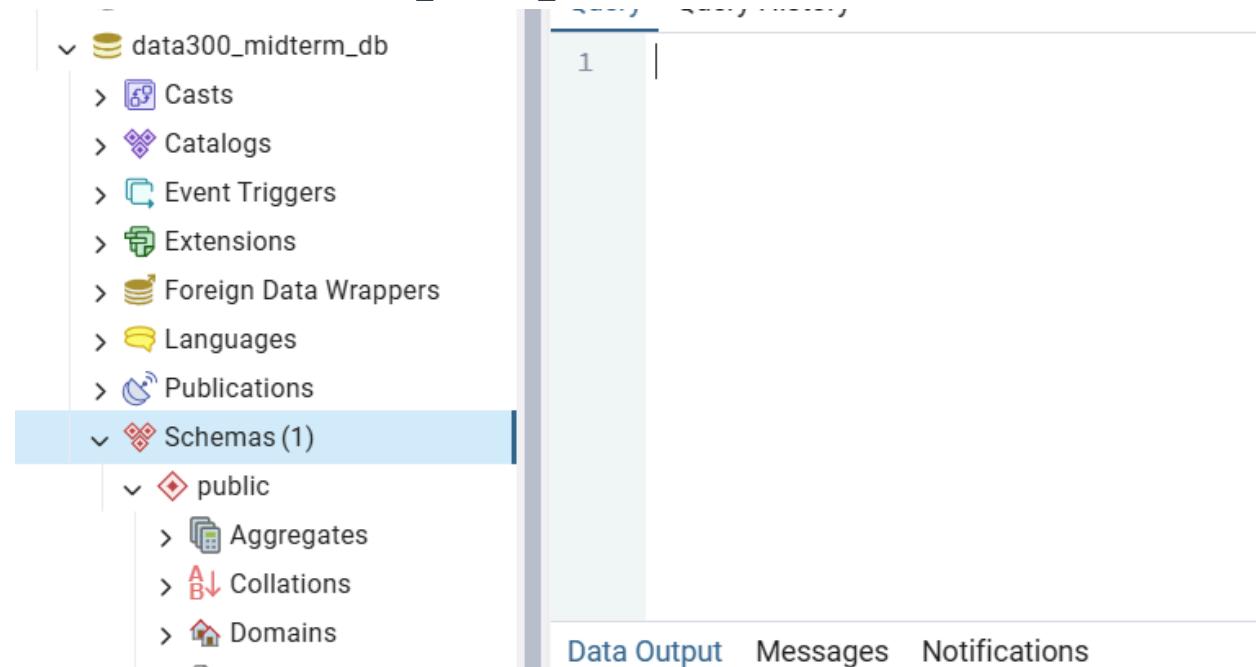


Name: Min Thant Hein
ID: PIUS20230001

Created the database “data300_midterm_db”:



```
#  
##
```

Creating tables:

Creating Customer table:

The screenshot shows the pgAdmin interface for managing a PostgreSQL database. On the left, the object browser tree is visible, showing various database objects like Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, Schemas, Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables. Under the 'Tables' node, there is a single entry for 'customer'. The main panel is a query editor titled 'Query' with the sub-tab 'Query History' selected. It contains the SQL code for creating the 'Customer' table:

```
-- 1) Customer
CREATE TABLE Customer (
    customer_id SERIAL PRIMARY KEY,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20) UNIQUE,
    date_of_birth DATE NOT NULL CHECK (date_of_birth <= CURRENT_DATE - INTERVAL '16 years'),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    CONSTRAINT created_at_imutable CHECK (created_at IS NOT NULL)
);

-- 2) Product table
CREATE TABLE Product (
    product_id TNT PRIMARY KEY.
```

Below the query editor, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the message 'CREATE TABLE'. Underneath, it says 'Query returned successfully in 206 msec.'

