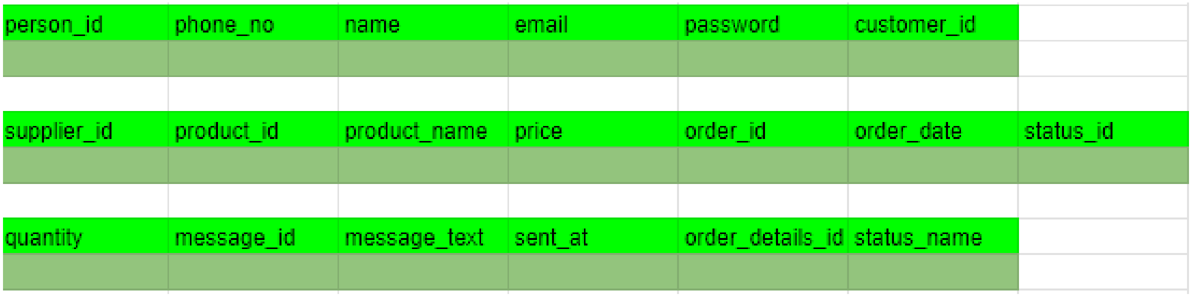
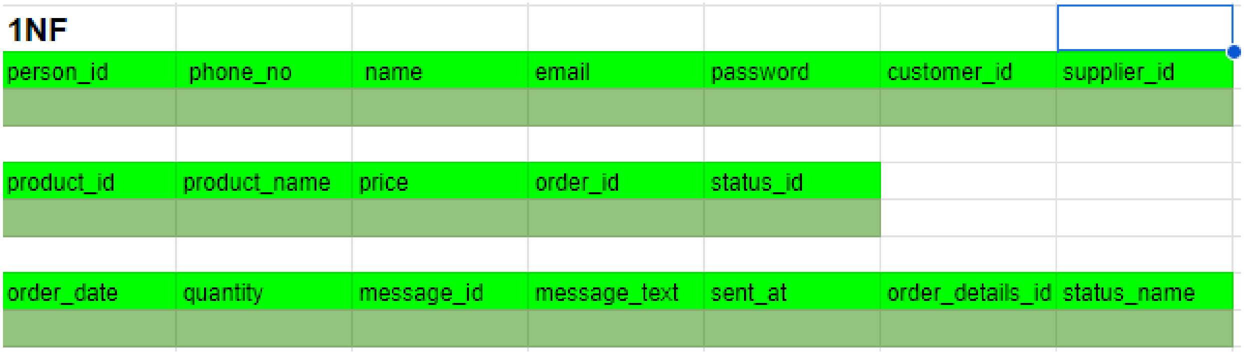
DBMS Project Review II
Group 7

1. Construct a universal table related to your assigned project. The table should consist of at least 12 attributes.



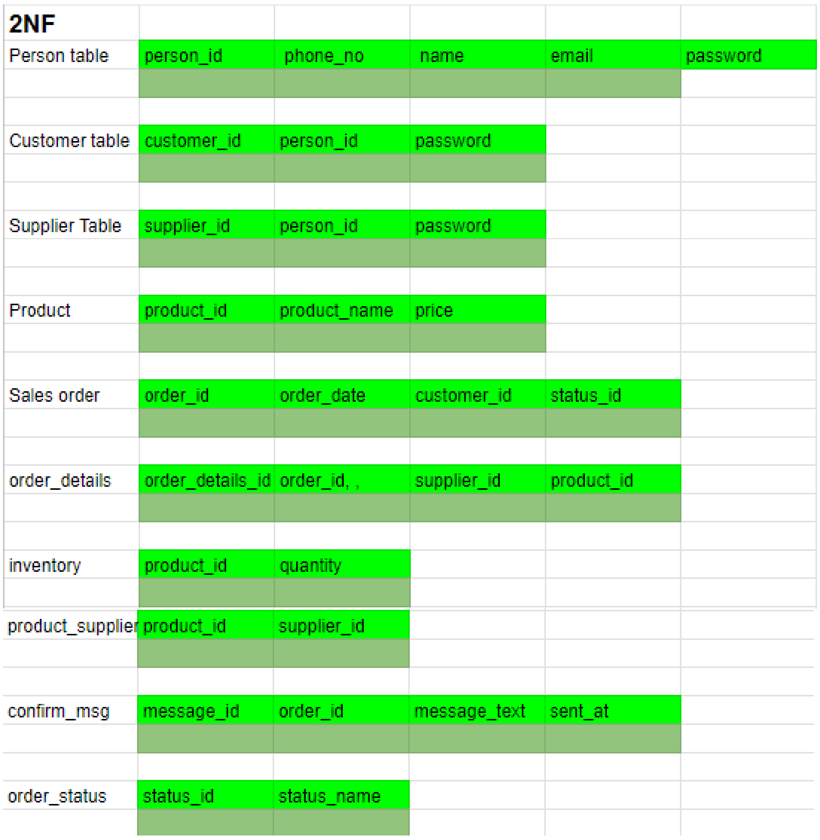
person_id	phone_no	name	email	password	customer_id	
supplier_id	product_id	product_name	price	order_id	order_date	status_id
quantity	message_id	message_text	sent_at	order_details_id	status_name	

2. Perform normalization process to ensure that these relations satisfy 1 NF, 2NF and 3 NF.



1NF						
person_id	phone_no	name	email	password	customer_id	supplier_id
product_id	product_name	price	order_id	status_id		
order_date	quantity	message_id	message_text	sent_at	order_details_id	status_name

There are no multivalued attributes. So table is in 1 NF form.

