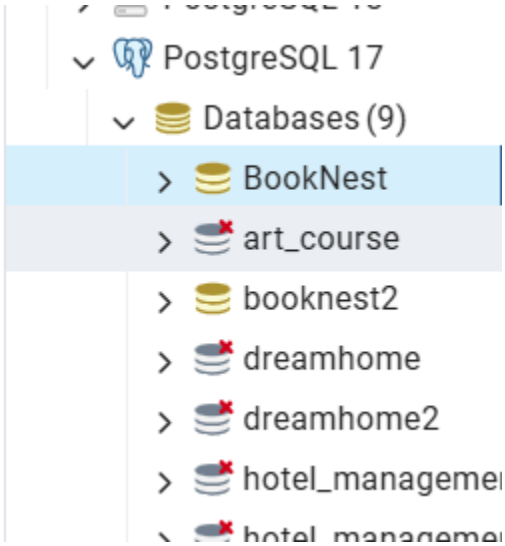
Name: Min Thant Hein
ID: PIUS20230001

Creating a database named “BookNest”:

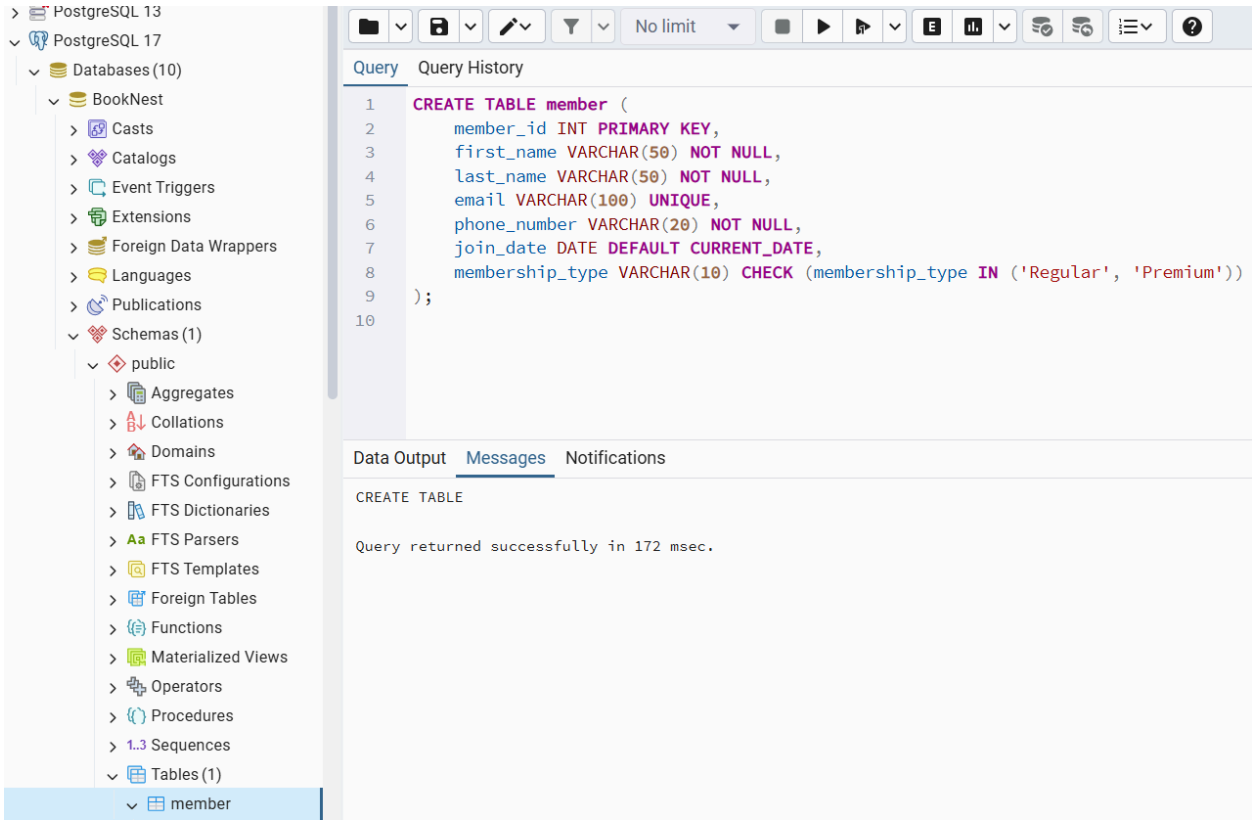


Part A and Part B

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'))
);
```

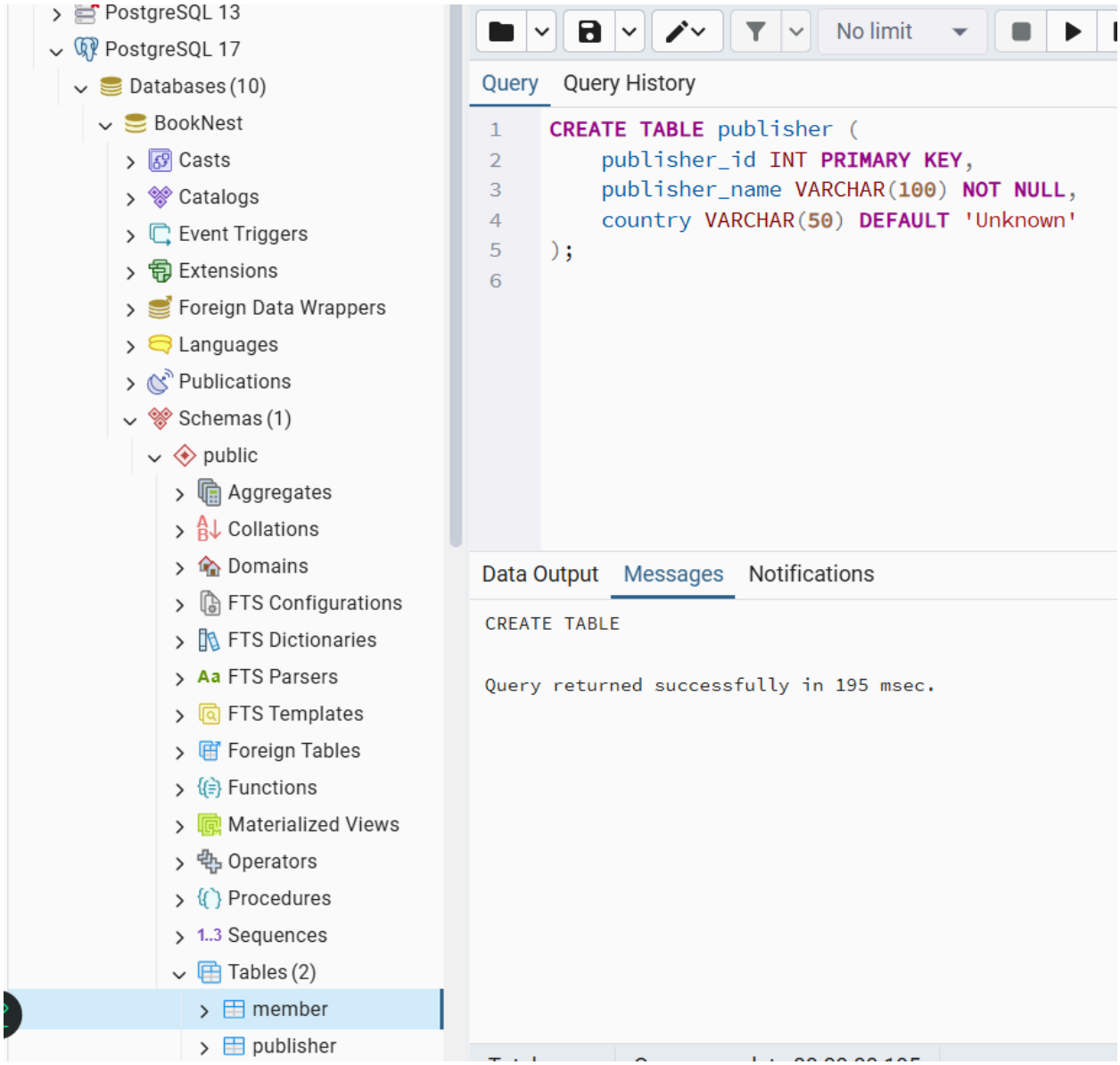
Note: to write Null!


PostgreSQL 13
