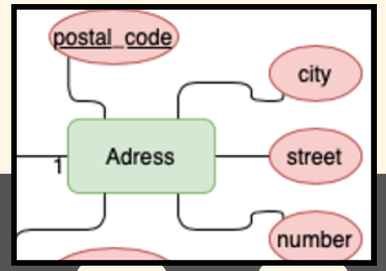
	restaurant_id [PK] integer	restaurent_name character varying (255)	address_postal_code character varying (10)
1	1	Sabich Shel Oved	12345
2	2	Pizza Haim	67890
3	3	Falafel Golani	54321
4	4	Sushi Express	11223
5	5	Pasta La Vista	33445
6	6	Grill House	55667
7	7	Taco Bell	77889
8	8	Chicken Delight	99887
9	9	Hummus & More	11111
10	10	Vegan Vibes	22222

Address

Contains address details for both restaurants and clients.

- **postal_code**: Unique identifier for each address.
- **city**
- **street**
- **num**

```sql
-- Create the Address table
CREATE TABLE Address (
    postal_code VARCHAR(10) NOT NULL,
    city VARCHAR(255),
    street VARCHAR(255),
    number VARCHAR(10),
    PRIMARY KEY (postal_code)
);
```




```sql
-- Insert Addresses
INSERT INTO Address (postal_code, city, street, num) VALUES
('12345', 'Tel Aviv', 'Dizengoff', '100'),
('67890', 'Jerusalem', 'Jaffa', '200'),
('54321', 'Haifa', 'Herzel', '300'),
('11223', 'Netanya', 'Sderot', '50'),
('33445', 'Ashdod', 'Yaffo', '150'),
('55667', 'Raanana', 'Chaim Weizmann', '250'),
('77889', 'Petah Tikva', 'Hahagana', '90'),
('99887', 'Herzliya', 'Hertzel', '30'),
('11111', 'Rehovot', 'Rabin', '70'),
('22222', 'Bat Yam', 'Ben Gurion', '80');
```

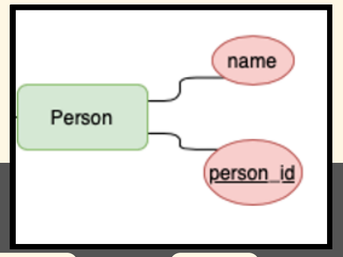
	postal_code [PK] character varying (10)	city character varying (255)	street character varying (255)	num character varying (10)
1	12345	Tel Aviv	Dizengoff	100
2	67890	Jerusalem	Jaffa	200
3	54321	Haifa	Herzel	300
4	11223	Netanya	Sderot	50
5	33445	Ashdod	Yaffo	150
6	55667	Raanana	Chaim Weizmann	250
7	77889	Petah Tikva	Hahagana	90
8	99887	Herzliya	Hertzel	30
9	11111	Rehovot	Rabin	70
10	22222	Bat Yam	Ben Gurion	80

Person

A general entity representing individuals involved.

- **person_id:** Unique identifier for each person
- **person_name**

```sql
-- Create the Person table
CREATE TABLE Person (
    person_id INT NOT NULL,
    person_name VARCHAR(255),
    PRIMARY KEY (person_id)
);
```




```sql
-- Insert People (Deliverers and Clients)
INSERT INTO Person (person_id, person_name) VALUES
(1, 'Moshe Levi'),
(2, 'David Cohen'),
(3, 'Shlomi Peretz'),
(4, 'Eli Ben-Ari'),
(5, 'Avi Harush'),
(6, 'Yael Katz'),
(7, 'Rina Shalom'),
(8, 'Haim Levy'),
(9, 'Noa Cohen'),
(10, 'Eden Malka'),
(11, 'Omer Yaakov'),
(12, 'Nir Shlomo'),
(13, 'Tali Amir'),
(14, 'Or Ben-Gurion'),
(15, 'Idan Eliraz');
```

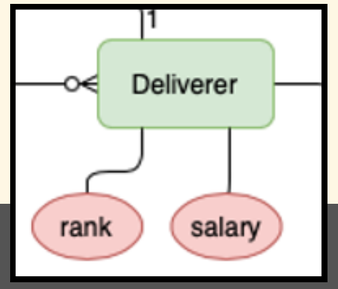
	person_id [PK] integer	person_name character varying (255)
1	1	Moshe Levi
2	2	David Cohen
3	3	Shlomi Peretz
4	4	Eli Ben-Ari
5	5	Avi Harush
6	6	Yael Katz
7	7	Rina Shalom
8	8	Haim Levy
9	9	Noa Cohen
10	10	Eden Malka
11	11	Omer Yaakov
12	12	Nir Shlomo
13	13	Tali Amir
14	14	Or Ben-Gurion
15	15	Idan Eliraz

Deliverer

The person responsible for delivering the orders.