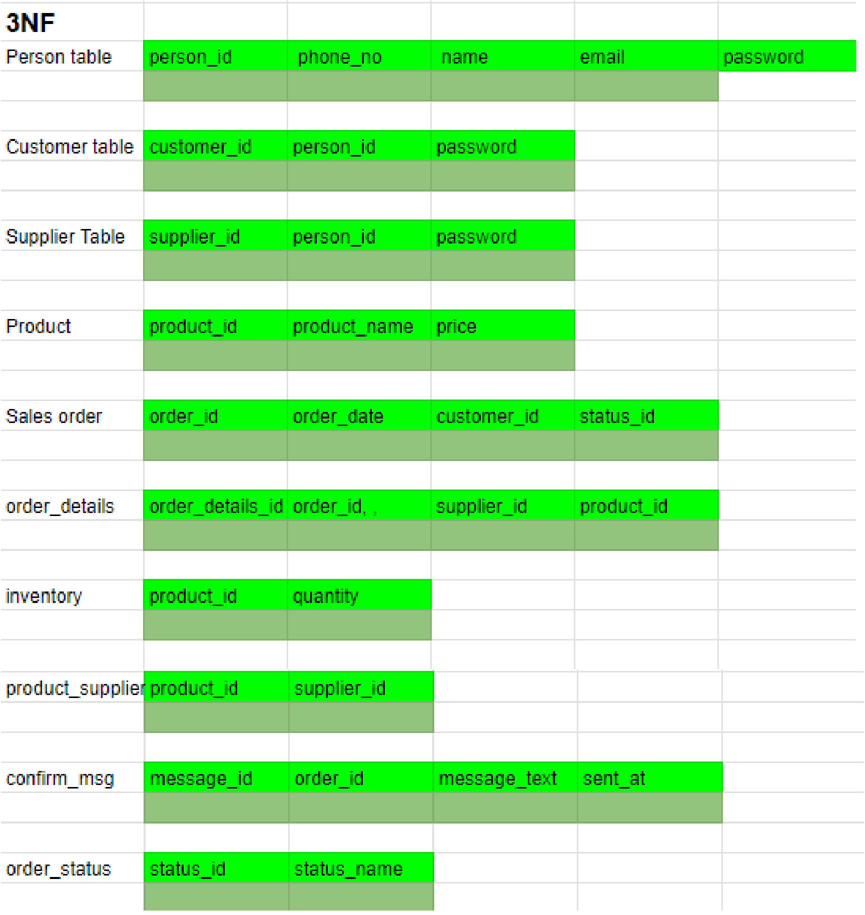


2NF					
Person table	person_id	phone_no	name	email	password
Customer table	customer_id	person_id	password		
Supplier Table	supplier_id	person_id	password		
Product	product_id	product_name	price		
Sales order	order_id	order_date	customer_id	status_id	
order_details	order_details_id	order_id, ,	supplier_id	product_id	
inventory	product_id	quantity			
product_supplier	product_id	supplier_id			
confirm_msg	message_id	order_id	message_text	sent_at	
order_status	status_id	status_name			

A table is in 2NF if:

- It is already in 1NF.
- It does not have any partial dependency, i.e., no non-prime attribute is dependent on a part of any candidate key.

Here, All tables are in 2NF since no partial dependencies exist.



3NF

Person table	person_id	phone_no	name	email	password
Customer table	customer_id	person_id	password		
Supplier Table	supplier_id	person_id	password		
Product	product_id	product_name	price		
Sales order	order_id	order_date	customer_id	status_id	
order_details	order_details_id	order_id, ,	supplier_id	product_id	
inventory	product_id	quantity			
product_supplier	product_id	supplier_id			
confirm_msg	message_id	order_id	message_text	sent_at	
order_status	status_id	status_name			

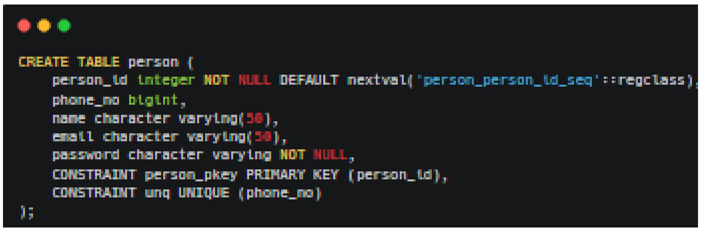
A table is in 3NF if:

- It is already in 2NF.
- It has no transitive dependency, i.e., non-key attributes are not dependent on other non-key attributes.

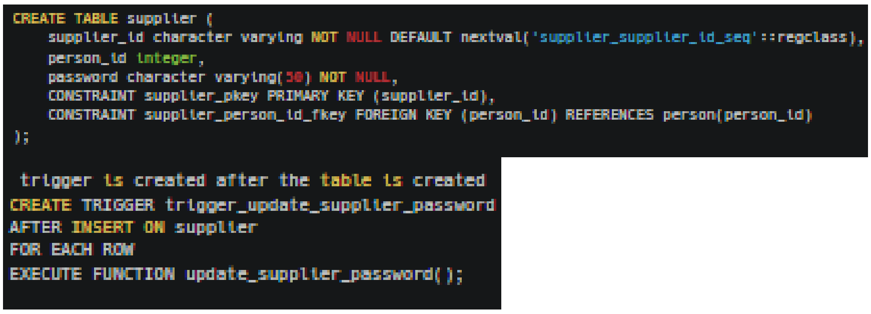
No partial or transitive dependencies exist among the non-key attributes. So the table is in 3NF

3. Based on this set of normalized relations obtained, **create at least 4 tables by writing proper DDL statements** (WITH CONSTRAINTS SET WHEREVER NECESSARY).

