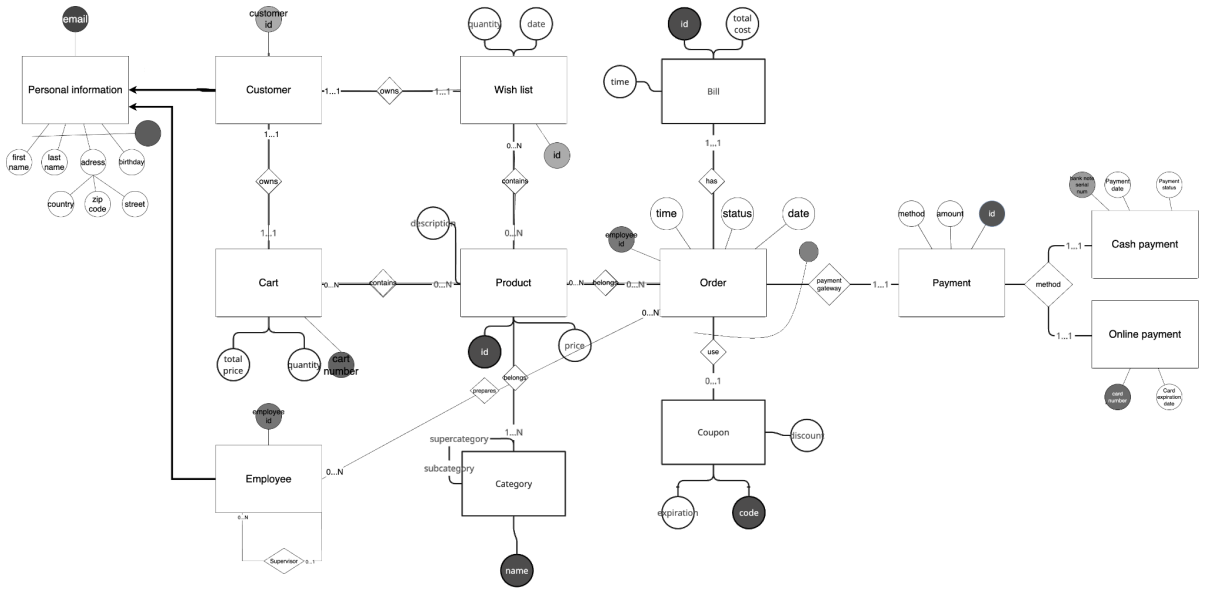
Conceptual model



Relational model

01

Personal information(first name, last name, address, birthday, email)

Address(personal information, country, zip code, street)

02

Employee(employee id, email)

FK: (email) ⊆ Personal information(email)

Supervisor(employee, supervisor)

FK: (employee) ⊆ Employee(employee id) FK: (supervisor) ⊆ Employee(employee id)

Prepares(employee id, employee id)

FK: (employee id) ⊆ Employee(employee id) FK: (employee id) ⊆ Order(employee id)

Customer(customer id, email)

FK: (email) ⊆ Personal information(email)

Cart(total price, quantity, cart number) Owns(customer id, cart number)

FK: (customer id) ⊆ Customer(customer id)

FK: (cart number) ⊆ Cart(cart number) Contains(product id, cart number)

FK: (product id) ⊆ Product(product id) FK: (cart number) ⊆ Cart(cart number)

Wish list(quantity, date, wish list id) Owns(customer id, wish list id)

FK: (customer id) ⊆ Customer(customer id)

FK: (wish list id) ⊆ Wish list(wish list id) Contains(product id, wishlist id)

FK: (product id) ⊆ Product(product id) FK: (wish list id) ⊆ Wish list(wish list id)

Product(description, id, price) Contains(product id, wishlist id)

FK: (product id) ⊆ Product(product id)

FK: (wish list id) ⊆ Wish list(wish list id) Contains(product id, cart number)

FK: (product id) ⊆ Product(product id)