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
Materialized Views
Operators
Procedures
Sequences
Tables (1)
member
No limit
Query
Query History
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'))
);
Data Output
Messages
Notifications
CREATE TABLE
Query returned successfully in 172 msec.

-- 2) Publisher

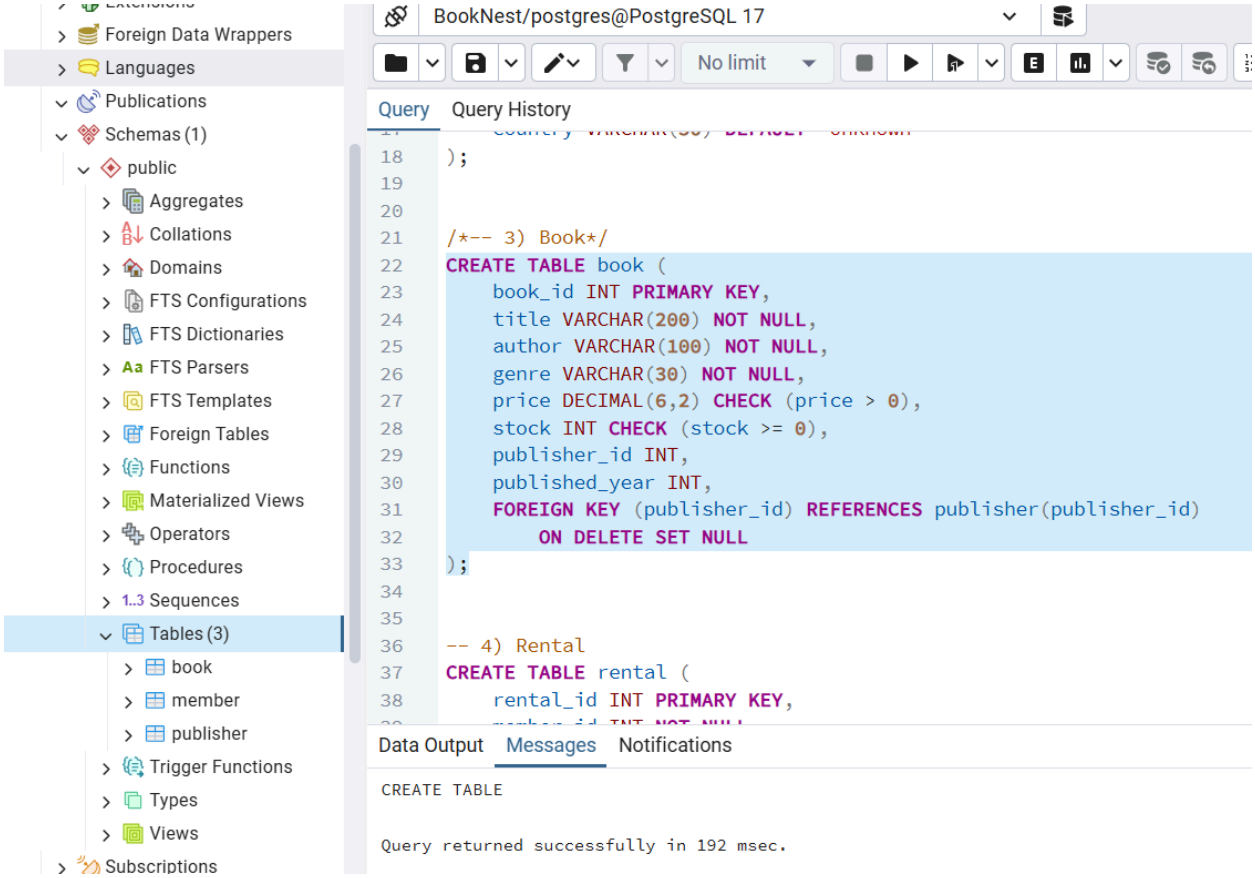
```
CREATE TABLE publisher (
    publisher_id INT PRIMARY KEY,
    publisher_name VARCHAR(100) NOT NULL,
    country VARCHAR(50) DEFAULT 'Unknown'
);
```

Note: to add Null and test



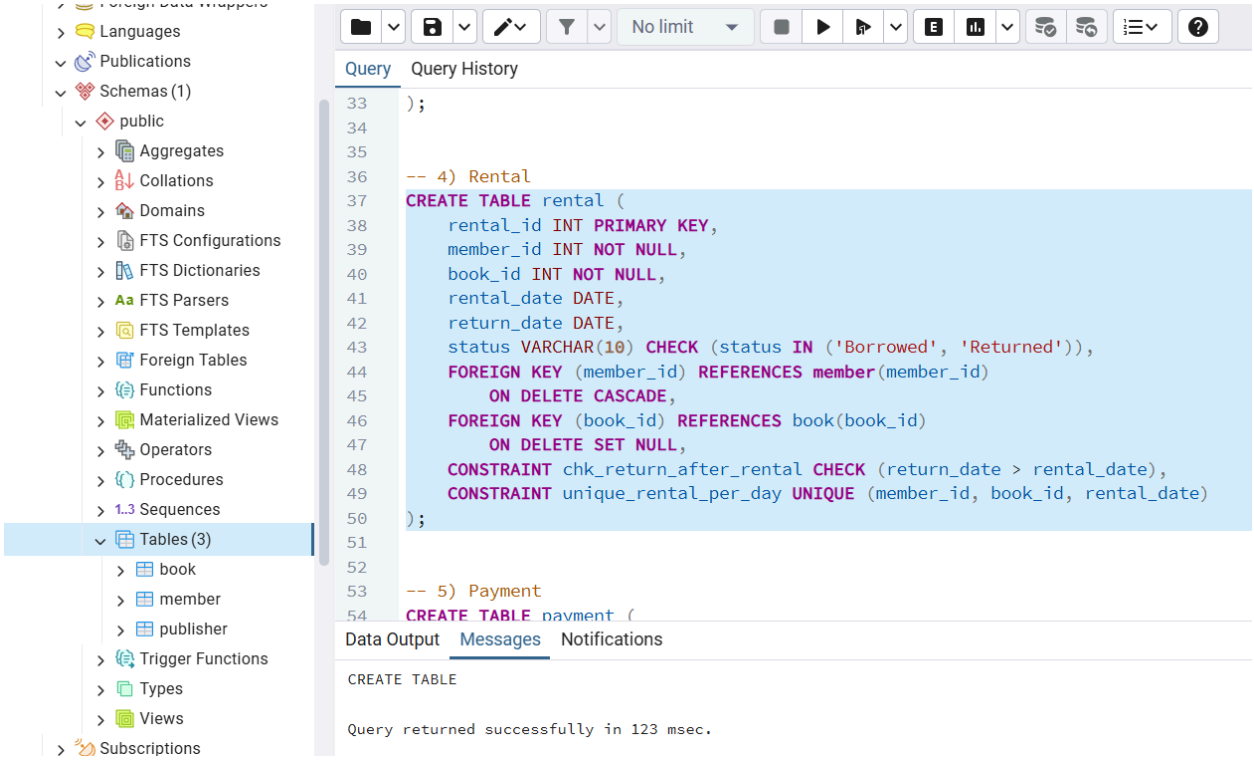
```
/*-- 3) Book*/
  CREATE TABLE book (
    book_id INT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    author VARCHAR(100) NOT NULL,
    genre VARCHAR(30) NOT NULL,
    price DECIMAL(6,2) CHECK (price > 0),
    stock INT CHECK (stock >= 0),
    publisher_id INT,
    published_year INT,
    FOREIGN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