- **person_id**: Unique identifier for each person
- **deliverer_rank**
- **salary**

```sql
-- Create the Deliverer table (FK referencing Person)
CREATE TABLE Deliverer (
    person_id INT NOT NULL,
    rank VARCHAR(50),
    salary DECIMAL(10, 2),
    PRIMARY KEY (person_id),
    FOREIGN KEY (person_id) REFERENCES Person(person_id)
);
```




```sql
-- Insert Deliverers' Details
INSERT INTO Deliverer (person_id, deliverer_rank, salary) VALUES
(1, 'Senior', 15000),
(2, 'Junior', 12000),
(3, 'Junior', 13000),
(4, 'Senior', 16000),
(5, 'Intermediate', 14000),
(6, 'Junior', 12500),
(7, 'Senior', 15500),
(8, 'Intermediate', 13500),
(9, 'Junior', 11000),
(10, 'Intermediate', 14500);
```

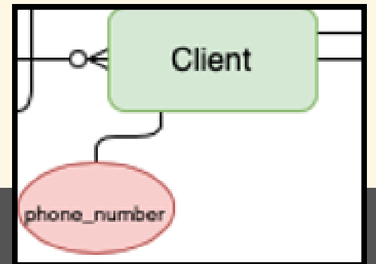
	person_id [PK] integer	deliverer_rank character varying (50)	salary numeric (10,2)
1	1	Senior	15000.00
2	2	Junior	12000.00
3	3	Junior	13000.00
4	4	Senior	16000.00
5	5	Intermediate	14000.00
6	6	Junior	12500.00
7	7	Senior	15500.00
8	8	Intermediate	13500.00
9	9	Junior	11000.00
10	10	Intermediate	14500.00

Client

Represents the customers receiving deliveries.

- **person_id:** Unique identifier for each person
- **phone_number**

```sql
-- Create the Client table (FK referencing Person)
CREATE TABLE Client (
    person_id INT NOT NULL,
    phone_number VARCHAR(15),
    PRIMARY KEY (person_id),
    FOREIGN KEY (person_id) REFERENCES Person(person_id)
);
```




```sql
-- Insert Clients (Customers)
INSERT INTO Client (person_id, phone_number) VALUES
(6, '054-1112233'),
(7, '054-2244333'),
(8, '054-3344555'),
(9, '054-4455666'),
(10, '054-5566777'),
(11, '054-6677888'),
(12, '054-7788999'),
(13, '054-8899000'),
(14, '054-9900112'),
(15, '054-1001223');
```

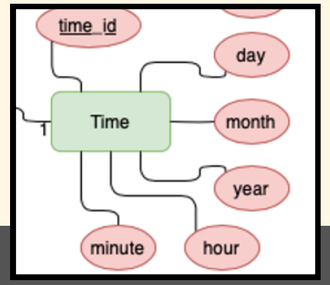
	person_id [PK] integer	phone_number character varying (15)
1	6	054-1112233
2	7	054-2244333
3	8	054-3344555
4	9	054-4455666
5	10	054-5566777
6	11	054-6677888
7	12	054-7788999
8	13	054-8899000
9	14	054-9900112
10	15	054-1001223

Time

A general entity for handling time-related data.

- **time_id:** Unique identifier for each time record.
- **t_day**
- **t_month**
- **t_year**
- **t_hour**
- **t_minute**

```sql
-- Create the Time table
CREATE TABLE Time (
    time_id INT NOT NULL,
    day INT,
    month INT,
    year INT,
    hour INT,
    minute INT,
    PRIMARY KEY (time_id)
);
```




```sql
-- Insert Time Entries
INSERT INTO Time (time_id, t_day, t_month, t_year, t_hour, t_minute
(1, 12, 7, 2023, 14, 30),
(2, 15, 8, 2023, 18, 45),
(3, 20, 9, 2023, 12, 15),
(4, 5, 10, 2023, 19, 50),
(5, 1, 11, 2023, 13, 25),
(6, 10, 12, 2023, 17, 0),
(7, 20, 11, 2023, 20, 30),
(8, 25, 10, 2023, 16, 15),
(9, 30, 9, 2023, 15, 45),
(10, 3, 8, 2023, 14, 5);
```

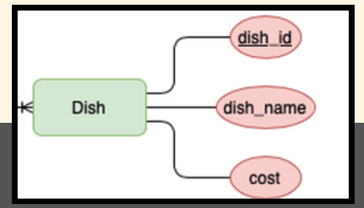
	time_id [PK] integer	t_day integer	t_month integer	t_year integer	t_hour integer	t_minute integer
1	1	12	7	2023	14	30
2	2	15	8	2023	18	45
3	3	20	9	2023	12	15
4	4	5	10	2023	19	50
5	5	1	11	2023	13	25
6	6	10	12	2023	17	0
7	7	20	11	2023	20	30
8	8	25	10	2023	16	15
9	9	30	9	2023	15	45
10	10	3	8	2023	14	5

Dish

Represents the individual food items available for order from a restaurant.

- **dish_id: Unique identifier for each dish**
- **dish_name**
- **dish_cost**

```sql
-- Create the Dish table
CREATE TABLE Dish (
    dish_id INT NOT NULL,
    dish_name VARCHAR(255),
    cost DECIMAL(10, 2),
    PRIMARY KEY (dish_id)
);
```




```sql
-- Insert Addresses
INSERT INTO Address (postal_code, city, street, num) VALUES
('12345', 'Tel Aviv', 'Dizengoff', '100'),
('67890', 'Jerusalem', 'Jaffa', '200'),
('54321', 'Haifa', 'Herzel', '300'),
('11223', 'Netanya', 'Sderot', '50'),
('33445', 'Ashdod', 'Yaffo', '150'),
('55667', 'Raanana', 'Chaim Weizmann', '250'),
('77889', 'Petah Tikva', 'Hahagana', '90'),
('99887', 'Herzliya', 'Hertzel', '30'),
('11111', 'Rehovot', 'Rabin', '70'),
('22222', 'Bat Yam', 'Ben Gurion', '80');
```

	dish_id [PK] integer	dish_name character varying (255)	dish_cost numeric (10,2)
1	1	Shawarma	45.00
2	2	Falafel	30.00
3	3	Schnitzel	50.00
4	4	Hummus Plate	25.00
5	5	Pizza	60.00
6	6	Sushi	75.00
7	7	Pasta	55.00
8	8	Grilled Chicken	70.00
9	9	Vegetable Stir-Fry	40.00
10	10	Tacos	50.00

Delivery_Dishes

Represents the individual food items that are ordered in a specific delivery.

