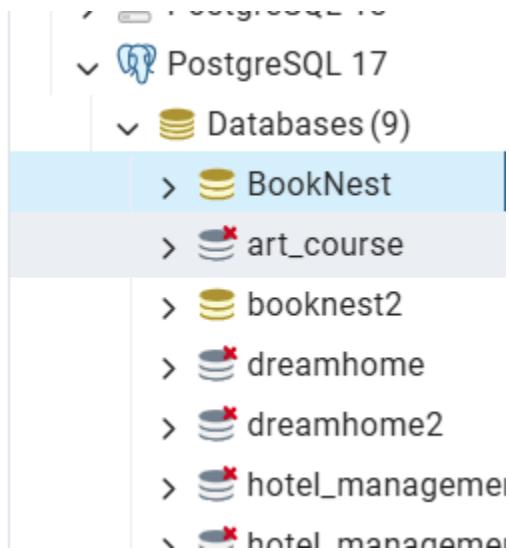


Name: Min Thant Hein

ID: PIUS20230001

Creating a database named “BookNest”:



Part A and Part B

Member:

-- 1) Member

```
CREATE TABLE member (
    member_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20) NOT NULL,
    join_date DATE DEFAULT CURRENT_DATE,
    membership_type VARCHAR(10) CHECK (membership_type IN ('Regular', 'Premium'))
);
```

Note: to write Null!

The screenshot shows the pgAdmin 4 interface for PostgreSQL 13. The left sidebar displays the database structure under 'PostgreSQL 17' with 'Databases (10)' and 'Schemas (1)'. The 'public' schema is selected, showing various objects like Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables. The 'member' table is currently selected. The main panel contains a query editor with the following SQL code:

```
1 CREATE TABLE member (
2     member_id INT PRIMARY KEY,
3     first_name VARCHAR(50) NOT NULL,
4     last_name VARCHAR(50) NOT NULL,
5     email VARCHAR(100) UNIQUE,
6     phone_number VARCHAR(20) NOT NULL,
7     join_date DATE DEFAULT CURRENT_DATE,
8     membership_type VARCHAR(10) CHECK (membership_type IN ('Regular', 'Premium'))
9 );
10
```

The 'Messages' tab in the results pane shows the output of the query:

```
CREATE TABLE
Query returned successfully in 172 msec.
```

-- 2) Publisher

```
CREATE TABLE publisher (
    publisher_id INT PRIMARY KEY,
    publisher_name VARCHAR(100) NOT NULL,
    country VARCHAR(50) DEFAULT 'Unknown'
);
```

Note: to add Null and test

The screenshot shows the pgAdmin 4 interface. On the left, the Object Browser tree view displays the database structure under 'PostgreSQL 17' > 'BookNest' > 'public' > 'Tables'. The 'publisher' table is selected, indicated by a blue border. On the right, the main window contains a 'Query' tab with the SQL code for creating the 'publisher' table. Below the query, the 'Messages' tab shows the execution results: 'CREATE TABLE' and 'Query returned successfully in 195 msec.'

```
CREATE TABLE publisher (
    publisher_id INT PRIMARY KEY,
    publisher_name VARCHAR(100) NOT NULL,
    country VARCHAR(50) DEFAULT 'Unknown'
);
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 195 msec.

```
/*-- 3) Book*/  
  
CREATE TABLE book (  
  
    book_id INT PRIMARY KEY,  
  
    title VARCHAR(200) NOT NULL,  
  
    author VARCHAR(100) NOT NULL,  
  
    genre VARCHAR(30) NOT NULL,  
  
    price DECIMAL(6,2) CHECK (price > 0),  
  
    stock INT CHECK (stock >= 0),  
  
    publisher_id INT,  
  
    published_year INT,  
  
    FOREIGN KEY (publisher_id) REFERENCES publisher(publisher_id)  
  
    ON DELETE SET NULL  
  
);
```

Note: publisher_year INT is not sure necessary or not

The screenshot shows the pgAdmin 4 interface connected to a PostgreSQL 17 database. The left sidebar displays the schema tree under the 'Tables' node, with 'Tables(3)' highlighted. The main pane shows the following SQL code:

```
18 );
19
20
21 /*-- 3) Book*/
22 CREATE TABLE book (
23     book_id INT PRIMARY KEY,
24     title VARCHAR(200) NOT NULL,
25     author VARCHAR(100) NOT NULL,
26     genre VARCHAR(30) NOT NULL,
27     price DECIMAL(6,2) CHECK (price > 0),
28     stock INT CHECK (stock >= 0),
29     publisher_id INT,
30     published_year INT,
31     FOREIGN KEY (publisher_id) REFERENCES publisher(publisher_id)
32         ON DELETE SET NULL
33 );
34
35
36 -- 4) Rental
37 CREATE TABLE rental (
38     rental_id INT PRIMARY KEY,
```

The 'Data Output' tab at the bottom shows the result of the last query: 'CREATE TABLE'. The message bar indicates the query was successful in 192 msec.

-- 4) Rental

```
CREATE TABLE rental (
    rental_id INT PRIMARY KEY,
    member_id INT,
    book_id INT,
    rental_date DATE,
    return_date DATE,
    status VARCHAR(10) CHECK (status IN ('Borrowed', 'Returned')),
    FOREIGN KEY (member_id) REFERENCES member(member_id)
        ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES book(book_id)
        ON DELETE SET NULL,
    CONSTRAINT chk_return_after_rental CHECK (return_date > rental_date),
    CONSTRAINT unique_rental_per_day UNIQUE (member_id, book_id, rental_date)
);
```

Note: to add Null, and to check foreign key constraints

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

Left Sidebar:

- Foreign Data Mappers
- Languages
- Publications
- Schemas (1)
 - public
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Operators
 - Procedures
 - Sequences
 - Tables (3)
 - book
 - member
 - publisher
 - Trigger Functions
 - Types
 - Views
- Subscriptions

Query Editor:

Query History

```
33 );  
34  
35  
36 -- 4) Rental  
37 CREATE TABLE rental (  
38     rental_id INT PRIMARY KEY,  
39     member_id INT NOT NULL,  
40     book_id INT NOT NULL,  
41     rental_date DATE,  
42     return_date DATE,  
43     status VARCHAR(10) CHECK (status IN ('Borrowed', 'Returned')),  
44     FOREIGN KEY (member_id) REFERENCES member(member_id)  
        ON DELETE CASCADE,  
45     FOREIGN KEY (book_id) REFERENCES book(book_id)  
        ON DELETE SET NULL,  
46     CONSTRAINT chk_return_after_rental CHECK (return_date > rental_date),  
47     CONSTRAINT unique_rental_per_day UNIQUE (member_id, book_id, rental_date)  
48 );  
49  
50  
51  
52  
53 -- 5) Payment  
54 CREATE TABLE payment (
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 123 msec.

-- 5) Payment

```
CREATE TABLE payment (
    payment_id INT PRIMARY KEY,
    rental_id INT,
    amount DECIMAL(6,2) CHECK (amount > 0),
    payment_date DATE,
    FOREIGN KEY (rental_id) REFERENCES rental(rental_id)
        ON DELETE CASCADE
);
```

Note: to add null and to check constraint foreign, table constraint is correct

The screenshot shows the pgAdmin 4 interface. On the left, the schema browser displays the database structure with nodes like Extensions, Foreign Data Wrappers, Languages, Publications, Schemas (1), public, and Tables (5). The Tables node is currently selected. The main window contains a query editor with the following code:

```
46     FOREIGN KEY (book_id) REFERENCES book(book_id)
47         ON DELETE SET NULL,
48     CONSTRAINT chk_return_after_rental CHECK (return_date > rental_date),
49     CONSTRAINT unique_rental_per_day UNIQUE (member_id, book_id, rental_date)
50 );
51
52
53 -- 5) Payment
54 CREATE TABLE payment (
55     payment_id INT PRIMARY KEY,
56     rental_id INT,
57     amount DECIMAL(6,2) CHECK (amount > 0),
58     payment_date DATE,
59     FOREIGN KEY (rental_id) REFERENCES rental(rental_id)
60         ON DELETE CASCADE
61 );
62
63
64 -- 6) Review
65 CREATE TABLE review (
66     review_id INT PRIMARY KEY,
67     member_id INT,
68     book_id INT,
```

The code for the payment table includes a primary key, an integer foreign key referencing the rental_id in the rental table, a decimal column for amount with a check constraint ensuring it's greater than zero, and a date column for payment_date. It also includes an ON DELETE CASCADE clause for the foreign key constraint. The review table is partially defined with a primary key and two integer foreign keys.