#

Creating Product table:

The screenshot shows the pgAdmin interface for a database named 'data300_midterm_db'. In the left sidebar, under the 'Tables' section, the 'product' table is selected. The main pane displays a SQL query for creating the 'Product' table:

```
-- 2) Product table
CREATE TABLE Product (
    product_id INT PRIMARY KEY,
    sku VARCHAR(50) UNIQUE NOT NULL,
    name VARCHAR(200) NOT NULL,
    unit_price DECIMAL(10,2) NOT NULL CHECK (unit_price > 0),
    stock_qty INT NOT NULL CHECK (stock_qty >= 0),
    status VARCHAR(15) NOT NULL CHECK (status IN ('ACTIVE', 'DISCONTINUED')),
    CONSTRAINT discontinued_stock_zero CHECK (
        (status = 'DISCONTINUED' AND stock_qty = 0) OR status = 'ACTIVE'
    )
);
```

The 'Messages' tab at the bottom indicates that the query was executed successfully.

#

Creating "Order" Table:

The screenshot shows the pgAdmin interface for the same database. In the left sidebar, under the 'Tables' section, the 'Order' table is selected. The main pane displays a SQL query for creating the 'Order' table:

```
-- 3) Order Table
CREATE TABLE "Order" (
    order_id SERIAL PRIMARY KEY,
    customer_id INT NOT NULL REFERENCES Customer(customer_id) ON DELETE RESTRICT,
    order_status VARCHAR(15) DEFAULT 'PENDING' NOT NULL CHECK (order_status IN ('PENDING', 'PAI
    payment_method VARCHAR(10) NOT NULL CHECK (payment_method IN ('CARD', 'TRANSFER', 'COD')),
    ordered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    shipping_country CHAR(2) NOT NULL,
    shipping_city VARCHAR(100) NOT NULL
);
```

The 'Messages' tab at the bottom indicates that the query was executed successfully.

#

Creating OrderItem Table:

The screenshot shows a database management interface with a sidebar navigation menu and a main query editor area.

Navigation Sidebar:

- > Aggregates
- > Collations
- > Domains
- > FTS Configurations
- > FTS Dictionaries
- > FTS Parsers
- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > Sequences
- > Tables (4)
- > Order
- > customer
- > orderitem
- > product
- > Trigger Functions
- > Types

Query Editor Area:

Query History

```
35
36 -- 4)OrderItem
37 CREATE TABLE OrderItem (
38     order_id INT NOT NULL REFERENCES "Order"(order_id) ON DELETE CASCADE,
39     line_no INT NOT NULL CHECK (Line_no >= 1),
40     product_id INT NOT NULL REFERENCES Product(product_id) ON DELETE RESTRICT,
41     quantity INT NOT NULL CHECK (quantity >= 1),
42     unit_price_at_order DECIMAL(10,2) NOT NULL CHECK (unit_price_at_order > 0),
43     PRIMARY KEY (order_id, line_no),
44     CONSTRAINT unique_product_per_order UNIQUE (order_id, product_id)
45 );
46
47
48
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 77 msec.

#

Cross-table business rules to enforce:

Running the trigger function, "fn_check_order_status":

This trigger handles the UPDATE key on the Order table!

The screenshot shows the pgAdmin 4 interface for PostgreSQL. The left sidebar displays the database schema with nodes for FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, Tables (4), Trigger Functions (1), Types, Views, Subscriptions, and the current database 'dreamhome'. The main query editor window shows the SQL code for creating the trigger function:

```
-- 1. Create the trigger function
CREATE OR REPLACE FUNCTION fn_check_order_status()
RETURNS TRIGGER AS $$
DECLARE
    item_count INT;
BEGIN
    -- Check only if the status is being changed to PAID or SHIPPED
    IF NEW.order_status IN ('PAID', 'SHIPPED') AND OLD.order_status <> NEW.order_status THEN
        -- Count the items for this order
        SELECT COUNT(*)
        INTO item_count
        FROM OrderItem
        WHERE order_id = NEW.order_id;
    END IF;
    -- Your logic here to prevent deletion of last item from PAID or SHIPPED order
END;
$$ LANGUAGE plpgsql;
```

The 'Messages' tab in the bottom right shows the output: 'CREATE TRIGGER' and 'Query returned successfully in 79 msec.'

To prevent deleting the last item from a 'PAID' or 'SHIPPED' order, I also add a similar trigger on the OrderItem table.

#

Running the trigger for rule2 and rule3 for OrderItem table:

The screenshot shows a database interface with a sidebar containing various schema objects like Aggregates, Collations, Domains, FTS Configurations, etc. The 'Tables' section is expanded, showing Order, customer, orderitem, and product tables. The 'Trigger Functions' section is also expanded, showing two functions: fn_check_order_status() and fn_check_orderitem_rules(). The main area is a query editor titled 'Query' with the following SQL code:

```
1 -- 1. Create the trigger function
2 CREATE OR REPLACE FUNCTION fn_check_orderitem_rules()
3 RETURNS TRIGGER AS $$ 
4 DECLARE
5     v_stock_qty INT;
6     v_product_status VARCHAR(20);
7 BEGIN
8     -- Get product info
9     SELECT stock_qty, product_status
10    INTO v_stock_qty, v_product_status
11   FROM Product
12  WHERE product_id = NEW.product_id;
13
14     -- Rule 3: Check if product is DISCONTINUED (only on INSERT)
15     IF TC_OP = 'ITNCEDT1' AND v_product_status = 'DISCONTINUED' THEN
16         -- Logic for Rule 3
17     END IF;
18
19     -- Rule 2: Check if stock_qty is less than or equal to zero
20     IF v_stock_qty <= 0 THEN
21         -- Logic for Rule 2
22     END IF;
23
24     RETURN NEW;
25 END;
26 $$ LANGUAGE plpgsql;
```

The 'Messages' tab is selected, showing the message: 'Query returned successfully in 53 msec.'

#

Inserting Data:

Insertion Data into Customer table:

The screenshot shows a database interface with a sidebar containing various catalog objects like Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, and Schemas. The 'Schemas(1)' section is selected, revealing the 'public' schema which contains Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, and Sequences. The main panel displays a query window with the following SQL code:

```
1 INSERT INTO Customer (customer_id, full_name, email, phone, date_of_birth, created_at)
2 VALUES
3 (1, 'Alice Tan', 'alice.tan@example.com', '+66812345678', '1998-04-15', CURRENT_TIMESTAMP),
4 (2, 'David Lee', 'david.lee@example.com', '+66987654321', '1995-09-02', CURRENT_TIMESTAMP),
5 (3, 'Maya Wong', 'maya.wong@example.com', NULL, '2000-01-25', CURRENT_TIMESTAMP);
```

The 'Data Output' tab shows the result of the insert operation: 'INSERT 0 3'. Below it, a message states 'Query returned successfully in 54 msec.'

Table view:

The screenshot shows a database interface with a sidebar containing Languages, Publications, Schemas(1), and public schema objects. The 'Tables(4)' section is expanded, showing the 'customer' table selected. The main panel displays a query window with the following SQL code:

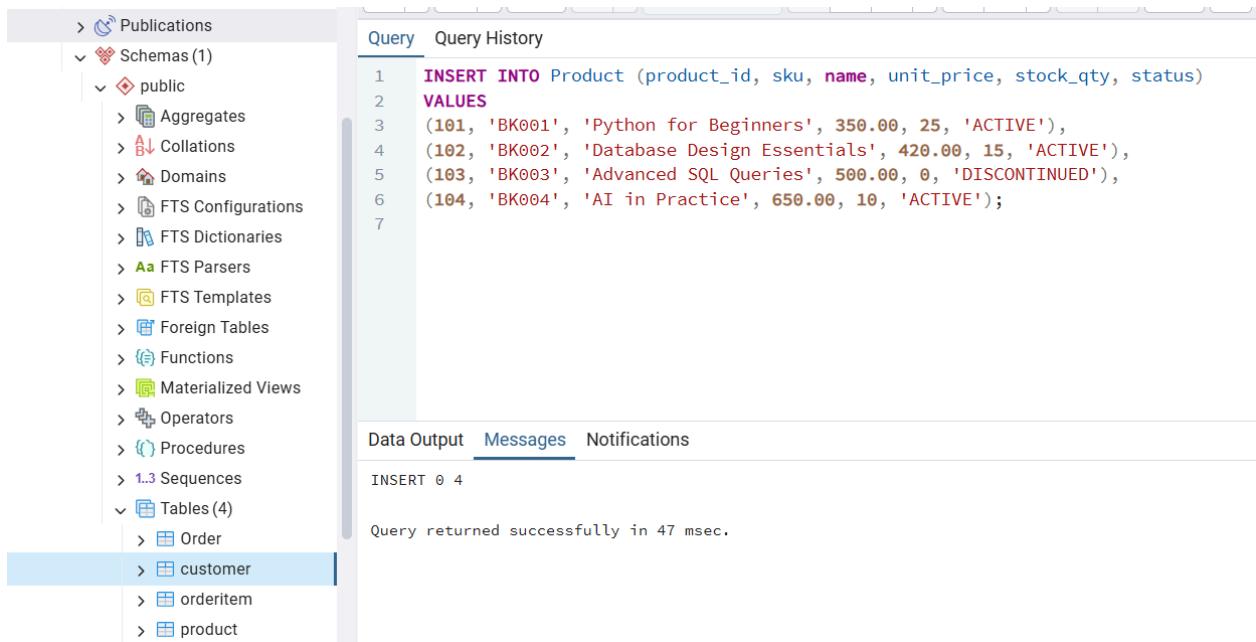
```
1 SELECT * FROM public.customer
2 ORDER BY customer_id ASC
```

The 'Data Output' tab shows the results of the select query. The table has columns: customer_id [PK] integer, full_name character varying (100), email character varying (100), phone character varying (20), date_of_birth date, and created_at timestamp without time zone. The data is as follows:

| | customer_id | full_name | email | phone | date_of_birth | created_at |
|---|-------------|-----------|-----------------------|--------------|---------------|---------------------------|
| 1 | 1 | Alice Tan | alice.tan@example.com | +66812345678 | 1998-04-15 | 2025-10-30 20:18:18.22431 |
| 2 | 2 | David Lee | david.lee@example.com | +66987654321 | 1995-09-02 | 2025-10-30 20:18:18.22431 |
| 3 | 3 | Maya Wong | maya.wong@example.com | [null] | 2000-01-25 | 2025-10-30 20:18:18.22431 |

##

Insertion Data into Product table:



The screenshot shows a database interface with a sidebar on the left containing a tree view of database objects. The 'Tables(4)' section is expanded, and the 'customer' table is selected, highlighted with a blue background. The main area is a query editor titled 'Query' with the sub-tab 'Query History'. It contains the following SQL code:

```
1 INSERT INTO Product (product_id, sku, name, unit_price, stock_qty, status)
2 VALUES
3 (101, 'BK001', 'Python for Beginners', 350.00, 25, 'ACTIVE'),
4 (102, 'BK002', 'Database Design Essentials', 420.00, 15, 'ACTIVE'),
5 (103, 'BK003', 'Advanced SQL Queries', 500.00, 0, 'DISCONTINUED'),
6 (104, 'BK004', 'AI in Practice', 650.00, 10, 'ACTIVE');
```

Below the code, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is active, showing the message 'INSERT 0 4' and the note 'Query returned successfully in 47 msec.'

#

Table view:

Query History