- **dish_id:** Unique identifier for each dish
- **delivery_id:** Unique identifier for each dish
- **dish_count**

```sql
CREATE TABLE Delivery_Dishes (
    delivery_id INT NOT NULL,
    dish_id INT NOT NULL,
    dish_count INT,
    PRIMARY KEY (delivery_id, dish_id),
    FOREIGN KEY (delivery_id) REFERENCES Delivery(delivery_id),
    FOREIGN KEY (dish_id) REFERENCES Dish(dish_id)
);
```

```sql
INSERT INTO Delivery_Dishes (delivery_id, dish_id, dish_count) VALUES
(1, 1, 1), (1, 2, 1),  -- Delivery 1 ordered Shawarma and Falafel from Sabich Shel Oved
(2, 5, 1),             -- Delivery 2 ordered Pizza from Pizza Haim
(3, 2, 1), (3, 4, 1),  -- Delivery 3 ordered Falafel and Hummus from Falafel Golani
(4, 1, 1), (4, 2, 1),  -- Delivery 4 ordered Shawarma and Falafel from Sabich Shel Oved
(5, 5, 1),             -- Delivery 5 ordered Pizza from Pizza Haim
(6, 6, 1),             -- Delivery 6 ordered Sushi from Sushi Express
(7, 7, 1),             -- Delivery 7 ordered Pasta from Pasta La Vista
(8, 8, 1),             -- Delivery 8 ordered Grilled Chicken from Grill House
(9, 10, 1),            -- Delivery 9 ordered Tacos from Taco Bell
(10, 4, 1),            -- Delivery 10 ordered Hummus Plate from Hummus & More
(11, 1, 1), (11, 4, 1),-- Delivery 11 ordered Shawarma and Hummus Plate from Sabich She
(12, 2, 1),            -- Delivery 12 ordered Falafel from Vegan Vibes
(13, 1, 1),            -- Delivery 13 ordered Shawarma from Sabich Shel Oved
(14, 2, 1),            -- Delivery 14 ordered Falafel from Falafel Golani
(15, 5, 1),            -- Delivery 15 ordered Pizza from Pizza Haim
```

	delivery_id [PK] integer	dish_id [PK] integer	dish_count integer
1	1	1	1
2	1	2	1
3	2	5	1
4	3	2	1
5	3	4	1
6	4	1	1
7	4	2	1
8	5	5	1
9	6	6	1
10	7	7	1

Restaurant_Dishes

This ensures that each restaurant only serves certain dishes, preventing scenarios like ordering pizza from a falafel restaurant.

- **dish_id:** Unique identifier for each dish
- **restaurant_id:** Unique identifier for each dish

```sql
CREATE TABLE Restaurant_Dishes (
    restaurant_id INT,
    dish_id INT,
    PRIMARY KEY (restaurant_id, dish_id),
    FOREIGN KEY (restaurant_id) REFERENCES Restaurant(restaurant_id),
    FOREIGN KEY (dish_id) REFERENCES Dish(dish_id)
);
```

```sql
-- Create Restaurant-Dishes mapping
-- Each restaurant serves specific dishes
INSERT INTO Restaurant_Dishes (restaurant_id, dish_id) VALUES
(1, 1), -- Sabich Shel Oved serves Shawarma
(1, 2), -- Sabich Shel Oved serves Falafel
(2, 5), -- Pizza Haim serves Pizza
(3, 2), -- Falafel Golani serves Falafel
(3, 4), -- Falafel Golani serves Hummus Plate
(4, 6), -- Sushi Express serves Sushi
(5, 7), -- Pasta La Vista serves Pasta
(6, 8), -- Grill House serves Grilled Chicken
(7, 10), -- Taco Bell serves Tacos
(8, 9), -- Chicken Delight serves Vegetable Stir-Fry
(9, 4), -- Hummus & More serves Hummus Plate
(10, 2); -- Vegan Vibes serves Falafel
```

	restaurant_id [PK] integer	dish_id [PK] integer
1	1	1
2	1	2
3	2	5
4	3	2
5	3	4
6	4	6
7	5	7
8	6	8
9	7	10
10	8	9
11	9	4
12	10	2

Users Type

Clients

Restaurant Managers

System Administrators

Deliverers

Client Case

- Total Deliveries Per Client
- Total Dishes Ordered by Client
- Get Client's Delivery History
- Total Spending by Client
- Total cost of delivery by ID

Deliverer Case

- Total Deliveries Per Client
- Total Revenue per Deliverer
- Total Deliveries by Deliverer
- History of all deliveries for a deliverer by person_id

Restaurant manager Case

- Deliveries Not Yet Delivered
- Most Popular Dish
- Total Number of Deliveries per Restaurant
- Deliveries per Restaurant per Month
- Retrieve all dishes offered by a specific restaurant

System Administrators Case

- Deliveries Not Yet Delivered
- Restaurant with Maximum Revenue
- Deliverer with the Most Undelivered Deliveries
- Client with the Most Deliveries
- Total Number of Deliveries per Restaurant
- Total Revenue per Deliverer
- Deliveries per Restaurant per Month
- view deliveries details
- Get system total revenue

Queries

Deliveries Not Yet Delivered

	delivery_id integer	person_id integer	person_name character varying (255)	is_deliver boolean
1	1	6	Yael Katz	false
2	4	9	Noa Cohen	false
3	8	13	Tali Amir	false
4	10	15	Idan Eliraz	false
5	17	12	Nir Shlomo	false
6	21	6	Yael Katz	false
7	23	8	Haim Levy	false
8	27	12	Nir Shlomo	false

```sql
SELECT d.delivery_id, c.person_id, p.person_name, d.is_deliver
FROM Delivery d
JOIN Client c ON d.client_id = c.person_id
JOIN Person p ON c.person_id = p.person_id
WHERE d.is_deliver = FALSE;
```