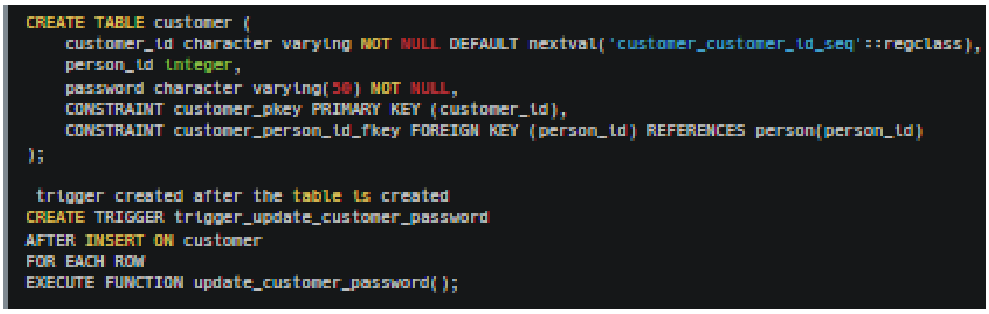
Data Definition Language Statements used are:



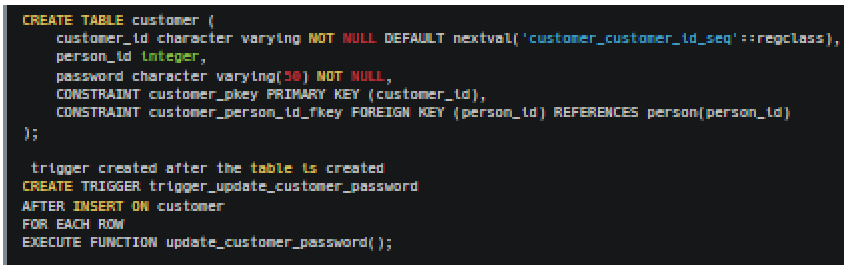
```
CREATE TABLE person (
    person_id integer NOT NULL DEFAULT nextval('person_person_id_seq'::regclass),
    phone_no bigint,
    name character varying(30),
    email character varying(30),
    password character varying NOT NULL,
    CONSTRAINT person_pkey PRIMARY KEY (person_id),
    CONSTRAINT unq UNIQUE (phone_no)
);
```



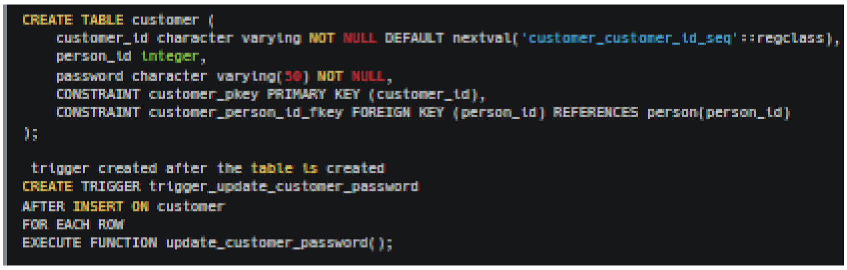
```
CREATE TABLE supplier (
    supplier_id character varying NOT NULL DEFAULT nextval('supplier_supplier_id_seq'::regclass),
    person_id integer,
    password character varying(30) NOT NULL,
    CONSTRAINT supplier_pkey PRIMARY KEY (supplier_id),
    CONSTRAINT supplier_person_id_fkey FOREIGN KEY (person_id) REFERENCES person(person_id)
);

 trigger is created after the table is created
CREATE TRIGGER trigger_update_supplier_password
AFTER INSERT ON supplier
FOR EACH ROW
EXECUTE FUNCTION update_supplier_password();
```



```
CREATE TABLE customer (
    customer_id character varying NOT NULL DEFAULT nextval('customer_customer_id_seq'::regclass),
    person_id integer,
    password character varying(50) NOT NULL,
    CONSTRAINT customer_pkey PRIMARY KEY (customer_id),
    CONSTRAINT customer_person_id_fkey FOREIGN KEY (person_id) REFERENCES person(person_id)
);

 trigger created after the table is created
CREATE TRIGGER trigger_update_customer_password
AFTER INSERT ON customer
FOR EACH ROW
EXECUTE FUNCTION update_customer_password();
```



```
CREATE TABLE customer (
    customer_id character varying NOT NULL DEFAULT nextval('customer_customer_id_seq'::regclass),
    person_id integer,
    password character varying(50) NOT NULL,
    CONSTRAINT customer_pkey PRIMARY KEY (customer_id),
    CONSTRAINT customer_person_id_fkey FOREIGN KEY (person_id) REFERENCES person(person_id)
);

 trigger created after the table is created
CREATE TRIGGER trigger_update_customer_password
AFTER INSERT ON customer
FOR EACH ROW
EXECUTE FUNCTION update_customer_password();
```