Note: publisher_year INT is not sure necessary or not


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 (3)
book
member
publisher
Trigger Functions
Types
Views
Subscriptions
BookNest/postgres@PostgreSQL 17
No limit
Query
Query History
);
/*-- 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
);
-- 4) Rental
CREATE TABLE rental (
rental_id INT PRIMARY KEY,
Data Output
Messages
Notifications
CREATE TABLE
Query returned successfully 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)
);
```

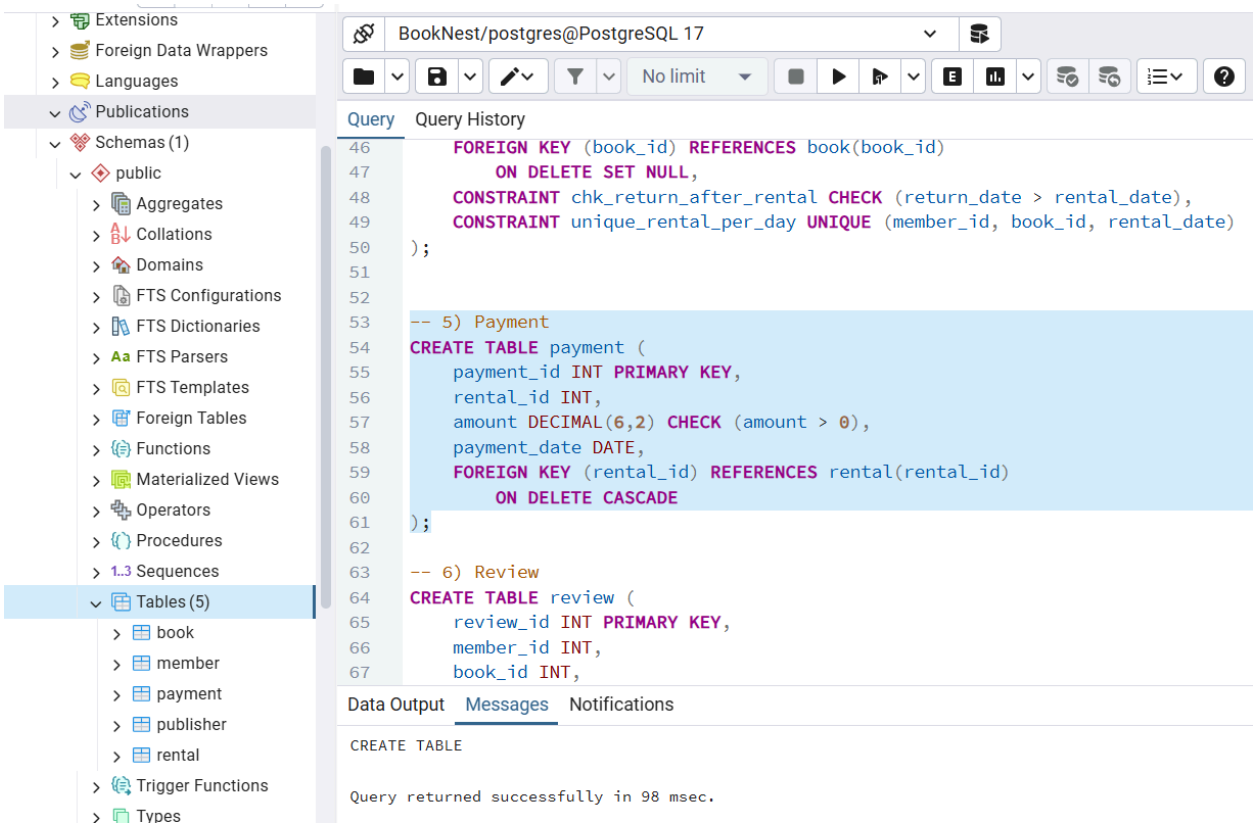
Note: to add Null, and to check foreign key constraints


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
No limit