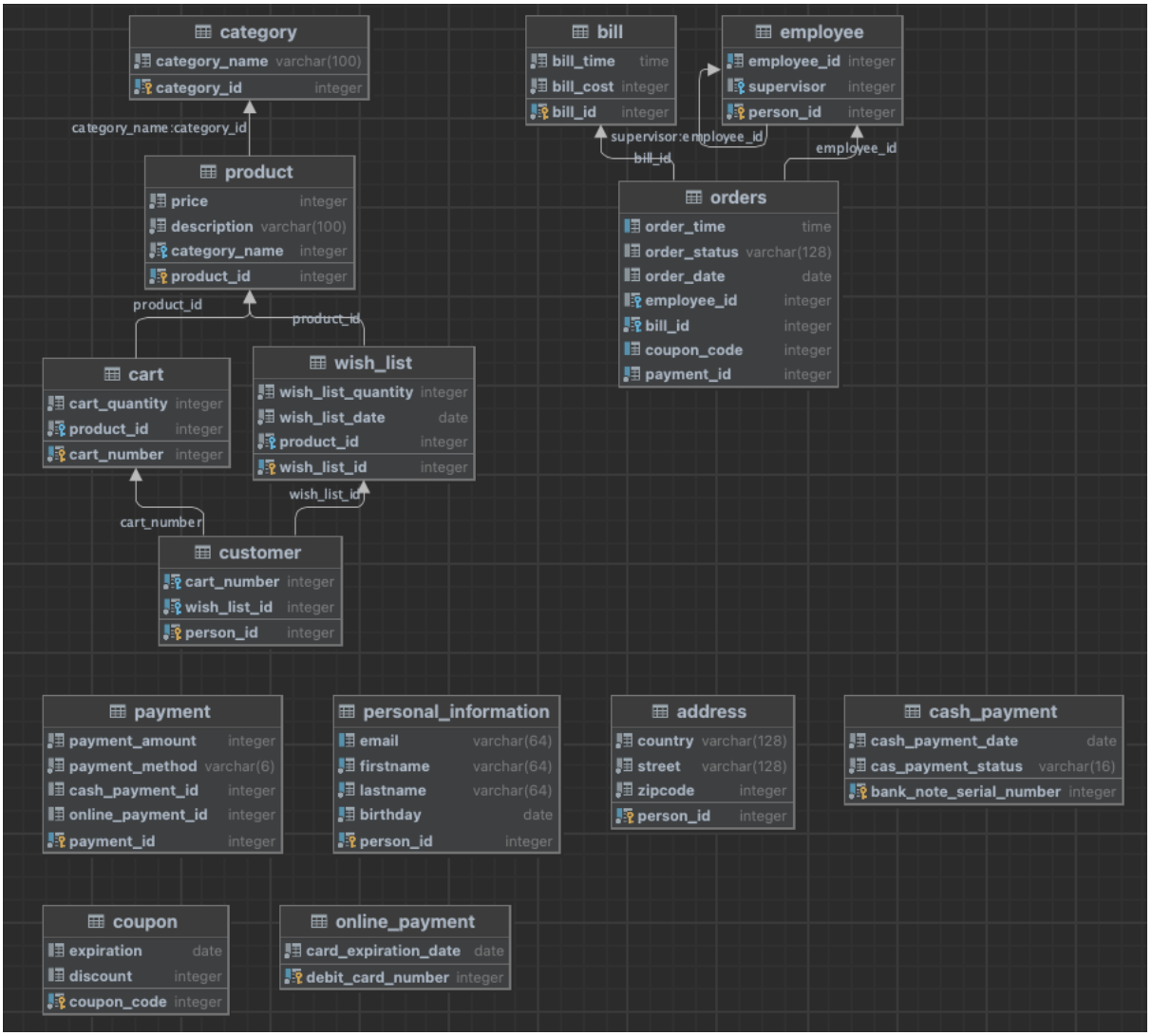
FK: (cart number) ⊆ Cart(cart number) Belongs(product id, name)