Total Deliveries Per Client

	person_id integer	person_name character varying (255)	total_deliveries bigint
1	10	Eden Malka	5
2	8	Haim Levy	5
3	9	Noa Cohen	5
4	15	Idan Eliraz	4
5	11	Omer Yaakov	5
6	6	Yael Katz	5
7	13	Tali Amir	4
8	7	Rina Shalom	5
9	14	Or Ben-Gurion	4
10	12	Nir Shlomo	5

```sql
SELECT c.person_id, p.person_name, COUNT(d.delivery_id) AS total_deliveries
FROM Client c
JOIN Delivery d ON c.person_id = d.client_id
JOIN Person p ON c.person_id = p.person_id
GROUP BY c.person_id, p.person_name;
```

Total Dishes Ordered by Client

	client_id integer	person_name character varying (255)	total_dishes_ordered bigint
1	11	Omer Yaakov	6
2	12	Nir Shlomo	5
3	10	Eden Malka	5
4	8	Haim Levy	11
5	6	Yael Katz	7
6	14	Or Ben-Gurion	4
7	9	Noa Cohen	7
8	13	Tali Amir	6
9	15	Idan Eliraz	4
10	7	Rina Shalom	5

```sql
SELECT c.person_id AS client_id, p.person_name, SUM(dd.dish_count) AS total_dishes_ordered
FROM Client c
JOIN Person p ON c.person_id = p.person_id
JOIN Delivery d ON c.person_id = d.client_id
JOIN Delivery_Dishes dd ON d.delivery_id = dd.delivery_id
GROUP BY c.person_id, p.person_name;
```

Restaurant with Maximum Revenue

	restaurant_id [PK] integer	restaurant_name character varying (255)	total_revenue numeric
1	2	Pizza Haim	485.00

```sql
SELECT r.restaurant_id, r.restaurant_name, SUM(dd.dish_count * di.dish_cost) AS total_revenue
FROM Restaurant r
JOIN Delivery d ON r.restaurant_id = d.restaurant_id
JOIN Delivery_Dishes dd ON d.delivery_id = dd.delivery_id
JOIN Dish di ON dd.dish_id = di.dish_id
GROUP BY r.restaurant_id, r.restaurant_name
ORDER BY total_revenue DESC
LIMIT 1;
```

Deliverer with the Most Undelivered Deliveries

	deliverer_id integer	person_name character varying (255)	count bigint
1	7	Rina Shalom	2

```sql
SELECT del.deliverer_id, p.person_name, COUNT(*)
FROM Delivery del
JOIN Deliverer d ON del.deliverer_id = d.person_id
JOIN Person p ON d.person_id = p.person_id
WHERE del.is_deliver = FALSE
GROUP BY del.deliverer_id, p.person_name
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Most Popular Dish

	dish_id [PK] integer	dish_name character varying (255)	total_orders bigint
1	2	Falafel	14

```sql
SELECT di.dish_id, di.dish_name, SUM(dd.dish_count) AS total_orders
FROM Dish di
JOIN Delivery_Dishes dd ON di.dish_id = dd.dish_id
GROUP BY di.dish_id, di.dish_name
ORDER BY total_orders DESC
LIMIT 1;
```

Client with the Most Deliveries

	person_name character varying (255)	total_deliveries bigint
1	Omer Yaakov	5

```sql
SELECT p.person_name, COUNT(del.delivery_id) AS total_deliveries
FROM Delivery del
JOIN Client c ON del.client_id = c.person_id
JOIN Person p ON c.person_id = p.person_id
GROUP BY p.person_name
ORDER BY total_deliveries DESC
LIMIT 1;
```

Total Number of Deliveries per Restaurant

	restaurant_name character varying (255)	total_deliveries bigint
1	Sushi Express	6
2	Pizza Haim	5
3	Falafel Golani	5
4	Sabich Shel Oved	5
5	Pasta La Vista	5
6	Chicken Delight	4
7	Taco Bell	4
8	Vegan Vibes	4
9	Hummus & More	4
10	Grill House	4

```sql
SELECT r.restaurant_name, COUNT(del.delivery_id) AS total_deliveries
FROM Delivery del
JOIN Restaurant r ON del.restaurant_id = r.restaurant_id
GROUP BY r.restaurant_name
ORDER BY total_deliveries DESC;
```

Total Revenue per Deliverer

	deliverer_id integer	person_name character varying (255)	total_revenue numeric
1	10	Eden Malka	240.00
2	8	Haim Levy	395.00
3	5	Avi Harush	270.00
4	6	Yael Katz	325.00
5	1	Moshe Levi	280.00
6	9	Noa Cohen	190.00
7	4	Eli Ben-Ari	260.00
8	2	David Cohen	260.00
9	3	Shlomi Peretz	395.00
10	7	Rina Shalom	235.00