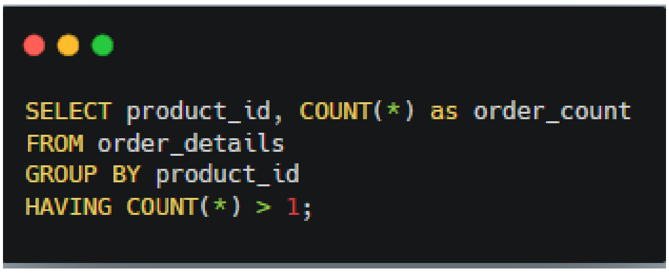
```
CREATE TABLE customer (
    customer_id character varying NOT NULL DEFAULT nextval('customer_customer_id_seq'::regclass),
    person_id integer,
    password character varying(50) NOT NULL,
    CONSTRAINT customer_pkey PRIMARY KEY (customer_id),
    CONSTRAINT customer_person_id_fkey FOREIGN KEY (person_id) REFERENCES person(person_id)
);

 trigger created after the table is created
CREATE TRIGGER trigger_update_customer_password
AFTER INSERT ON customer
FOR EACH ROW
EXECUTE FUNCTION update_customer_password();
```

4. Insert appropriate data into the tables and generate **10 queries.**

The queries must be based on the following:

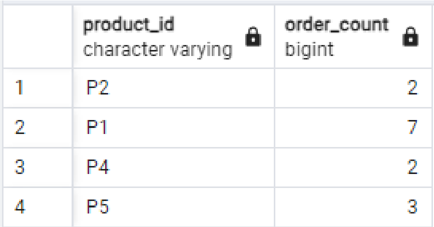
i. Aggregate functions, Group by…having

Query:



```
SELECT product_id, COUNT(*) as order_count
FROM order_details
GROUP BY product_id
HAVING COUNT(*) > 1;
```

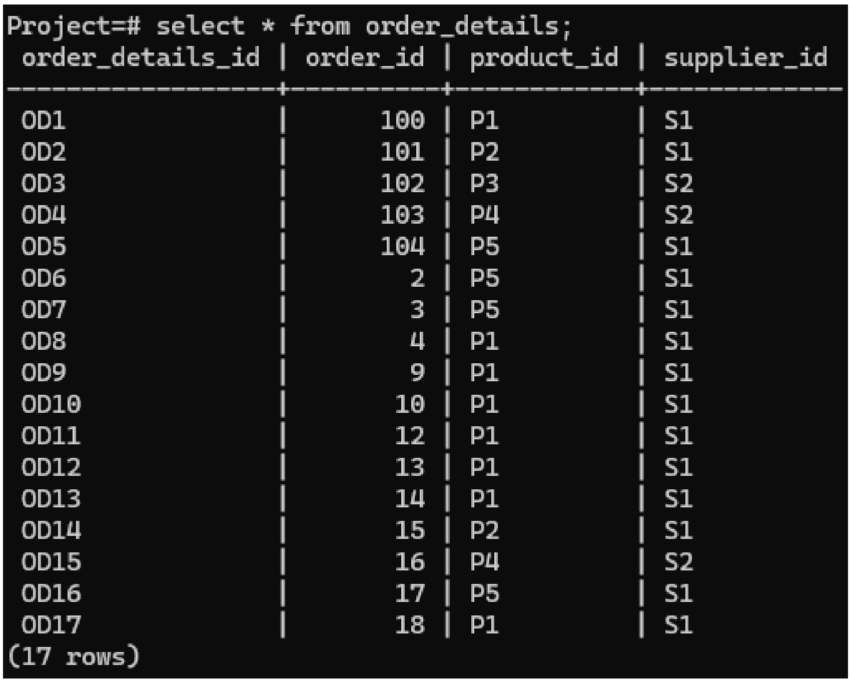
Output:



	product_id character varying	order_count bigint
1	P2	2
2	P1	7
3	P4	2
4	P5	3

Each of these products have been ordered that much amount of time.

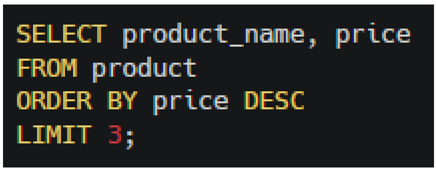
To confirm the output we can check the data:



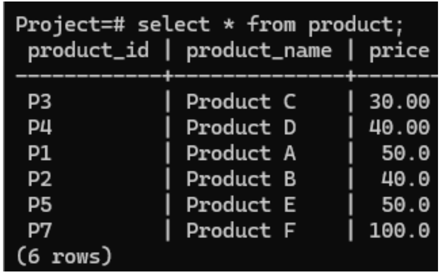
```
Project=# select * from order_details;
 order_details_id | order_id | product_id | supplier_id
------------------+----------+------------+-------------
 OD1              |      100 | P1         | S1
 OD2              |      101 | P2         | S1
 OD3              |      102 | P3         | S2
 OD4              |      103 | P4         | S2
 OD5              |      104 | P5         | S1
 OD6              |        2 | P5         | S1
 OD7              |        3 | P5         | S1
 OD8              |        4 | P1         | S1
 OD9              |        9 | P1         | S1
 OD10             |       10 | P1         | S1
 OD11             |       12 | P1         | S1
 OD12             |       13 | P1         | S1
 OD13             |       14 | P1         | S1
 OD14             |       15 | P2         | S1
 OD15             |       16 | P4         | S2
 OD16             |       17 | P5         | S1
 OD17             |       18 | P1         | S1
(17 rows)
```

ii. Order by

Ans:

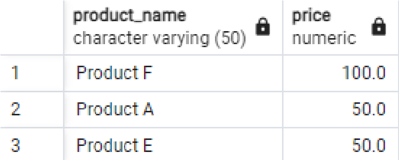


```
SELECT product_name, price
FROM product
ORDER BY price DESC
LIMIT 3;
```



```
Project=# select * from product;
 product_id | product_name | price
------------+--------------+------
 P3         | Product C    | 30.00
 P4         | Product D    | 40.00
 P1         | Product A    |  50.0
 P2         | Product B    |  40.0
 P5         | Product E    |  50.0
 P7         | Product F    | 100.0
(6 rows)
```

Output:

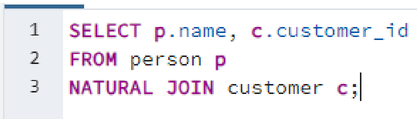


	product_name character varying (50)	price numeric
1	Product F	100.0
2	Product A	50.0
3	Product E	50.0

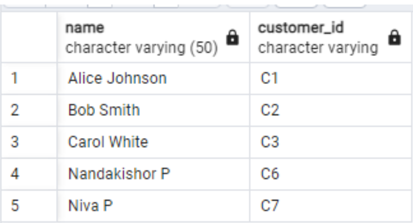
Based on the shell image given here, we can see that the top 3 products with high price are retrieved correct through the order by clause used.

iii. Join, Outer Join

Natural Join:

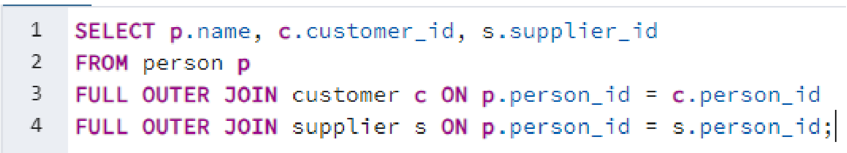


```
SELECT p.name, c.customer_id
FROM person p
NATURAL JOIN customer c;
```

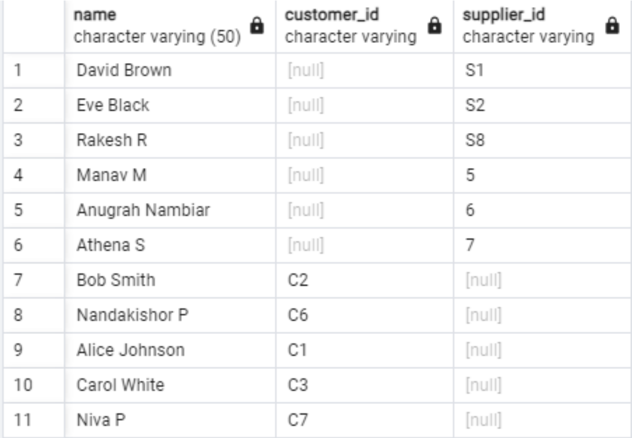


	name character varying (50)	customer_id character varying
1	Alice Johnson	C1
2	Bob Smith	C2
3	Carol White	C3
4	Nandakishor P	C6
5	Niva P	C7

Outer Join:



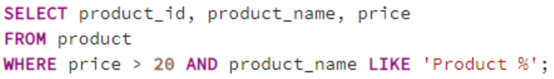
```
SELECT p.name, c.customer_id, s.supplier_id
FROM person p
FULL OUTER JOIN customer c ON p.person_id = c.person_id
FULL OUTER JOIN supplier s ON p.person_id = s.person_id;
```



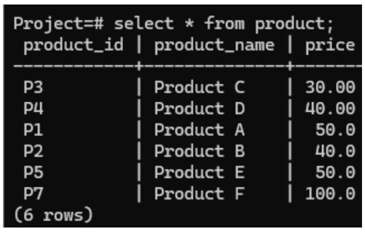
	name character varying (50)	customer_id character varying	supplier_id character varying
1	David Brown	[null]	S1
2	Eve Black	[null]	S2
3	Rakesh R	[null]	S8
4	Manav M	[null]	5
5	Anugrah Nambiar	[null]	6
6	Athena S	[null]	7
7	Bob Smith	C2	[null]
8	Nandakishor P	C6	[null]
9	Alice Johnson	C1	[null]
10	Carol White	C3	[null]
11	Niva P	C7	[null]

NATURAL JOIN only returns matching rows (inner join behavior), while the FULL OUTER JOIN returns all rows from all tables, using NULL where there's no match.

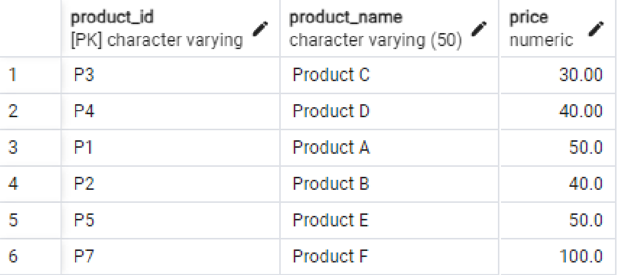
iv. Query having Boolean operators



```
SELECT product_id, product_name, price
FROM product
WHERE price > 20 AND product_name LIKE 'Product %';
```