FK: (product id) ⊆ Product(product id)

FK: (name) ⊆ Category(name) Belongs(product id, employee id)

FK: (product id) ⊆ Product(product id) FK: (employee id) ⊆ Order(employee id)

Category(name)
Subcategory(category, Subcategory) FK: (category) ⊆ Category(name) FK: (subcategory) ⊆ Category(name)
Order(employee id, time, status, date, payment, coupon) Has(bill id, employee id)
FK: (employee id) ⊆ Order(employee id)
FK: (bill id) ⊆ Bill(bill id) Use(code, employee id)
FK: (employee id) ⊆ Order(employee id) FK: (code) ⊆ Coupon(code)
Prepares(employee id, employee id)
FK: (employee id) ⊆ Employee(employee id) FK: (employee id) ⊆ Order(employee id)
Payment gateway(payment id, employee id)
FK: (employee id) ⊆ Order(employee id) FK: (code) ⊆ Coupon(code)
Bill(bill id, time, total cost) Has(bill id, employee id)
FK: (employee id) ⊆ Order(employee id) FK: (bill id) ⊆ Bill(bill id)
Coupon(expiration, code, discount) Use(code, employee id)
FK: (employee id) ⊆ Order(employee id) FK: (code) ⊆ Coupon(code)
Payment(payment id, method, amount) Method(payment id, bank note serial number)
FK: (payment id) ⊆ Payment(payment id)
FK: (bank note serial number) ⊆ Cash payment(bank note serial number)
Method(payment id, card number)
FK: (payment id) ⊆ Payment(payment id)
FK: (card number) ⊆ Card payment(card number)



Data to my SQL database was generated using the **python module Faker**.
Here is how 32.000 data records were generated:

```python
from faker import Faker
import random

fake = Faker('cs_CZ')

with open('personalinformation.txt', 'w') as f:
    used_emails = set()
    used_names = set()
    for i in range(32000):

        while True:
            email = fake.email()
            if email not in used_emails:
                used_emails.add(email)
                break

        gender = random.choice(['male', 'female'])
        if gender == 'male':
            first_name = fake.first_name_male()
            last_name = fake.last_name_male()
        else:
            first_name = fake.first_name_female()
            last_name = fake.last_name_female()
        birthday = fake.date_of_birth(minimum_age=18, maximum_age=65).strftime('%Y-%m-%d')

        if (first_name, last_name, birthday) in used_names:
            continue
        used_names.add((first_name, last_name, birthday))

        f.write(f"INSERT INTO Personal_information (email, firstname, lastname, birthday) VALUES ('{email}', '{first_name}', '{last_name}',
```

It randomly generates male or female names. However, because the email, also name with birthday are both UNIQUE, it has to check if it was already used. Then I copypasted all the code from .txt file to sql console.

Here is how the payment table was created. Saving the bank serial number and card number from online_payment and cash_payment table was needed, so it can be used in the payment table. If the online payment was selected the cash payment was set to NULL and viceversa.

```python
serialnum_list = list(serialnum)
card_list = list(card)

with open('payment.txt', 'w') as f:
    for i in range(500):
        amount = random.randint(10, 100)
        payment_id = i + 1
        payment_method = random.choice(['cash', 'online'])

        if payment_method == 'cash':
            payment_method_id = random.choice(serialnum_list)
            serialnum_list.remove(payment_method_id)
            online_payment_id = 'NULL'
        else:
            payment_method_id = 'NULL'
            online_payment_id = random.choice(card_list)
            card_list.remove(online_payment_id)

        f.write(f"INSERT INTO payment (payment_amount, payment_id, payment_method, cash_payment_id, online_payment_id) VALUES ({amount}, {pay
```

In the employee table first 50 employees were selected as supervisors and they were randomly assigned to each employee.