```sql
SELECT del.person_id AS deliverer_id, p.person_name, SUM(dd.dish_count * di.dish_cost) AS total_revenue
FROM Deliverer del
JOIN Person p ON del.person_id = p.person_id
JOIN Delivery d ON del.person_id = d.deliverer_id
JOIN Delivery_Dishes dd ON d.delivery_id = dd.delivery_id
JOIN Dish di ON dd.dish_id = di.dish_id
GROUP BY del.person_id, p.person_name;
```

Deliveries per Restaurant per Month

	restaurant_name character varying (255)	t_month integer	total_deliveries bigint
1	Chicken Delight	8	4
2	Falafel Golani	9	1
3	Falafel Golani	11	4
4	Grill House	10	4
5	Hummus & More	7	4
6	Pasta La Vista	11	5
7	Pizza Haim	10	4
8	Pizza Haim	11	1
9	Sabich Shel Oved	9	4
10	Sabich Shel Oved	10	1
11	Sushi Express	11	1
12	Sushi Express	12	5
13	Taco Bell	9	4
14	Vegan Vibes	8	4

```sql
SELECT r.restaurant_name, t.t_month, COUNT(del.delivery_id) AS total_deliveries
FROM Delivery del
JOIN Restaurant r ON del.restaurant_id = r.restaurant_id
JOIN Time t ON del.time_id = t.time_id
GROUP BY r.restaurant_name, t.t_month
ORDER BY r.restaurant_name, t.t_month;
```

Get Client's Delivery History

```
SELECT * FROM Get_Client_Delivery_History(8);
```

	delivery_id integer	restaurant_name character varying	delivery_status boolean
1	43	Sabich Shel Oved	true
2	33	Sabich Shel Oved	true
3	23	Sabich Shel Oved	false
4	13	Sabich Shel Oved	true
5	3	Falafel Golani	false

```
CREATE OR REPLACE FUNCTION Get_Client_Delivery_History(p_client_id INT)
RETURNS TABLE(delivery_id INT, restaurant_name VARCHAR, delivery_status BOOLEAN)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT d.delivery_id, r.restaurant_name, d.is_deliver
    FROM Delivery d
    JOIN Restaurant r ON d.restaurant_id = r.restaurant_id
    WHERE d.client_id = p_client_id;
END;
$$;
```

Total Spending by Client

```
SELECT total_spending_by_client(8);
```

	total_spending_by_client numeric
1	2850.00

```
CREATE OR REPLACE FUNCTION total_spending_by_client(p_client_id INTEGER)
RETURNS NUMERIC AS $$
DECLARE
    total_spending NUMERIC;
BEGIN
    SELECT SUM(dish_cost * dd.dish_count) INTO total_spending
    FROM Delivery d
    JOIN Delivery_Dishes dd ON d.delivery_id = dd.delivery_id
    JOIN Dish ds ON dd.dish_id = ds.dish_id
    WHERE d.client_id = client_id;

    RETURN COALESCE(total_spending, 0); -- Return 0 if no spending is found
END;
$$ LANGUAGE plpgsql;
```

Total Deliveries by Deliverer

```
SELECT Get_Total_Deliveries_By_Deliverer(3);
```

get_total_deliveries_by_deliverer integer
6

```
CREATE OR REPLACE FUNCTION Get_Total_Deliveries_By_Deliverer(p_deliverer_id INT)
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
    total_deliveries INT;
BEGIN
    SELECT COUNT(*)
    INTO total_deliveries
    FROM Delivery
    WHERE deliverer_id = p_deliverer_id;

    RETURN total_deliveries;
END;
$$;
```

Retrieve all dishes offered by a specific restaurant

```
SELECT * FROM Get_Restaurant_Dishes(1);
```

	dish_name character varying	dish_cost numeric
1	Shawarma	45.00
2	Falafel	30.00

```
CREATE OR REPLACE FUNCTION Get_Restaurant_Dishes(p_restaurant_id INT)
RETURNS TABLE(dish_name VARCHAR, dish_cost NUMERIC)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT d.dish_name, d.dish_cost
    FROM Restaurant_Dishes rd
    JOIN Dish d ON rd.dish_id = d.dish_id
    WHERE rd.restaurant_id = p_restaurant_id;
END;
$$;
```

Get system total revenue

```
SELECT get_system_total_revenue();
```

	get_system_total_revenue numeric
1	2850.00

```
CREATE OR REPLACE FUNCTION get_system_total_revenue()
RETURNS NUMERIC AS $$
DECLARE
    total_revenue NUMERIC;
BEGIN
    SELECT SUM(dish_cost * dd.dish_count) INTO total_revenue
    FROM Delivery d
    JOIN Delivery_Dishes dd ON d.delivery_id = dd.delivery_id
    JOIN Dish ds ON dd.dish_id = ds.dish_id;

    RETURN COALESCE(total_revenue, 0); -- Return 0 if no revenue is found
END;
$$ LANGUAGE plpgsql;
```

History of all deliveries for a deliverer by person_id

```
SELECT * FROM get_deliverer_delivery_history(3);
```

delivery_id integer	client_id integer	restaurant_name character varying	is_deliver boolean	delivery_date date	delivery_time time without time zone
13	8	Sabich Shel Oved	true	2023-09-20	12:15:00
3	8	Falafel Golani	false	2023-09-20	12:15:00
48	6	Sushi Express	true	2023-11-01	13:25:00

```
CREATE OR REPLACE FUNCTION get_deliverer_delivery_history(deliverer_person_id INT)
RETURNS TABLE (
    delivery_id INT,
    client_id INT,
    restaurant_name VARCHAR,
    is_deliver BOOLEAN,
    delivery_date DATE,
    delivery_time TIME
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        d.delivery_id,
        d.client_id,
        r.restaurant_name,
        d.is_deliver,
        TO_DATE(CONCAT(t.t_day, '-', t.t_month, '-', t.t_year), 'DD-MM-YYYY') AS delivery_date,
        MAKE_TIME(t.t_hour, t.t_minute, 0) AS delivery_time
    FROM delivery d
    JOIN deliverer del ON d.deliverer_id = del.person_id
    JOIN restaurant r ON d.restaurant_id = r.restaurant_id
    JOIN time t ON d.time_id = t.time_id
    WHERE del.person_id = deliverer_person_id;
END;
$$ LANGUAGE plpgsql;
```

Total cost of delivery by ID