Query
Query History
);
-- 4) Rental
CREATE TABLE rental (
rental_id INT PRIMARY KEY,
member_id INT NOT NULL,
book_id INT NOT NULL,
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)
);
-- 5) Payment
CREATE TABLE payment (
Data Output
Messages
Notifications
CREATE TABLE
Query returned successfully in 123 msec.

-- 5) Payment

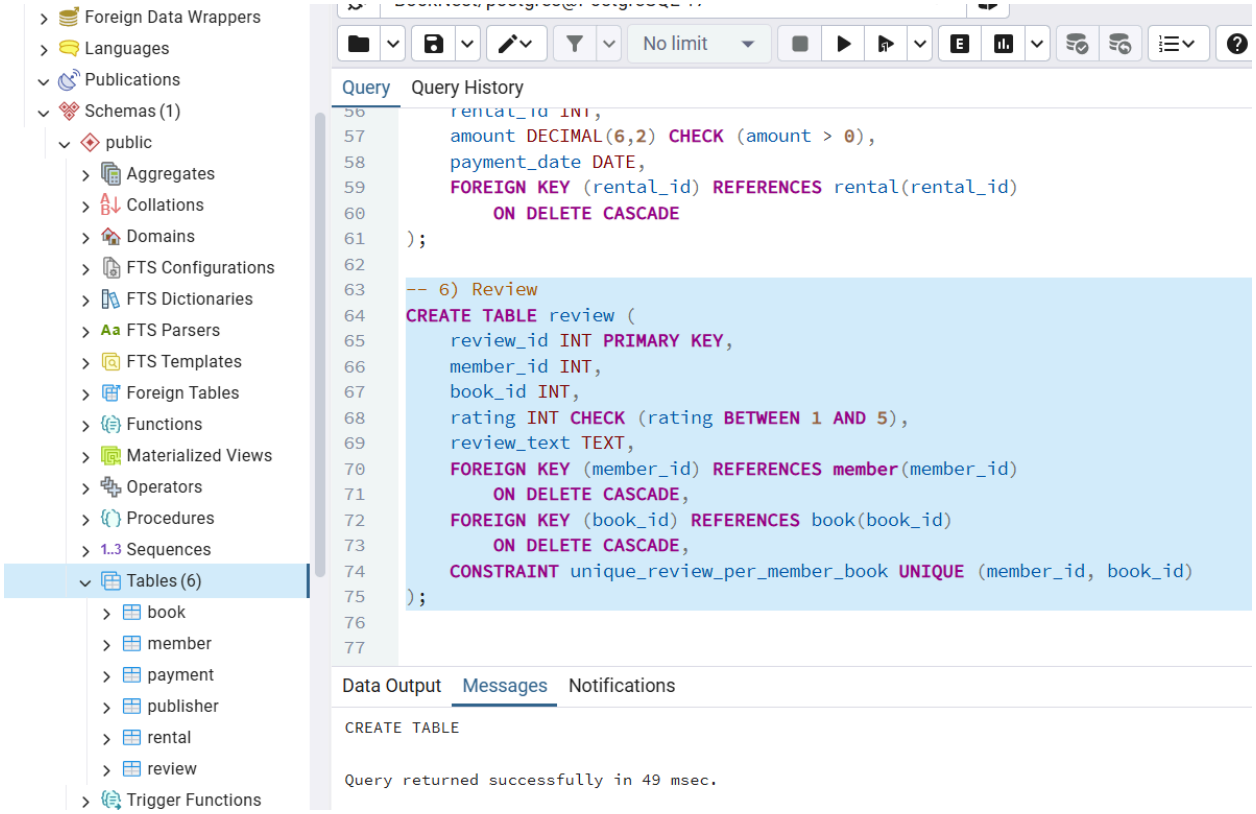
```
CREATE TABLE payment (
    payment_id INT PRIMARY KEY,
    rental_id INT,
    amount DECIMAL(6,2) CHECK (amount > 0),
    payment_date DATE,
    FOREIGN KEY (rental_id) REFERENCES rental(rental_id)
        ON DELETE CASCADE
);
```

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