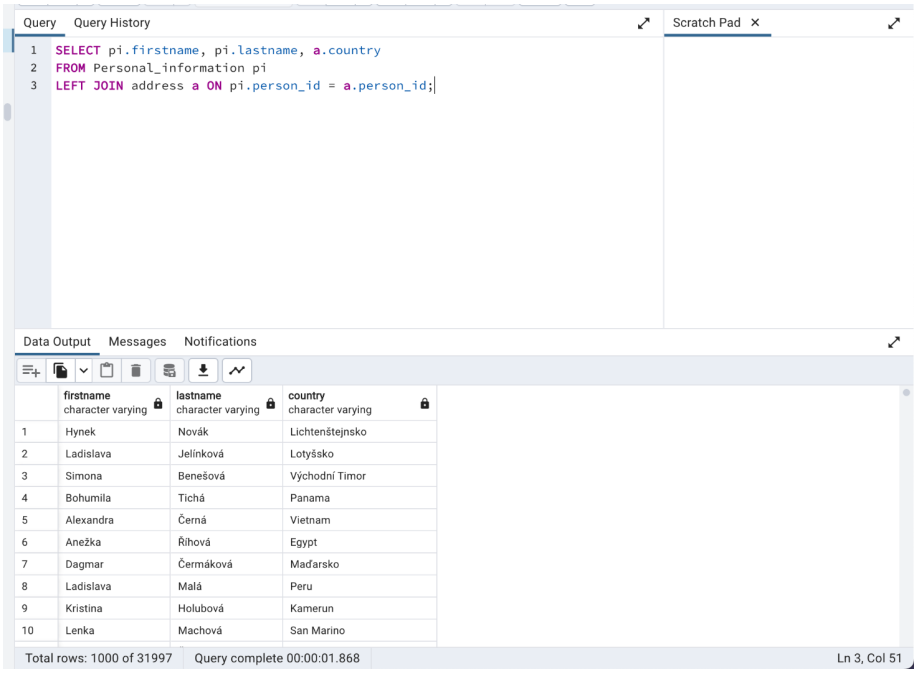
```python
# GENERATE EMPLOYEE
with open('employee.txt', 'w') as f:
    for i in range(500):
        personid = i + 1
        employeeid = i + 1001

        if i < 50:
            supervisor = "NULL"
        else:
            supervisor = str(random.randint(1000, 1050))

        f.write(f"INSERT INTO employee (person_id, employee_id, supervisor) VALUES ('{personid}', '{employeeid}', {supervisor});\n")
```

SQL queries to retrieve data from a database

1.) outer join of tables



It returns a table with three columns - firstname, lastname, and country.

2.) inner join of tables

Query | Query History | Scratch Pad

```
SELECT p.product_id, p.description, c.category_name
FROM product p
INNER JOIN category c ON p.category_name = c.category_id;
```

Data Output | Messages | Notifications

	product_id integer	description character varying	category_name character varying
1	1	wonderful	Fishing and Hunting
2	2	determined	Pet Supplies
3	3	radiant	Pet Supplies
4	4	wonderful	Books and Stationery
5	5	magnificent	Tools and Home Improvement
6	6	passionate	Toys
7	7	brave	Camping and Hiking
8	8	amazing	Toys
9	9	polite	Cycling
10	10	reliable	Tools and Home Improvement

Total rows: 500 of 500 | Query complete 00:00:00.518 | Ln 3, Col 58

It selects the product_id, description, and category_name columns from the product and category tables and JOIN both of them together.