-- 6) Review

```
CREATE TABLE review (
    review_id INT PRIMARY KEY,
    member_id INT,
    book_id INT,
    rating INT CHECK (rating BETWEEN 1 AND 5),
    review_text TEXT,
    FOREIGN KEY (member_id) REFERENCES member(member_id)
        ON DELETE CASCADE,
    FOREIGN KEY (book_id) REFERENCES book(book_id)
        ON DELETE CASCADE,
    CONSTRAINT unique_review_per_member_book UNIQUE (member_id, book_id)
);
```

Note: correct columns, to add Null, and to check foreign constraints

The screenshot shows the pgAdmin interface for PostgreSQL. On the left, a tree view displays database objects under the 'public' schema:

- Foreign Data Wrappers
- Languages
- Publications
- Schemas (1)
 - public
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Operators
 - Procedures
 - Sequences
 - Tables (6)
 - book
 - member
 - payment
 - publisher
 - rental
 - review
 - Trigger Functions

The main window contains two tabs:

- Query**: Displays two SQL scripts. The first script creates a 'rental' table with columns: rental_id (INT), amount (DECIMAL(6,2) CHECK(amount > 0)), and payment_date (DATE). It includes a FOREIGN KEY constraint referencing the 'rental' table's rental_id column with ON DELETE CASCADE. The second script creates a 'review' table with columns: review_id (INT PRIMARY KEY), member_id (INT), book_id (INT), rating (INT CHECK(rating BETWEEN 1 AND 5)), and review_text (TEXT). It includes FOREIGN KEY constraints referencing the 'member' and 'book' tables with ON DELETE CASCADE, and a UNIQUE constraint named unique_review_per_member_book.
- Data Output**: Shows the CREATE TABLE command and a message indicating the query was successful.

Below the tabs, there are three buttons: Data Output, Messages (which is selected), and Notifications.

```
56 rental_id INT,  
57     amount DECIMAL(6,2) CHECK (amount > 0),  
58     payment_date DATE,  
59     FOREIGN KEY (rental_id) REFERENCES rental(rental_id)  
       ON DELETE CASCADE  
60 );  
61  
62 -- 6) Review  
63 CREATE TABLE review (  
64     review_id INT PRIMARY KEY,  
65     member_id INT,  
66     book_id INT,  
67     rating INT CHECK (rating BETWEEN 1 AND 5),  
68     review_text TEXT,  
69     FOREIGN KEY (member_id) REFERENCES member(member_id)  
       ON DELETE CASCADE,  
70     FOREIGN KEY (book_id) REFERENCES book(book_id)  
       ON DELETE CASCADE,  
71     CONSTRAINT unique_review_per_member_book UNIQUE (member_id, book_id)  
72 );  
73  
74  
75  
76  
77
```

CREATE TABLE

Query returned successfully in 49 msec.

Part C

-- a) Add a column published_year INT to Book.

ALTER TABLE Book

ADD COLUMN published_year INT;

-- b) Remove phone_number from Member.

ALTER TABLE Member

DROP COLUMN phone_number;

-- c) Change country in Publisher to default to "Unknown".

ALTER TABLE Publisher

ALTER COLUMN country SET DEFAULT 'Unknown';

-- d) Make join_date in Member default to today's date.

ALTER TABLE Member

ALTER COLUMN join_date SET DEFAULT CURRENT_DATE;

Note: Do I need to make changes? (since I included some as NOT NULL)

```

1 -- a) Add a column published_year INT to Book.
2 ALTER TABLE Book
3   ADD COLUMN published_year INT;
4
5 -- b) Remove phone_number from Member.
6 ALTER TABLE Member
7   DROP COLUMN phone_number;
8
9 -- c) Change country in Publisher to default to "Unknown".
10 ALTER TABLE Publisher
11   ALTER COLUMN country SET DEFAULT 'Unknown';
12
13 -- d) Make join_date in Member default to today's date.
14 ALTER TABLE Member
15   ALTER COLUMN join_date SET DEFAULT CURRENT_DATE;
16
17
18

```

The screenshot shows the pgAdmin 4 interface. On the left, the database browser displays the schema of the 'BookNest' database, including various objects like Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications, Schemas, and Tables. The 'Tables' node is expanded, showing the 'book' and 'member' tables. On the right, the query editor window has a title bar 'BookNest/postgres@PostgreSQL 17'. The toolbar includes standard database management icons. Below the toolbar, tabs for 'Query' and 'Query History' are visible, with 'Query' selected. The main area of the query editor contains a multi-line SQL script. The script consists of several numbered comments (1 through 18) preceding specific SQL statements. Comments 1, 4, and 12 are Alter Table statements. Comment 8 is a Drop Column statement. Comments 11 and 15 are Alter Column statements setting defaults. The script ends with two blank lines. Below the script, there are tabs for 'Data Output', 'Messages', and 'Notifications', with 'Messages' being the active tab. The message content shows the results of the executed queries, indicating they were successful.