```
-- 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


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
No limit
Query
Query History
rental_id INT,
amount DECIMAL(6,2) CHECK (amount > 0),
payment_date DATE,
FOREIGN KEY (rental_id) REFERENCES rental(rental_id)
ON DELETE CASCADE
);
-- 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)
);
Data Output
Messages
Notifications
CREATE TABLE
Query returned successfully in 49 msec.

Part C

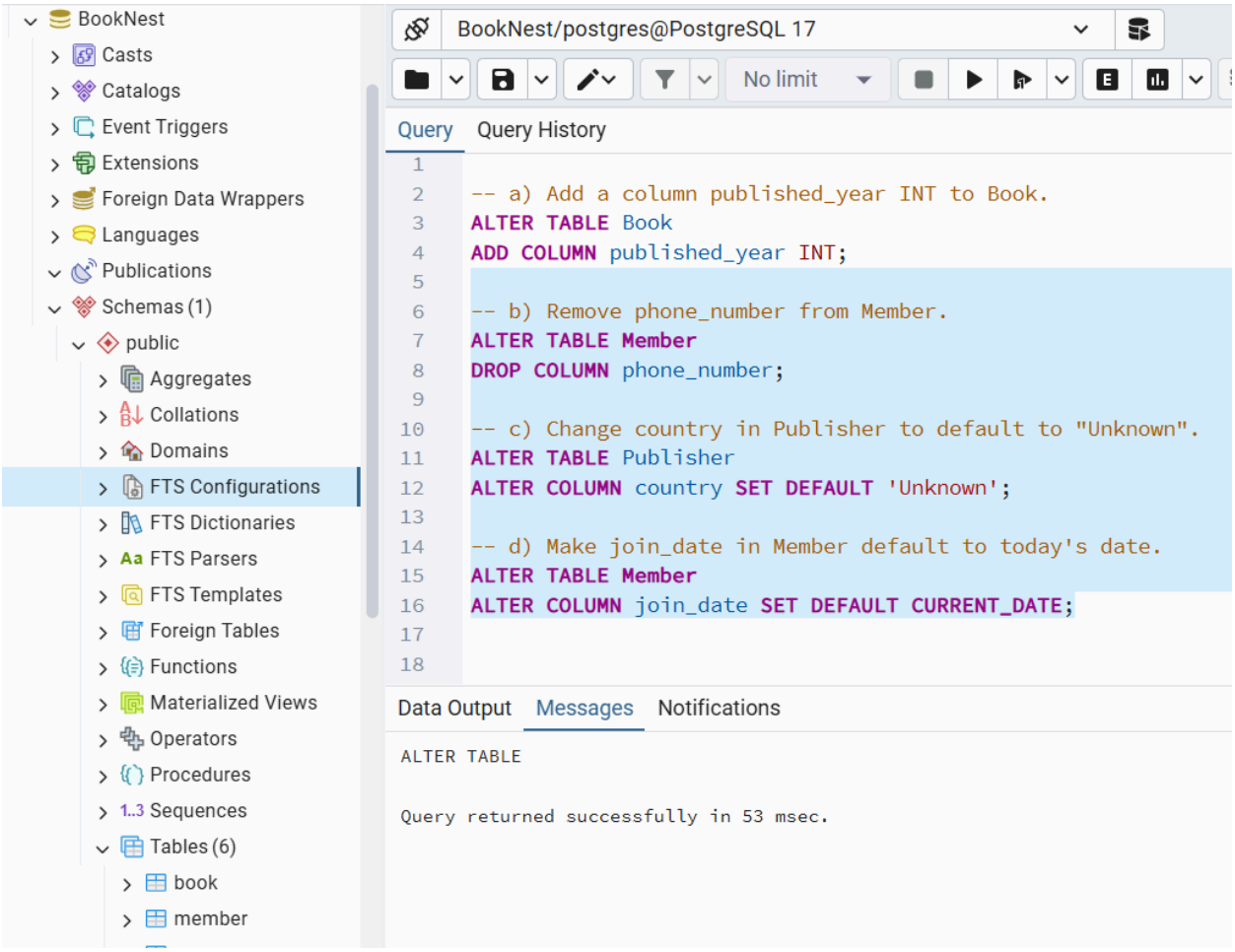
-- a) Add a column published_year INT to Book.
ALTER TABLE Book
ADD COLUMN published_year INT;

-- 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)



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.');