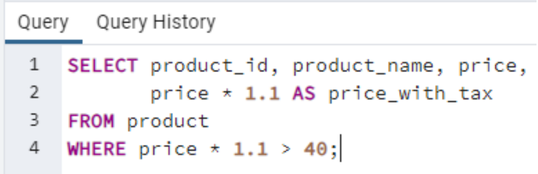
```
Project=# select * from product;
 product_id | product_name | price
------------+--------------+-------
 P3         | Product C    | 30.00
 P4         | Product D    | 40.00
 P1         | Product A    |  50.0
 P2         | Product B    |  40.0
 P5         | Product E    |  50.0
 P7         | Product F    | 100.0
(6 rows)
```



	product_id [PK] character varying	product_name character varying (50)	price numeric
1	P3	Product C	30.00
2	P4	Product D	40.00
3	P1	Product A	50.0
4	P2	Product B	40.0
5	P5	Product E	50.0
6	P7	Product F	100.0

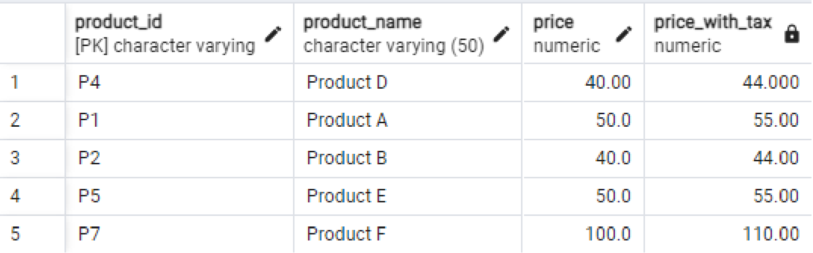
As you can see, all the products have price above 20. So, the query returns all products fetched.

v. Query having arithmetic operators



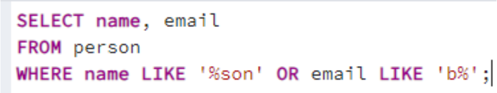
```
SELECT product_id, product_name, price,
       price * 1.1 AS price_with_tax
FROM product
WHERE price * 1.1 > 40;
```

Only product 3 has the constraint price * 1.1 is < 40. So it is not fetched in the query



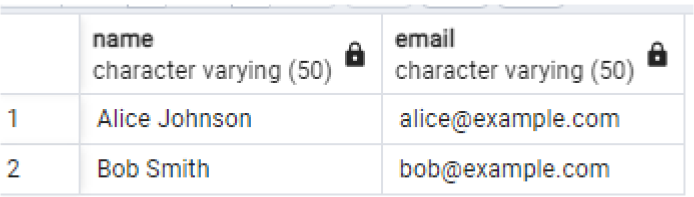
	product_id [PK] character varying	product_name character varying (50)	price numeric	price_with_tax numeric
1	P4	Product D	40.00	44.000
2	P1	Product A	50.0	55.00
3	P2	Product B	40.0	44.00
4	P5	Product E	50.0	55.00
5	P7	Product F	100.0	110.00

vi. A search query using string operators



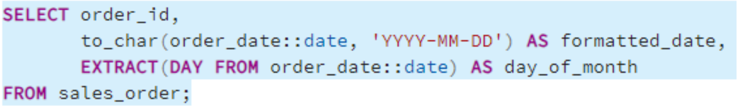
```
SELECT name, email
FROM person
WHERE name LIKE '%son' OR email LIKE 'b%';
```

Output:

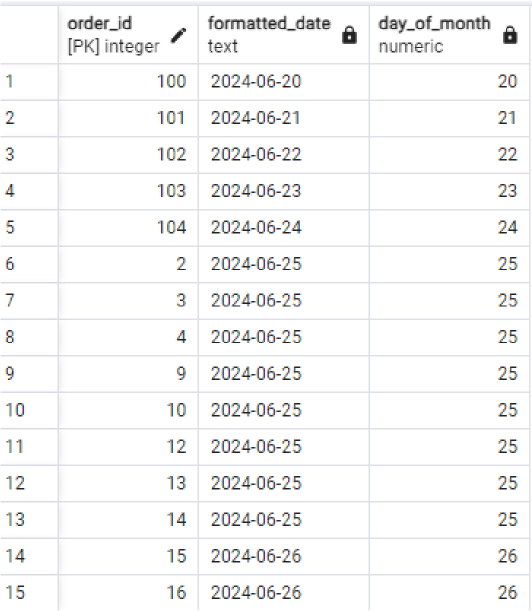


	name character varying (50)	email character varying (50)
1	Alice Johnson	alice@example.com
2	Bob Smith	bob@example.com

vii. Usage of to_char, extract



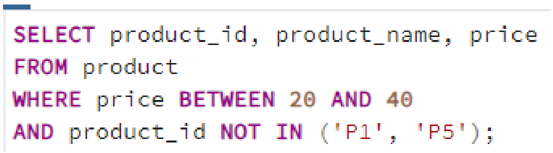
```
SELECT order_id,
       to_char(order_date::date, 'YYYY-MM-DD') AS formatted_date,
       EXTRACT(DAY FROM order_date::date) AS day_of_month
FROM sales_order;
```