Note: Since I misunderstood and I applied certain concepts. I only require to run from number b to number c.

##

Inserting Data:

-- Insert some publishers

```
INSERT INTO publisher (publisher_id, publisher_name, country)
```

```
VALUES
```

```
(1, 'TechPress', 'USA'),
```

```
(2, 'EduBooks', 'UK'),
```

```
(3, 'Global Reads', 'Thailand');
```

-- Insert members with different membership types

```
INSERT INTO member (member_id, first_name, last_name, email, phone_number, membership_type)
```

```
VALUES
```

```
(101, 'Alice', 'Wong', 'alice@example.com', '0812345678', 'Regular'),
```

```
(102, 'Bob', 'Smith', 'bob@example.com', '0898765432', 'Premium'),
```

```
(103, 'Charlie', 'Ng', 'charlie@example.com', NULL, 'Regular');
```

-- Insert books linked to publishers

```
INSERT INTO book (book_id, title, author, genre, price, stock, publisher_id)
```

```
VALUES
```

```
(201, 'Database Systems', 'C. J. Date', 'Science', 50.00, 3, 1),
```

```
(202, 'Machine Learning Basics', 'Andrew Ng', 'Science', 60.00, 2, 1),
```

```
(203, 'Storytelling for Data', 'Cole N. Knaflic', 'Non-Fiction', 45.00, 5, 2),
```

```
(204, 'Children of Time', 'Adrian Tchaikovsky', 'Fiction', 30.00, 4, 3);
```

-- Insert rentals for members

```
INSERT INTO rental (rental_id, member_id, book_id, rental_date, return_date, status)
```

```
VALUES
```

```
(301, 101, 201, DATE '2025-09-01', DATE '2025-09-07', 'Borrowed'),
```

```
(302, 102, 202, DATE '2025-09-02', DATE '2025-09-10', 'Borrowed'),
```

```
(303, 101, 203, DATE '2025-09-05', DATE '2025-09-12', 'Returned');
```

-- Insert payments related to rentals

```
INSERT INTO payment (payment_id, rental_id, amount, payment_date)
```

```
VALUES
```

```
(401, 303, 10.00, DATE '2025-09-06'),
```

```
(402, 302, 15.00, DATE '2025-09-03');
```

-- Insert reviews for books

```
INSERT INTO review (review_id, member_id, book_id, rating, review_text)
```

```
VALUES
```

```
(501, 101, 201, 5, 'Very informative book, highly recommended!'),
```

```
(502, 102, 203, 4, 'Great insights on data storytelling.'),
```

```
(503, 101, 204, 3, 'Interesting but a bit long.');
```

##

There is an error related to phone_number!

Here is the error,

```
ERROR: column "phone_number" of relation "member" does not exist
LINE 9: ... member (member_id, first_name, last_name, email, phone_numb...
^
```

SQL state: 42703

Character: 301

Thus, I add the phone_number column again as a solution!

The screenshot shows the pgAdmin 4 interface. On the left, the 'Servers' tree view is expanded to show 'PostgreSQL 13' and 'PostgreSQL 17'. Under 'PostgreSQL 17', 'Databases' is expanded, showing 'BookNest' which contains 'Casts', 'Catalogs', 'Event Triggers', 'Extensions', 'Foreign Data Wrappers', 'Languages', 'Publications', and 'Schemas (1)'. The 'public' schema is selected. Inside 'public', there are 'Aggregates', 'Collations', 'Domains', 'FTS Configurations', 'FTS Dictionaries', 'FTS Parsers', 'FTS Templates', and 'Foreign Tables'. The main pane shows a query editor with the following content:

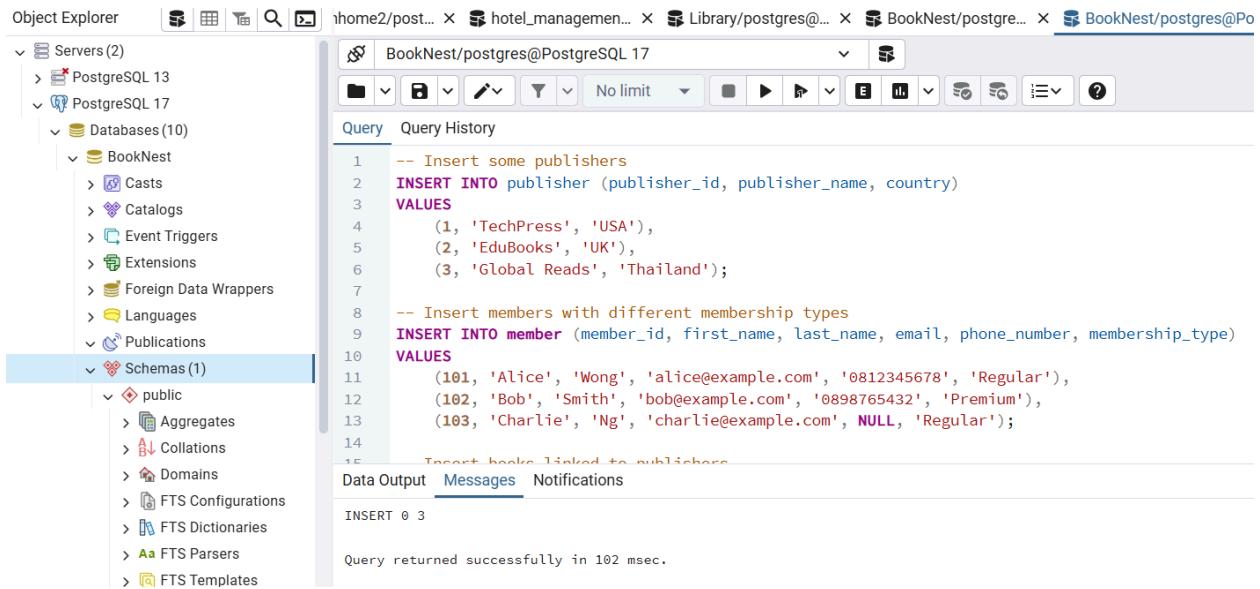
```
ALTER TABLE member ADD COLUMN phone_number VARCHAR(20);
```

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

```
ALTER TABLE
Query returned successfully in 59 msec.
```

##

Thus, inserting data run successfully!



The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer tree view is expanded to show 'Servers' (2), 'PostgreSQL 13', 'PostgreSQL 17', 'Databases' (10), 'BookNest' (with sub-items: Casts, Catalogs, Event Triggers, Extensions, Foreign Data Wrappers, Languages, Publications), and 'Schemas (1)' (with sub-item: public). The 'public' schema is selected. On the right, the main window displays a query editor titled 'BookNest/postgres@PostgreSQL 17'. The query is:

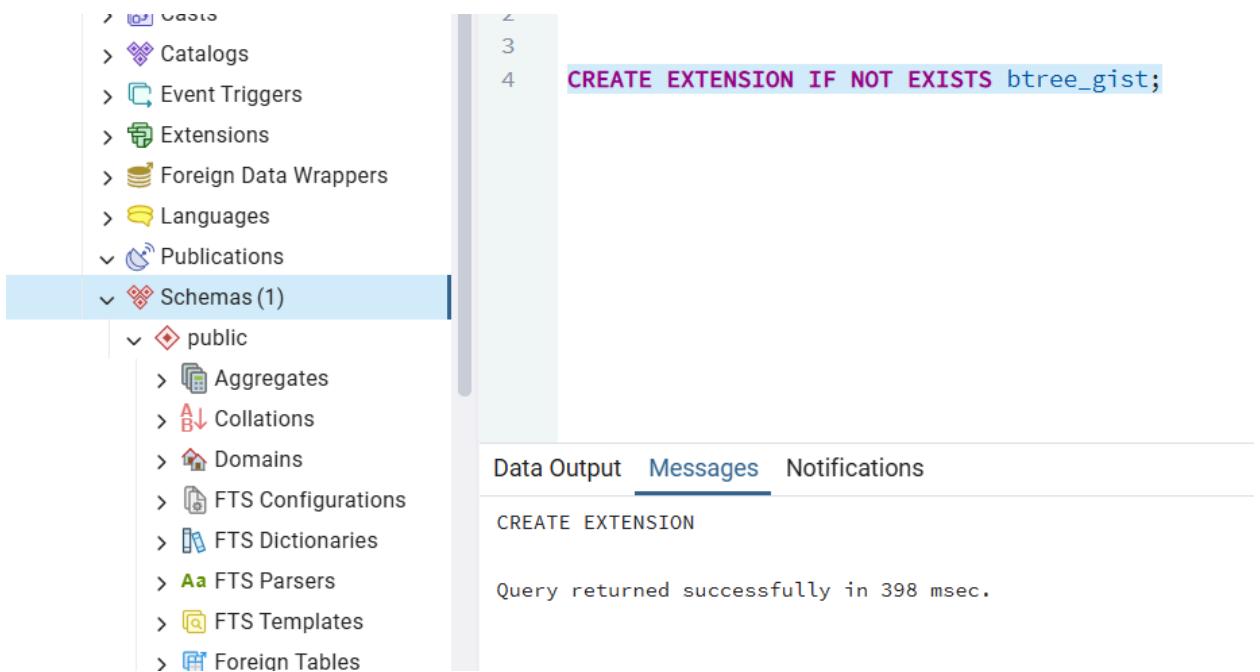
```
-- Insert some publishers
INSERT INTO publisher (publisher_id, publisher_name, country)
VALUES
    (1, 'TechPress', 'USA'),
    (2, 'EduBooks', 'UK'),
    (3, 'Global Reads', 'Thailand');

-- Insert members with different membership types
INSERT INTO member (member_id, first_name, last_name, email, phone_number, membership_type)
VALUES
    (101, 'Alice', 'Wong', 'alice@example.com', '0812345678', 'Regular'),
    (102, 'Bob', 'Smith', 'bob@example.com', '0898765432', 'Premium'),
    (103, 'Charlie', 'Ng', 'charlie@example.com', NULL, 'Regular');
```

The 'Messages' tab in the results pane shows 'Query returned successfully in 102 msec.'

##

Importing an extension



The screenshot shows the pgAdmin 4 interface. The Object Explorer tree view is expanded to show 'Schemas (1)' (with sub-item: public) under 'BookNest'. The 'public' schema is selected. On the right, the main window displays a query editor titled 'BookNest/postgres@PostgreSQL 17'. The query is:

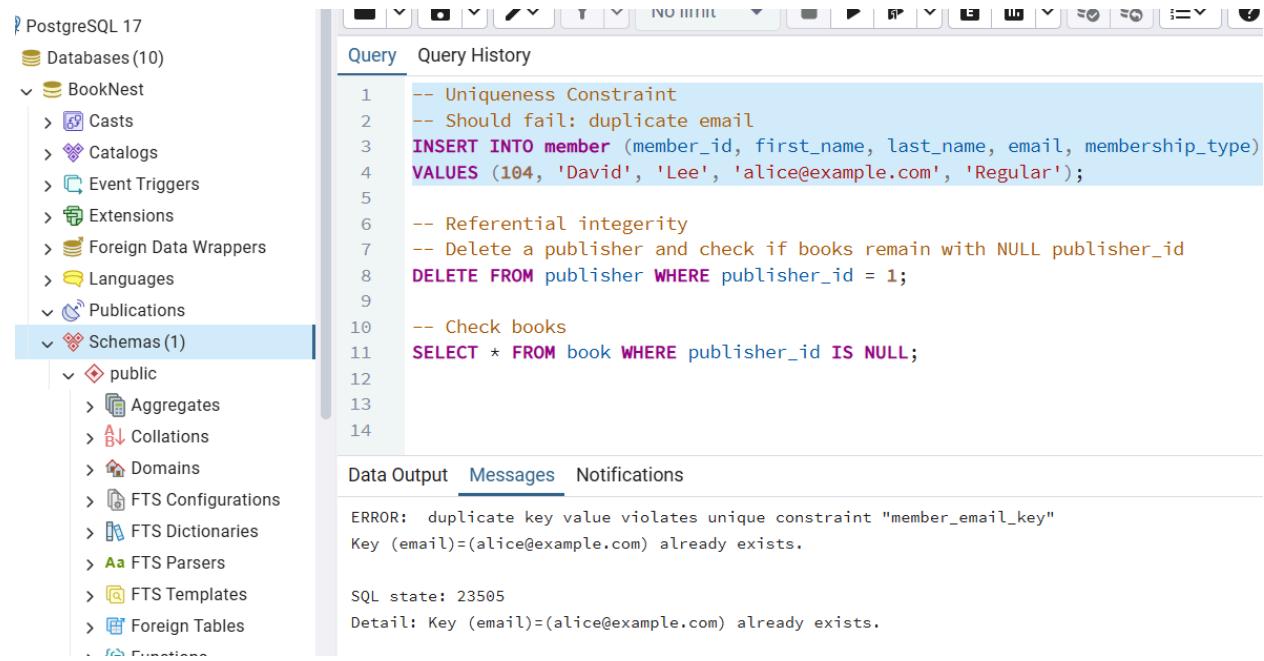
```
CREATE EXTENSION IF NOT EXISTS btree_gist;
```