3.) data condition

Query | Query History | Scratch Pad

```
SELECT pi.firstname, pi.lastname, pi.birthday
FROM Personal_information pi
WHERE pi.birthday BETWEEN '2003-01-01' AND '2003-12-31';
```

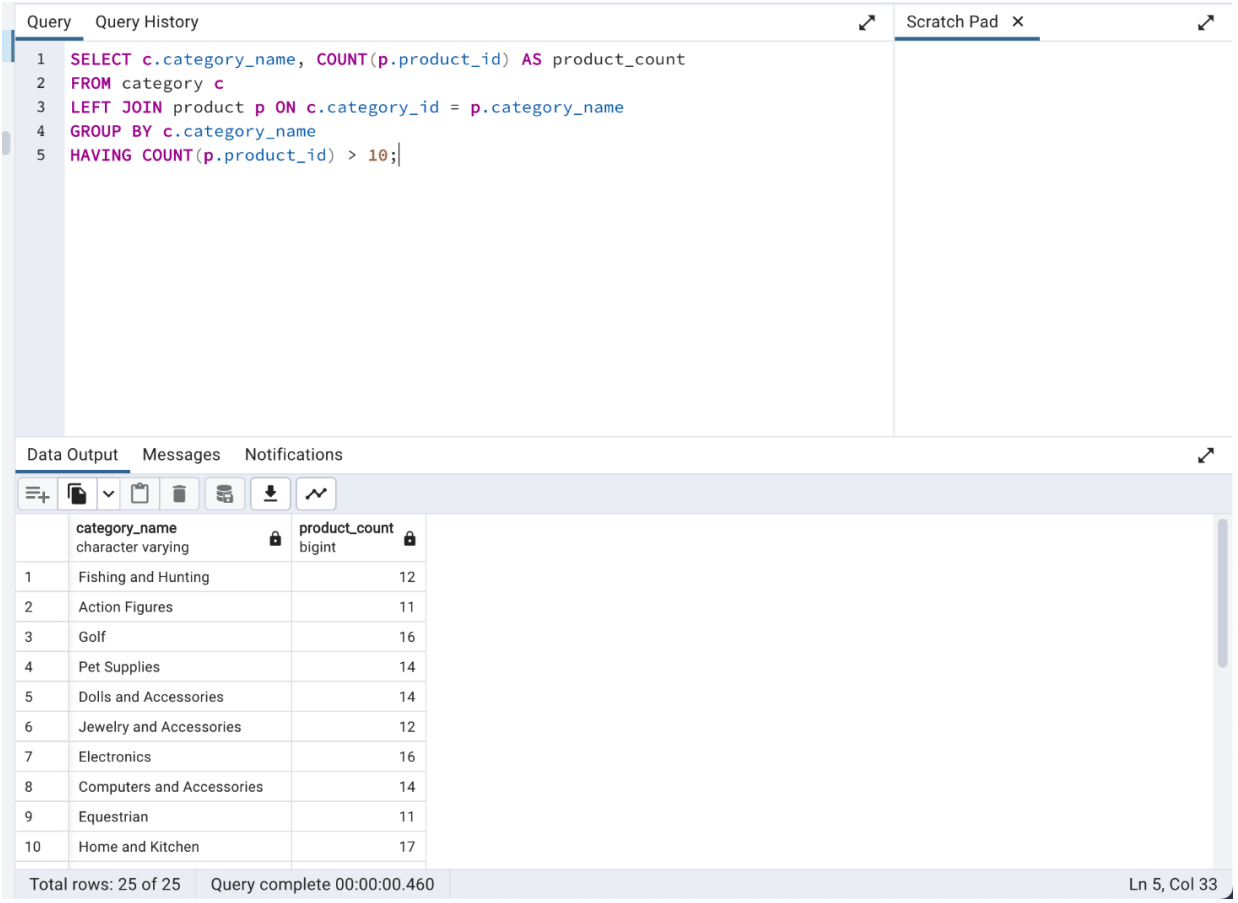
Data Output | Messages | Notifications

	firstname character varying	lastname character varying	birthday date
1	Marie	Doležalová	2003-05-05
2	Milada	Kolářová	2003-02-08
3	Eduard	Zeman	2003-12-21
4	Lenka	Dvořáková	2003-01-13
5	Hynek	Jelínek	2003-03-19
6	Ivo	Kopecký	2003-03-20
7	Jarmila	Kučerová	2003-12-13
8	Miroslava	Křížová	2003-05-06
9	Natálie	Bláhová	2003-11-15
10	Marcel	Dvořák	2003-03-01