```sql
SELECT get_total_delivery_cost(49);
```

	get_total_delivery_cost numeric
1	180.00

```sql
CREATE OR REPLACE FUNCTION get_total_delivery_cost(delivery_id_input INT)
RETURNS NUMERIC AS $$
DECLARE
    total_cost NUMERIC := 0;
BEGIN
    -- Calculate total cost by summing up the cost of all dishes in the delivery considering their count
    SELECT SUM(d.dish_cost * dd.dish_count)
    INTO total_cost
    FROM delivery_dishes dd
    JOIN dish d ON dd.dish_id = d.dish_id
    WHERE dd.delivery_id = delivery_id_input;

    -- Return the total cost
    RETURN COALESCE(total_cost, 0);  -- Returns 0 if no dishes are found for the delivery
END;
$$ LANGUAGE plpgsql;
```

INSERT/UPDATE/DELETE Procedures (examples)

Insert deliverer

```
CALL Insert_New_Deliverer(11, 'Senior', 2500.00);
```

	person_id [PK] integer	deliverer_rank character varying (50)	salary numeric (10,2)
10	10	Intermediate	14500.00
11	11	Senior	2500.00

```
CREATE OR REPLACE PROCEDURE Insert_New_Deliverer(p_person_id INT, p_rank VARCHAR(50), p_salary NUMERIC(10, 2))
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO Deliverer (person_id, deliverer_rank, salary)
    VALUES (p_person_id, p_rank, p_salary);
END;
$$;
```

Update deliverer salary

```
CALL Update_Deliverer_Salary(11, 3000.00);
```

	person_id [PK] integer	deliverer_rank character varying (50)	salary numeric (10,2)
10	10	Intermediate	14500.00
11	11	Senior	3000.00

```
CREATE OR REPLACE PROCEDURE Update_Deliverer_Salary(p_person_id INT, p_new_salary NUMERIC(10, 2))
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Deliverer
    SET salary = p_new_salary
    WHERE person_id = p_person_id;
END;
$$;
```

Delete deliverer

```
CALL Delete_Deliverer(11);
```

	person_id [PK] integer	deliverer_rank character varying (50)	salary numeric (10,2)
9	9	Junior	11000.00
10	10	Intermediate	14500.00

```
CREATE OR REPLACE PROCEDURE Delete_Deliverer(p_person_id INT)
LANGUAGE plpgsql
AS $$
BEGIN
    DELETE FROM Deliverer WHERE person_id = p_person_id;
END;
$$;
```

Update Delivery Status

	delivery_id [PK] integer	client_id integer	deliverer_id integer	restaurant_id integer	time_id integer	is_deliver boolean
1	4	9	4	1	4	false

delivery_id [PK] integer	client_id integer	deliverer_id integer	restaurant_id integer	time_id integer	is_deliver boolean
4	9	4	1	4	true

```
CALL Update_Delivery_Status(4, TRUE);
```

```
CREATE OR REPLACE PROCEDURE Update_Delivery_Status(p_delivery_id INT, p_is_deliver BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
    UPDATE Delivery
    SET is_deliver = p_is_deliver
    WHERE delivery_id = p_delivery_id;

    RAISE NOTICE 'Delivery ID % has been updated to status %', p_delivery_id, p_is_deliver
END;
$$;
```

Add or update delivery

```sql
CALL add_or_update_delivery(
    49,              -- p_delivery_id
    8,            -- p_client_id
    3,            -- p_deliverer_id
    2,            -- p_restaurant_id
    1,            -- p_time_id
    TRUE,            -- p_is_deliver
    ARRAY[1, 2],    -- p_dish_ids
    ARRAY[2, 3]     -- p_dish_counts
);
```

	delivery_id [PK] integer	dish_id [PK] integer	dish_count integer
56	49	1	2
57	49	2	3

```sql
CREATE OR REPLACE PROCEDURE add_or_update_delivery(
    p_delivery_id INTEGER,
    p_client_id INTEGER,
    p_deliverer_id INTEGER,
    p_restaurant_id INTEGER,
    p_time_id INTEGER,
    p_is_deliver BOOLEAN,
    p_dish_ids INTEGER[],       -- Array of dish IDs
    p_dish_counts INTEGER[]     -- Array of corresponding dish counts
)
LANGUAGE plpgsql
AS $$
DECLARE
    v_dish_id INTEGER;          -- Loop variable for dish IDs
    v_count INTEGER;            -- Loop variable for dish counts
BEGIN
    -- Check if the delivery_id already exists
    IF EXISTS (SELECT 1 FROM Delivery WHERE delivery_id = p_delivery_id) THEN
        -- Update existing delivery
        UPDATE Delivery
        SET client_id = p_client_id,
            deliverer_id = p_deliverer_id,
            restaurant_id = p_restaurant_id,
            time_id = p_time_id,
            is_deliver = p_is_deliver
        WHERE delivery_id = p_delivery_id;
    ELSE
        -- Insert a new delivery
        INSERT INTO Delivery (delivery_id, client_id, deliverer_id, restaurant_id, time_id, is_deliver)
        VALUES (p_delivery_id, p_client_id, p_deliverer_id, p_restaurant_id, p_time_id, p_is_deliver);
    END IF;

    -- Add or update the related dishes in the Delivery_Dishes table
    FOR i IN 1 .. array_length(p_dish_ids, 1) LOOP
        v_dish_id := p_dish_ids[i];             -- Get current dish ID
        v_count := p_dish_counts[i];            -- Get corresponding dish count