The 'Messages' tab in the results pane shows 'CREATE EXTENSION' and 'Query returned successfully in 398 msec.'

To avoid double booking, I created an extension, “btree_gist”. Actually, the query lines are already completed. However, there is still a condition called double booking

Suggested Student Tasks with This Data

Check uniqueness: Try inserting a member with a duplicate email → should fail.



The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under 'BookNest'. Under 'Schemas', the 'public' schema is selected. In the center is a query editor window with the following content:

```
1 -- Uniqueness Constraint
2 -- Should fail: duplicate email
3 INSERT INTO member (member_id, first_name, last_name, email, membership_type)
4 VALUES (104, 'David', 'Lee', 'alice@example.com', 'Regular');
5
6 -- Referential integrity
7 -- Delete a publisher and check if books remain with NULL publisher_id
8 DELETE FROM publisher WHERE publisher_id = 1;
9
10 -- Check books
11 SELECT * FROM book WHERE publisher_id IS NULL;
```

Below the query editor is a 'Messages' tab showing the error output:

```
ERROR: duplicate key value violates unique constraint "member_email_key"
Key (email)=(alice@example.com) already exists.

SQL state: 23505
Detail: Key (email)=(alice@example.com) already exists.
```

#

Test referential integrity:

- Delete a publisher → check if `publisher_id` in related books becomes NULL or default as per their design.
- Delete a member → verify that rentals and reviews are automatically deleted.

PostgreSQL 17

Databases (10)

- BookNest
 - Casts
 - Catalogs
 - Event Triggers
 - Extensions
 - Foreign Data Wrappers
 - Languages
 - Publications
- Schemas (1)
 - public
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions

Query Query History

```

1 -- 1.Uniqueness Constraint
2 -- Should fail: duplicate email
3 INSERT INTO member (member_id, first_name, last_name, email, membership_type)
4 VALUES (104, 'David', 'Lee', 'alice@example.com', 'Regular');
5
6 -- 2.Referential integrity
7 -- Delete a publisher and check if books remain with NULL publisher_id
8 DELETE FROM publisher WHERE publisher_id = 1;
9
10 -- Check books
11 SELECT * FROM book WHERE publisher_id IS NULL;
12
13 -- 3.Cascading Deletes
14 -- Delete a member and check if rentals and reviews are deleted
15 DELETE FROM member WHERE member_id = 101;

```

Data Output Messages Notifications

	book_id [PK] integer	title character varying (200)	author character varying (100)	genre character varying (30)	price numeric (6,2)	stock integer	publisher_id integer	published_year integer
1	201	Database Systems	C. J. Date	Science	50.00	3	[null]	[null]
2	202	Machine Learning Basics	Andrew Ng	Science	60.00	2	[null]	[null]

Showing rows: 1 to 2 Page No: 1

#

> Casts

> Catalogs

> Event Triggers

> Extensions

> Foreign Data Wrappers

> Languages

Publications

Schemas (1)

- public
 - Aggregates
 - Collations
 - Domains
 - FTS Configurations
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates

8 DELETE FROM publisher WHERE publisher_id = 1;

9

10 -- Check books

11 SELECT * FROM book WHERE publisher_id IS NULL;

12

13 -- 3.Cascading Deletes

14 -- Delete a member and check if rentals and reviews are deleted

15 DELETE FROM member WHERE member_id = 101;

16

17 -- Check rentals and reviews

18 SELECT * FROM rental WHERE member_id = 101;

19 SELECT * FROM review WHERE member_id = 101;

20

21

Data Output Messages Notifications

	review_id [PK] integer	member_id integer	book_id integer	rating integer	review_text text
--	------------------------	-------------------	-----------------	----------------	------------------