```
1 SELECT * FROM public.product
2 ORDER BY product_id ASC
```

Data Output Messages Notifications

Showing rows: 1 to 4

| | product_id [PK] integer | sku character varying (50) | name character varying (200) | unit_price numeric (10,2) | stock_qty integer | status character varying (15) |
|---|----------------------------|-------------------------------|---------------------------------|------------------------------|----------------------|----------------------------------|
| 1 | 101 | BK001 | Python for Beginners | 350.00 | 25 | ACTIVE |
| 2 | 102 | BK002 | Database Design Essentials | 420.00 | 15 | ACTIVE |
| 3 | 103 | BK003 | Advanced SQL Queries | 500.00 | 0 | DISCONTINUED |
| 4 | 104 | BK004 | AI in Practice | 650.00 | 10 | ACTIVE |

##

Insertion Data into Order table:

The screenshot shows the pgAdmin interface. On the left, the database tree is visible under the 'public' schema, showing tables like 'Order', 'customer', 'orderitem', and 'product'. The 'Tables' node is expanded. In the main pane, a query window displays an 'INSERT' statement into the 'Order' table. The query is as follows:

```

1  INSERT INTO "Order" (order_id, customer_id, order_status, payment_method, ordered_at, shipping_country, shipping_city)
2   VALUES
3   (1001, 1, 'PAID', 'CARD', CURRENT_TIMESTAMP, 'TH', 'Bangkok'),
4   (1002, 2, 'PENDING', 'TRANSFER', CURRENT_TIMESTAMP, 'TH', 'Chiang Mai'),
5   (1003, 3, 'SHIPPED', 'CARD', CURRENT_TIMESTAMP, 'TH', 'Phuket');
6

```

Below the query, the 'Messages' tab is selected, showing the message: 'Query returned successfully in 47 msec.'

Table view:

The screenshot shows the pgAdmin interface with a query window containing a 'SELECT' statement:

```

1  SELECT * FROM public."Order"
2  ORDER BY order_id ASC

```

Below the query, the 'Data Output' tab is selected, displaying the results of the query. The table has the following structure and data:

| | order_id [PK] integer | customer_id integer | order_status character varying (15) | payment_method character varying (10) | ordered_at timestamp without time zone | shipping_country character (2) | shipping_city character varying (100) |
|---|--------------------------|------------------------|--|--|---|-----------------------------------|--|
| 1 | 1001 | 1 | PAID | CARD | 2025-10-30 20:21:41.091083 | TH | Bangkok |
| 2 | 1002 | 2 | PENDING | TRANSFER | 2025-10-30 20:21:41.091083 | TH | Chiang Mai |
| 3 | 1003 | 3 | SHIPPED | CARD | 2025-10-30 20:21:41.091083 | TH | Phuket |

##

Insertion Data into OrderItem table:

The screenshot shows the pgAdmin 4 interface. On the left, the Object Browser tree is visible, showing the database structure. The 'Tables' node under 'public' schema has four entries: Order, customer, orderitem, and product. The 'orderitem' table is selected. The main pane displays a query editor window with the following content:

```
1 INSERT INTO OrderItem (order_id, line_no, product_id, quantity, unit_price_at_order)
2 VALUES
3 (1001, 1, 101, 2, 350.00),
4 (1001, 2, 102, 1, 420.00),
5 (1002, 1, 104, 1, 650.00),
6 (1003, 1, 101, 1, 350.00),
7 (1003, 2, 102, 2, 420.00);
8
```

Below the query editor, the 'Messages' tab is selected, showing the message: "Query returned successfully in 48 msec."

Table view:

Query History

```
1 SELECT * FROM public.orderitem
2 ORDER BY order_id ASC, line_no ASC
```

Data Output Messages Notifications

| | order_id [PK] integer | line_no [PK] integer | product_id integer | quantity integer | unit_price_at_order numeric (10,2) |
|---|--------------------------|-------------------------|-----------------------|---------------------|---------------------------------------|
| 1 | 1001 | 1 | 101 | 2 | 350.00 |
| 2 | 1001 | 2 | 102 | 1 | 420.00 |
| 3 | 1002 | 1 | 104 | 1 | 650.00 |
| 4 | 1003 | 1 | 101 | 1 | 350.00 |
| 5 | 1003 | 2 | 102 | 2 | 420.00 |

##