        INSERT INTO Delivery_Dishes (delivery_id, dish_id, dish_count)
        VALUES (p_delivery_id, v_dish_id, v_count) -- Use the loop variable
        ON CONFLICT (delivery_id, dish_id)
        DO UPDATE SET dish_count = Delivery_Dishes.dish_count + EXCLUDED.dish_count; -- Update count if exists
    END LOOP;
END $$;
```

Delete delivery

```
-- CALL delete_delivery(1);
-- CALL delete_delivery(2);

select * from delivery
```

```
CREATE OR REPLACE PROCEDURE delete_delivery(
    p_delivery_id INTEGER  -- Delivery ID to be deleted
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Check if the delivery exists
    IF EXISTS (SELECT 1 FROM Delivery WHERE delivery_id = p_delivery_id) THEN
        -- First, delete related entries from Delivery_Dishes
        DELETE FROM Delivery_Dishes
        WHERE delivery_id = p_delivery_id;

        -- Then, delete the delivery itself
        DELETE FROM Delivery
        WHERE delivery_id = p_delivery_id;

        RAISE NOTICE 'Delivery with ID % has been deleted.', p_delivery_id;
    ELSE
        -- If the delivery doesn't exist, raise an error
        RAISE NOTICE 'Delivery with ID % does not exist.', p_delivery_id;
    END IF;
END;
$$;
```

	delivery_id [PK] integer	client_id integer	deliverer_id integer	restaurant_id integer	time_id integer	is_deliver boolean
1	3	8	3	3	3	true
2	4	9	4	1	4	false

View

view deliveries details

```sql
CREATE OR REPLACE VIEW detailed_delivery_info AS
SELECT
    del.delivery_id,
    p.person_name AS client_name,
    r.restaurant_name,
    SUM(dsh.dish_cost) AS total_delivery_cost,
    del.is_deliver AS delivery_status,
    delr.deliverer_rank
FROM Delivery del
JOIN Client c ON del.client_id = c.person_id
JOIN Person p ON c.person_id = p.person_id
JOIN Restaurant r ON del.restaurant_id = r.restaurant_id
JOIN Delivery_Dishes dd ON del.delivery_id = dd.delivery_id
JOIN Dish dsh ON dd.dish_id = dsh.dish_id
JOIN Deliverer delr ON del.deliverer_id = delr.person_id
GROUP BY del.delivery_id, p.person_name, r.restaurant_name, del.is_deliver, delr.deliverer_rank;
```

```sql
SELECT * FROM detailed_delivery_info
ORDER BY total_delivery_cost DESC;
```

	delivery_id integer	client_name character varying (255)	restaurant_name character varying (255)	total_delivery_cost numeric	delivery_status boolean	deliverer_rank character varying (50)
1	48	Yael Katz	Sushi Express	125.00	true	Junior
2	20	Idan Eliraz	Chicken Delight	75.00	true	Intermediate
3	6	Omer Yaakov	Sushi Express	75.00	true	Junior
4	44	Noa Cohen	Pizza Haim	75.00	true	Senior
5	28	Tali Amir	Grill House	75.00	true	Intermediate

Triggers

```sql
CREATE TABLE Client_Log (
    person_id INT,
    old_phone_number VARCHAR(15),
    new_phone_number VARCHAR(15),
    command_ts TIMESTAMP,
    command VARCHAR(25)
);
```

Client_Log

- INSERT
- UPDATE
- DELETE

```sql
CREATE OR REPLACE FUNCTION Client_Log_Insert()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Client_Log (person_id, new_phone_number, command_ts, command)
    VALUES (NEW.person_id, NEW.phone_number, NOW(), 'insert');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Client_Log_Insert
AFTER INSERT ON Client
FOR EACH ROW
EXECUTE FUNCTION Client_Log_Insert();
```

```sql
CREATE OR REPLACE FUNCTION Client_Log_Update()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Client_Log (person_id, old_phone_number, new_phone_number, command_ts, command)
    VALUES (NEW.person_id, OLD.phone_number, NEW.phone_number, NOW(), 'update');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Client_Log_Update
AFTER UPDATE ON Client
FOR EACH ROW
EXECUTE FUNCTION Client_Log_Update();
```

```sql
CREATE OR REPLACE FUNCTION Client_Log_Delete()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Client_Log (person_id, old_phone_number, command_ts, command)
    VALUES (OLD.person_id, OLD.phone_number, NOW(), 'delete');
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Client_Log_Delete
AFTER DELETE ON Client
FOR EACH ROW
EXECUTE FUNCTION Client_Log_Delete();
```

```sql
CREATE TABLE Deliverer_Log (
    person_id INT,
    old_deliverer_rank VARCHAR(50),
    new_deliverer_rank VARCHAR(50),
    old_salary NUMERIC(10,2),
    new_salary NUMERIC(10,2),
    command_ts TIMESTAMP,
    command VARCHAR(25)
);
```

Deliverer_Log

- INSERT
- UPDATE
- DELETE

```sql
CREATE OR REPLACE FUNCTION Deliverer_Log_Insert()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Deliverer_Log (person_id, new_deliverer_rank, new_salary, command_ts, command)
    VALUES (NEW.person_id, NEW.deliverer_rank, NEW.salary, NOW(), 'insert');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Deliverer_Log_Insert
AFTER INSERT ON Deliverer
FOR EACH ROW
EXECUTE FUNCTION Deliverer_Log_Insert();
```