Total rows: 675 of 675 | Query complete 00:00:00.495 | Ln 3, Col 57

This returns only people who were born between dates 1.1.2003 and 31.12.2003

4.) aggregation and the condition on the value of the aggregation function



It selects the category name from the category table and counts the number of products associated with each category. It uses a LEFT JOIN to combine the category table with the product table on the category_id and category_name. The HAVING is used to filter the result set by the product count. It only selects the rows where the product count is greater than 10.

5.) sorting and pagination

Query | Query History | Scratch Pad

```
SELECT pi.firstname, pi.lastname, pi.birthday
FROM Personal_information pi
ORDER BY pi.birthday DESC
OFFSET 10
LIMIT 5;
```

Data Output | Messages | Notifications

	firstname character varying	lastname character varying	birthday date
1	Ludvík	Kadlec	2005-04-19
2	Emilie	Kratochvílová	2005-04-19
3	Zdeněk	Štěpánek	2005-04-19
4	Ladislav	Kadlec	2005-04-18
5	Martina	Štěpánková	2005-04-17

Total rows: 5 of 5 | Query complete 00:00:00.345 | Ln 5, Col 9

This query retrieves the first 5 rows from the Personal_information table, starting from the 11th row, ordered by the birthday in descending order.

6.) set operations

Query | Query History | Scratch Pad

```
SELECT pi.firstname, pi.lastname, pi.birthday
FROM Personal_information pi
WHERE pi.person_id IN (
  SELECT c.person_id
  FROM customer c
  WHERE c.cart_number IS NOT NULL
  INTERSECT
  SELECT w.person_id
  FROM customer w
  WHERE w.wish_list_id IS NOT NULL
);
```