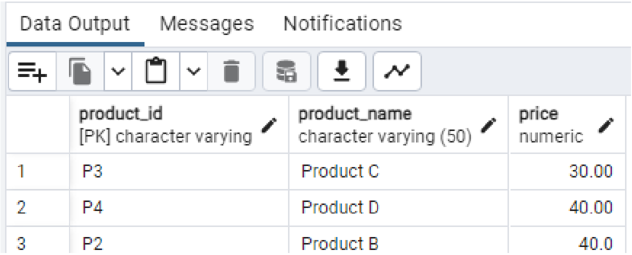
	order_id [PK] integer	formatted_date text	day_of_month numeric
1	100	2024-06-20	20
2	101	2024-06-21	21
3	102	2024-06-22	22
4	103	2024-06-23	23
5	104	2024-06-24	24
6	2	2024-06-25	25
7	3	2024-06-25	25
8	4	2024-06-25	25
9	9	2024-06-25	25
10	10	2024-06-25	25
11	12	2024-06-25	25
12	13	2024-06-25	25
13	14	2024-06-25	25
14	15	2024-06-26	26
15	16	2024-06-26	26

Extract is used to extract various parts of a date/time value here specifically the day from the date

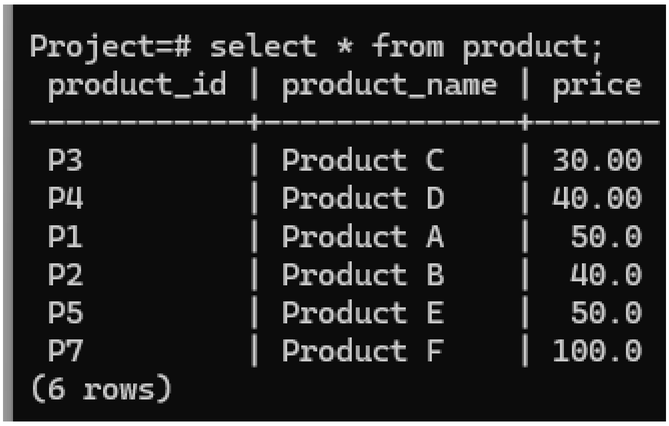
viii. Between, IN, Not between, Not IN



```
SELECT product_id, product_name, price
FROM product
WHERE price BETWEEN 20 AND 40
AND product_id NOT IN ('P1', 'P5');
```



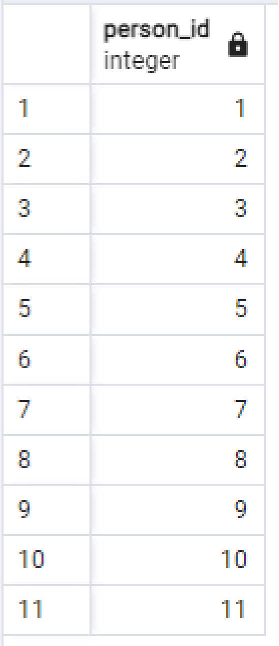
	product_id [PK] character varying	product_name character varying (50)	price numeric
1	P3	Product C	30.00
2	P4	Product D	40.00
3	P2	Product B	40.0



```
Project=# select * from product;
 product_id | product_name | price
------------+--------------+-------
 P3         | Product C    | 30.00
 P4         | Product D    | 40.00
 P1         | Product A    |  50.0
 P2         | Product B    |  40.0
 P5         | Product E    |  50.0
 P7         | Product F    | 100.0
(6 rows)
```

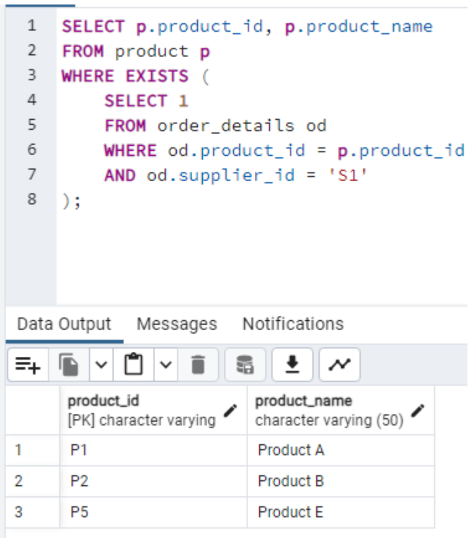
ix. Set operations

```
(SELECT person_id FROM customer)
UNION
(SELECT person_id FROM supplier)
ORDER BY person_id;
```



	person_id integer
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11

x. Subquery using EXISTS / NOT EXISTS, ANY, ALL

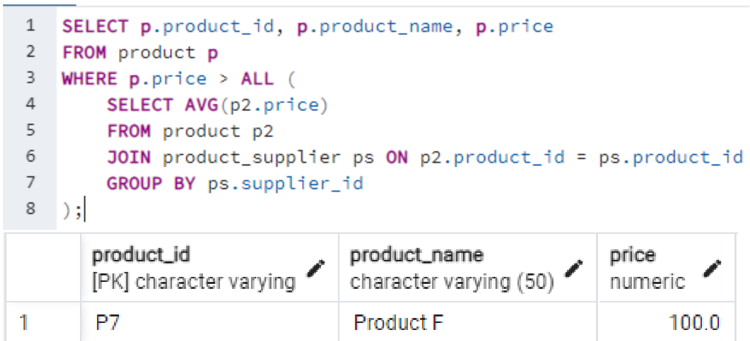


```sql
SELECT p.product_id, p.product_name
FROM product p
WHERE EXISTS (
    SELECT 1
    FROM order_details od
    WHERE od.product_id = p.product_id
    AND od.supplier_id = 'S1'
);
```

	product_id [PK] character varying	product_name character varying (50)
1	P1	Product A
2	P2	Product B
3	P5	Product E

EXISTS is used to check if any rows are returned by the subquery.