```sql
CREATE OR REPLACE FUNCTION Deliverer_Log_Update()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Deliverer_Log (person_id, old_deliverer_rank, new_deliverer_rank, old_salary, new_salary, command_ts, command)
    VALUES (NEW.person_id, OLD.deliverer_rank, NEW.deliverer_rank, OLD.salary, NEW.salary, NOW(), 'update');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Deliverer_Log_Update
AFTER UPDATE ON Deliverer
FOR EACH ROW
EXECUTE FUNCTION Deliverer_Log_Update();
```

```sql
CREATE OR REPLACE FUNCTION Deliverer_Log_Delete()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Deliverer_Log (person_id, old_deliverer_rank, old_salary, command_ts, command)
    VALUES (OLD.person_id, OLD.deliverer_rank, OLD.salary, NOW(), 'delete');
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Deliverer_Log_Delete
AFTER DELETE ON Deliverer
FOR EACH ROW
EXECUTE FUNCTION Deliverer_Log_Delete();
```

```sql
CREATE TABLE Delivery_Log (
    delivery_id INT,
    old_client_id INT,
    new_client_id INT,
    old_is_deliver BOOLEAN,
    new_is_deliver BOOLEAN,
    command_ts TIMESTAMP,
    command VARCHAR(25)
);
```

Delivery_Log

- INSERT
- UPDATE
- DELETE

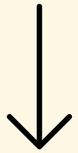
```sql
CREATE OR REPLACE FUNCTION Delivery_Log_Insert()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Delivery_Log (delivery_id, new_client_id, new_is_deliver, command_ts, command)
    VALUES (NEW.delivery_id, NEW.client_id, NEW.is_deliver, NOW(), 'insert');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Delivery_Log_Insert
AFTER INSERT ON Delivery
FOR EACH ROW
EXECUTE FUNCTION Delivery_Log_Insert();
```

```sql
CREATE OR REPLACE FUNCTION Delivery_Log_Update()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Delivery_Log (delivery_id, old_client_id, new_client_id, old_is_deliver, new_is_deliver, command_ts, command)
    VALUES (NEW.delivery_id, OLD.client_id, NEW.client_id, OLD.is_deliver, NEW.is_deliver, NOW(), 'update');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Delivery_Log_Update
AFTER UPDATE ON Delivery
FOR EACH ROW
EXECUTE FUNCTION Delivery_Log_Update();
```

```sql
CREATE OR REPLACE FUNCTION Delivery_Log_Delete()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO Delivery_Log (delivery_id, old_client_id, old_is_deliver, command_ts, command)
    VALUES (OLD.delivery_id, OLD.client_id, OLD.is_deliver, NOW(), 'delete');
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_Delivery_Log_Delete
AFTER DELETE ON Delivery
FOR EACH ROW
EXECUTE FUNCTION Delivery_Log_Delete();
```

Enforce minumum dish cost

```
UPDATE Dish SET dish_cost = 0.75 WHERE dish_id = 1;
```



```
ERROR:  Dish cost must be at least $1.00.
```

```
CREATE OR REPLACE FUNCTION enforce_minimum_dish_cost()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.dish_cost < 1.00 THEN  -- Minimum cost is set to $1.00
        RAISE EXCEPTION 'Dish cost must be at least $1.00.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_enforce_minimum_dish_cost
BEFORE INSERT OR UPDATE ON Dish
FOR EACH ROW EXECUTE FUNCTION enforce_minimum_dish_cost();
```

Prevent duplicate client phone number

```
UPDATE Client
SET phone_number = '054-4455666'
WHERE person_id = 8;
```

```
CREATE OR REPLACE FUNCTION prevent_duplicate_client_phone_number()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM Client WHERE phone_number = NEW.phone_number) THEN
        RAISE EXCEPTION 'Phone number "%s" already exists for another client.', NEW.phone_number;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_prevent_duplicate_client_phone_number
BEFORE INSERT OR UPDATE ON Client
FOR EACH ROW EXECUTE FUNCTION prevent_duplicate_client_phone_number();
```

```
ERROR:  Phone number "054-4455666s" already exists for another client.
```

References

- Course presentations
- Class practice
- YOUTUBE videos
- CHAT GPT
- Geeks for geeks website
- Stack overFlow website

Thank you for listening

General Triggers

Prevent a restaurant from being deleted if there are associated deliveries

```
DELETE FROM Restaurant WHERE restaurant_id = 1;
```

```
ERROR:  Cannot delete restaurant with existing deliveries
CONTEXT:  PL/pgSQL function prevent_restaurant_deletion() line 4 at RAISE

SQL state: P0001
```

```
CREATE OR REPLACE FUNCTION prevent_restaurant_deletion()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM Delivery WHERE restaurant_id = OLD.restaurant_id) THEN
        RAISE EXCEPTION 'Cannot delete restaurant with existing deliveries';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_prevent_restaurant_deletion
BEFORE DELETE ON Restaurant
FOR EACH ROW EXECUTE FUNCTION prevent_restaurant_deletion();
```

trigger to delete all deliveries associated with a client when the client is deleted

```
-- Function to delete deliveries when a client is deleted
CREATE OR REPLACE FUNCTION delete_client_deliveries()
RETURNS TRIGGER AS $$
BEGIN
    -- Delete all deliveries associated with the client being deleted
    DELETE FROM Delivery WHERE client_id = OLD.person_id;
    RETURN OLD;  -- Return the old row (the deleted client)
END;
$$ LANGUAGE plpgsql;

-- Trigger that calls the function before a client is deleted
CREATE TRIGGER delete_client_trigger
AFTER DELETE ON Client  -- When a client is deleted
FOR EACH ROW EXECUTE FUNCTION delete_client_deliveries();
```