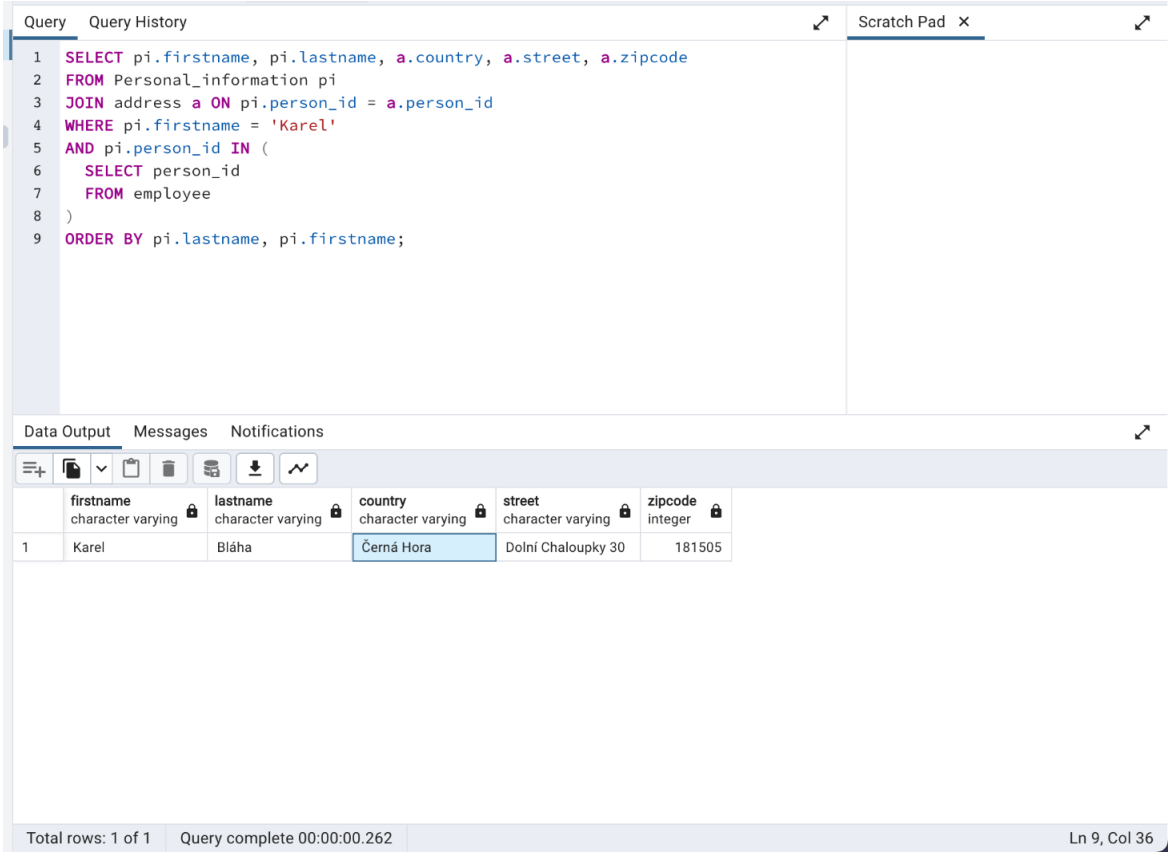
Data Output | Messages | Notifications

	firstname character varying	lastname character varying	birthday date
1	Hynek	Novák	1965-05-10
2	Ladislava	Jelínková	1966-05-12
3	Simona	Benešová	1996-03-30
4	Bohumila	Tichá	1974-08-03
5	Alexandra	Černá	1957-08-21
6	Anežka	Říhová	1993-12-15
7	Dagmar	Čermáková	2004-06-21
8	Ladislava	Malá	1989-08-17
9	Kristina	Holubová	1985-06-09
10	Lenka	Machová	1973-03-02

Total rows: 500 of 500 | Query complete 00:00:00.403 | Ln 11, Col 3

It selects personal information (firstname, lastname, and birthday) for all customers whose person_id is in the set of customer IDs that have both a **cart** and a **wish list**.

7.) nested SELECT



This query selects the first name, last name, country, street, and zipcode of all **employees** named "Karel" along with their corresponding address information. The results are ordered by last name and then first name in ascending order.

CP-4

Transaction

```
CREATE FUNCTION add_to_cart(prod_id INT, car_id INT, num INT)
RETURNS BOOLEAN
AS $$
DECLARE
    count1 INT;
    count2 INT;
BEGIN
    count1 := (SELECT product_count FROM product WHERE product_id = prod_id);
    count2 := (SELECT cart_quantity FROM cart WHERE cart_number = car_id);
    IF (count1 <= 0) OR (num > count1) THEN RETURN false; END IF;
    UPDATE product SET product_count = product_count - num WHERE product_id = prod_id;
    UPDATE cart SET cart_quantity = cart_quantity + num WHERE cart_number = car_id;
RETURN true;
END;
$$
language plpgsql;
```

This function adds a certain product to the cart. However, there is only a certain number of products in the storage, so it not only adds a number of products to the cart, but also subtract the number from the number of products in the product table. It also has to check if the number of products is more than 0. If not we can't add the product to cart.