Using All:



```sql
SELECT p.product_id, p.product_name, p.price
FROM product p
WHERE p.price > ALL (
    SELECT AVG(p2.price)
    FROM product p2
    JOIN product_supplier ps ON p2.product_id = ps.product_id
    GROUP BY ps.supplier_id
);
```

	product_id [PK] character varying	product_name character varying (50)	price numeric
1	P7	Product F	100.0

Product F has higher price than all the average prices calculated for each supplier.

5. Execute the queries and paste the screenshots with results.

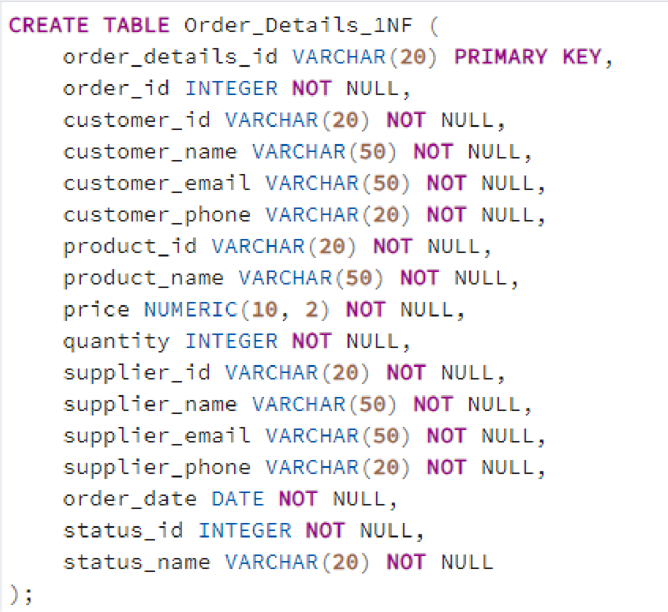
Additional Information

a. Assumptions/Constraints:

a. person_id: Unique identifier for each person. One person will have only one ID.
b. phone_no: Unique phone number for each person. One person will have only one phone number.
c. name: Name of the person. Multiple people may have the same name.
d. email: Unique email address for each person. One person will have only one email.
e. password: Password for the person's account. Can be changed but must exist.
f. customer_id: Unique identifier for each customer. One customer will have only one ID.
g. supplier_id: Unique identifier for each supplier. One supplier will have only one ID.
h. product_id: Unique identifier for each product. One product will have only one ID.
i. product_name: Name of the product. Different products may have the same name.
j. price: Current price of the product. One product will have only one price at a time.
k. quantity: Current quantity of the product in inventory.
l. order_id: Unique identifier for each order. One order will have only one ID.
m.order_date: Date when the order was placed.
n. status_id: Identifier for the current status of the order.

b. Table in 1 NF: With the attributes identified, form a table in 1NF. The table should consistof at least 12 attributes.

Ans:



```
CREATE TABLE Order_Details_1NF (
    order_details_id VARCHAR(20) PRIMARY KEY,
    order_id INTEGER NOT NULL,
    customer_id VARCHAR(20) NOT NULL,
    customer_name VARCHAR(50) NOT NULL,
    customer_email VARCHAR(50) NOT NULL,
    customer_phone VARCHAR(20) NOT NULL,
    product_id VARCHAR(20) NOT NULL,
    product_name VARCHAR(50) NOT NULL,
    price NUMERIC(10, 2) NOT NULL,
    quantity INTEGER NOT NULL,
    supplier_id VARCHAR(20) NOT NULL,
    supplier_name VARCHAR(50) NOT NULL,
    supplier_email VARCHAR(50) NOT NULL,
    supplier_phone VARCHAR(20) NOT NULL,
    order_date DATE NOT NULL,
    status_id INTEGER NOT NULL,
    status_name VARCHAR(20) NOT NULL
);
```

c. Functional Dependencies: Identify all the functional dependencies in the 1NF table formed

Ans:

1. order_details_id -> all other attributes
2. order_id -> customer_id, order_date, status_id
3. customer_id -> customer_name, customer_email, customer_phone
4. product_id -> product_name, price
5. supplier_id -> supplier_name, supplier_email, supplier_phone
6. status_id -> status_name

d. Tables in 2NF: Form tables that are in 2NF and give proper justification for the same.

To make the tables into 2NF, we created the following tables earlier,

1. Order_Details (order_details_id, order_id, product_id, supplier_id, quantity)
2. Orders (order_id, customer_id, order_date, status_id)
3. Customers (customer_id, customer_name, customer_email, customer_phone)
4. Products (product_id, product_name, price)
5. Suppliers (supplier_id, supplier_name, supplier_email, supplier_phone)

6. Order_Status (status_id, status_name)

Each non-key attribute is now fully functionally dependent on the primary key of its table.

e. Tables in 3NF: Form tables that are in 3NF and give proper justification for the same.

The tables from 2NF are already in 3NF because there are no transitive dependencies. Each non-key attribute is dependent only on the primary key, not on other non-key attributes.

RELATIONAL SCHEMA UPDATED:

Person(person_id, phone_no, name, email, password)

Customer(person_id, customer_id, password)

Supplier(person_id, supplier_id, password)

Product(product_id, product_name, price)

Sales_Order(order_id, status_id order_date, customer_id)

order_details(order_details_id, order_id, product_id, supplier_id)

inventory(product_id, quantity)

product_supplier(product_id, supplier_id)

confirm_msg(message_id, order_id, message_text, sent_at)

order_status(status_id, status_name)