#

##

Test constraints:

- Try inserting a book with negative price or stock → should fail.
- Try inserting a rental where `return_date <= rental_date` → should fail.
- Try renting the same book twice for the same date → should fail.

```
18 SELECT * FROM rental WHERE member_id = 101;
19
20
21 -- 4.Constraint Violations
22 -- Should fail: negative price
23 INSERT INTO book (book_id, title, author, genre, price, stock, publisher_id)
24 VALUES (205, 'Bad Book', 'Author X', 'Fiction', -10.00, 1, 2);
25
26 -- Should fail: negative stock
27 INSERT INTO book (book_id, title, author, genre, price, stock, publisher_id)
28 VALUES (206, 'Bad Stock', 'Author Y', 'Fiction', 20.00, -5, 2);
29
30 -- Should fail: return_date <= rental_date
31 INSERT INTO rental (rental_id, member_id, book_id, rental_date, return_date, status)
32 VALUES (304, 102, 203, DATE '2025-09-10', DATE '2025-09-09', 'Borrowed');
```

Data Output Messages Notifications

ERROR: new row for relation "book" violates check constraint "book_price_check"
Failing row contains (205, Bad Book, Author X, Fiction, -10.00, 1, 2, null).

SQL state: 23514
Detail: Failing row contains (205, Bad Book, Author X, Fiction, -10.00, 1, 2, null).

#

Test cascading deletes: Delete a rental and confirm its payment is deleted automatically.

The screenshot shows the pgAdmin 4 interface. On the left, the database tree is displayed under 'PostgreSQL 17' > 'Databases (10)' > 'BookNest'. Under 'Schemas (1)', the 'public' schema is selected. The right side features a query editor with the following SQL code:

```
-- Should fail: same book rented twice on same date
INSERT INTO rental (rental_id, member_id, book_id, r
VALUES (304, 102, 203, DATE '2025-09-10', DATE '2025-09-10');

-- Should fail: same book rented twice on same date
INSERT INTO rental (rental_id, member_id, book_id, r
VALUES (305, 102, 202, DATE '2025-09-02', DATE '2025-09-02');

--5.Cascading Delete for Payment
-- Delete rental and check if payment is deleted
DELETE FROM rental WHERE rental_id = 302;

-- Check payment
SELECT * FROM payment WHERE rental_id = 302;
```

Below the query editor is a 'Data Output' tab showing the structure of the 'payment' table:

	payment_id [PK] integer	rental_id integer	amount numeric (6,2)	payment_date date
--	----------------------------	----------------------	-------------------------	----------------------

```
#  
##
```

Test scenarios (write sample statements to demonstrate behavior)

- Deleting a **Member** with existing rentals, payments, and reviews—show what remains.
- Deleting a **Publisher** with existing books—show how books are affected.
- Deleting a **Book** with past rentals and reviews—show what remains/changes.
- Attempting:
 - a negative book price or stock,
 - a rental where `return_date ≤ rental_date`,
 - two rentals of the same book that overlap,
 - a duplicate review by the same member for the same book,
 - exceeding rental limits for Regular vs Premium.

The screenshot shows a database interface with a sidebar on the left containing various system objects:

- > Extensions
- > Foreign Data Wrappers
- > Languages
- < Publications
- < Schemas(1)
 - < public
 - > Aggregates
 - > Collations
 - > Domains
 - > FTS Configurations
 - > FTS Dictionaries
 - > FTS Parsers
 - > FTS Templates
 - > Foreign Tables

The main panel displays the following SQL code:

```
39      DELETE rental and check if payment is deleted
40      DELETE FROM rental WHERE rental_id = 302;
41
42      -- Check payment
43      SELECT * FROM payment WHERE rental_id = 302;
44
45      --6.Duplicate Reviews
46      -- Should fail: same member reviewing same book again
47      INSERT INTO review (review_id, member_id, book_id, rating, review_text)
48      VALUES (504, 102, 203, 5, 'Another review');
```

Below the code, there are tabs for Data Output, Messages, and Notifications. The Messages tab shows the following error message:

ERROR: duplicate key value violates unique constraint "unique_review_per_member_book"
Key (member_id, book_id)=(102, 203) already exists.
SQL state: 23505
Detail: Key (member_id, book_id)=(102, 203) already exists.

##