Calling the function

```
1  select add_to_cart(308,101,40);
```

Data Output | Messages | Notifications

	add_to_cart boolean
1	true

Cart table before calling the function

	cart_number [PK] integer	cart_quantity integer	product_id integer
1	101	141	308

Product table before calling the function

	product_id [PK] integer	price integer	description character varying	category_name integer	product_count integer
1	308	7510	vibrant	1	152

Cart table after calling the function

	cart_number [PK] integer	cart_quantity integer	product_id integer
1	101	181	308

Product table after calling the function

	product_id [PK] integer	price integer	description character varying	category_name integer	product_count integer
1	308	7510	vibrant	1	112

If we want to put more products than we have in storage

```
select add_to_cart(310, 230, 10000000)
```

Data Output | Messages | Notifications

	add_to_cart boolean
1	false

Trigger

```
CREATE FUNCTION add_to_wish_list()
RETURNS TRIGGER
AS $$
DECLARE
    product_count1 INT;
BEGIN
    SELECT product_count INTO product_count1 FROM product WHERE product_id = NEW.product_i
    IF NEW.wish_list_quantity > product_count1 THEN
     RAISE EXCEPTION 'Sorry, so many items are not available';
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER check_wish_list_quantity
BEFORE INSERT ON wish_list
FOR EACH ROW
EXECUTE FUNCTION add_to_wish_list();
```

This trigger is created to check if someone wants to add a certain product to the wish list and if there are less items in storage than we want to add to the wish list it raises an exception.

Exception

```
leductha.public> INSERT INTO wish_list(wish_list_quantity, wish_list_date, wish_list_id, product_id)
                 VALUES (150, '2003-03-04', 555, 666)
[2023-05-04 13:05:09] [P0001] ERROR: Sorry, the requested quantity is not available
[2023-05-04 13:05:09] Where: PL/pgSQL function add_to_wish_list() line 4 at RAISE
```

View

```sql
CREATE VIEW name_address AS
SELECT person.person_id, person.lastname, a.country, a.street
 FROM personal_information person
JOIN address a ON person.person_id = a.person_id
```

This view joins two tables - personal information and address. It takes person_id, lastname add joins it with a persons street address and country.

Table

person_id	lastname	country	street
1	Novák	Lichtenštejnsko	Na Bojišti 26
2	Jelínková	Lotyšsko	Mlékárenská 478
3	Benešová	Východní Timor	Spojovací 95
4	Tichá	Panama	Semická 478
5	Černá	Vietnam	U Balabenky 645
6	Říhová	Egypt	U Mlýnského Rybníka 16
7	Čermáková	Maďarsko	Pod Lochkovem 1
8	Malá	Peru	Pohnertova 6
9	Holubová	Kamerun	U Hostivařského Nádraží 50
10	Machová	San Marino	Budapešťská 84
11	Čermák	Kazachstán	Dělostřelecká 654
12	Kučera	Srí Lanka	Uljanovská 13
13	Doležalová	Nikaragua	Moldavská 8
14	Benešová	Čad	Za Arielem 5
15	Kříž	Německo	U Pumpy 3
16	Křížová	Guyana	Habartovská 5
17	Zeman	Irák	U Dálnice 29
18	Bartošová	Sierra Leone	Pod Čertovou Skalou 896
19	Hájek	Saúdská Arábie	Veronské Nám. 7

Index

```
explain (analyze) select * from personal_information where lastname = 'Novak'
```

Output | ON personal_informat...lastname, birthday);

5 rows

	QUERY PLAN
1	Seq Scan on personal_information (cost=0…
2	Filter: ((lastname)::text = 'Novak'::te…
3	Rows Removed by Filter: 31997
4	Planning Time: 0.192 ms
5	Execution Time: 12.765 ms

From approximately 32k records, we want to find all the people with the last name Novak. It takes approximately 12.765ms.

Index function

```
CREATE INDEX personal_information_index
    ON personal_information(person_id, email, firstname, lastname,  birthday);
explain (analyze) select * from personal_information where lastname = 'Novak'
```

Output | Result 43-2

5 rows

	QUERY PLAN
1	Seq Scan on personal_information (cost=0…
2	Filter: ((lastname)::text = 'Novak'::te…
3	Rows Removed by Filter: 31997
4	Planning Time: 0.506 ms
5	Execution Time: 3.668 ms

After indexing all the columns in personal information, the execution time goes from 12ms to 3ms.