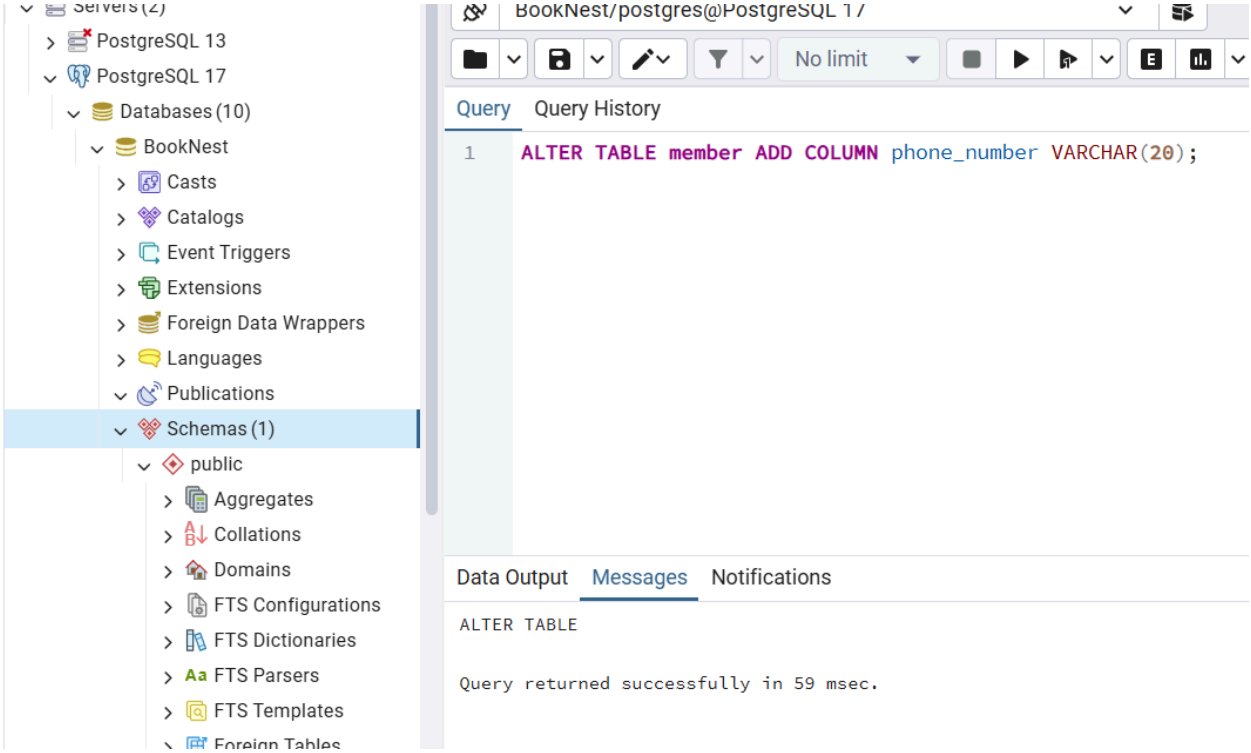
##

There is an error related to phone_number!

Here is the error,

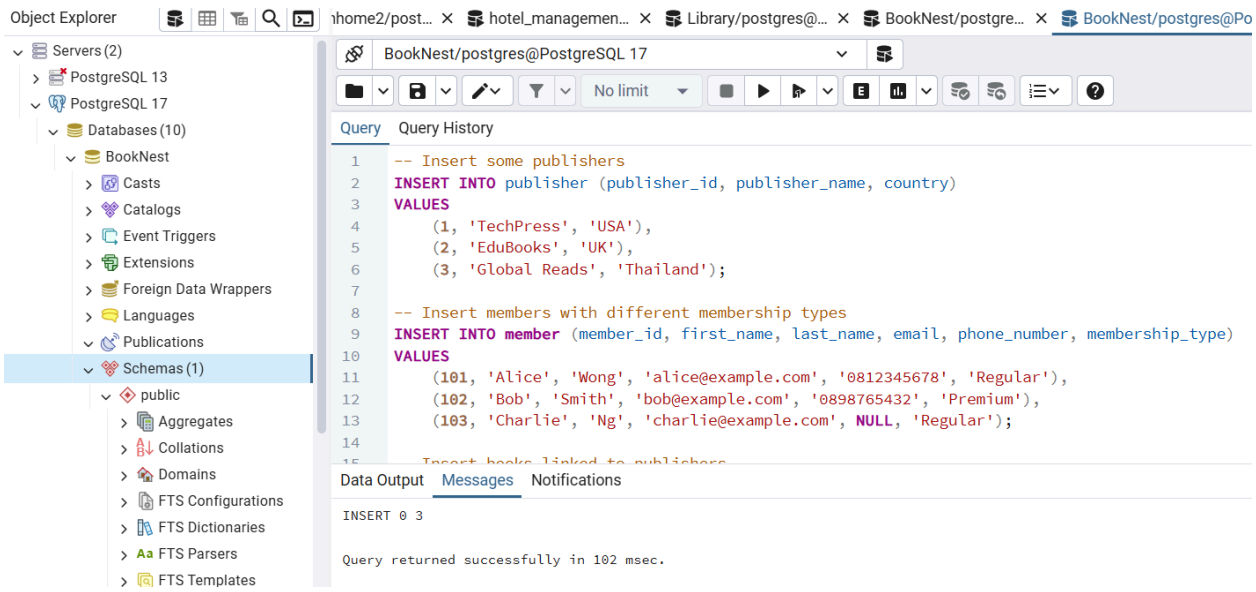
```
ERROR:  column "phone_number" of relation "member" does not exist
LINE 9: ... member (member_id, first_name, last_name, email, phone_numb...
                                                             ^

SQL state: 42703
Character: 301
```

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



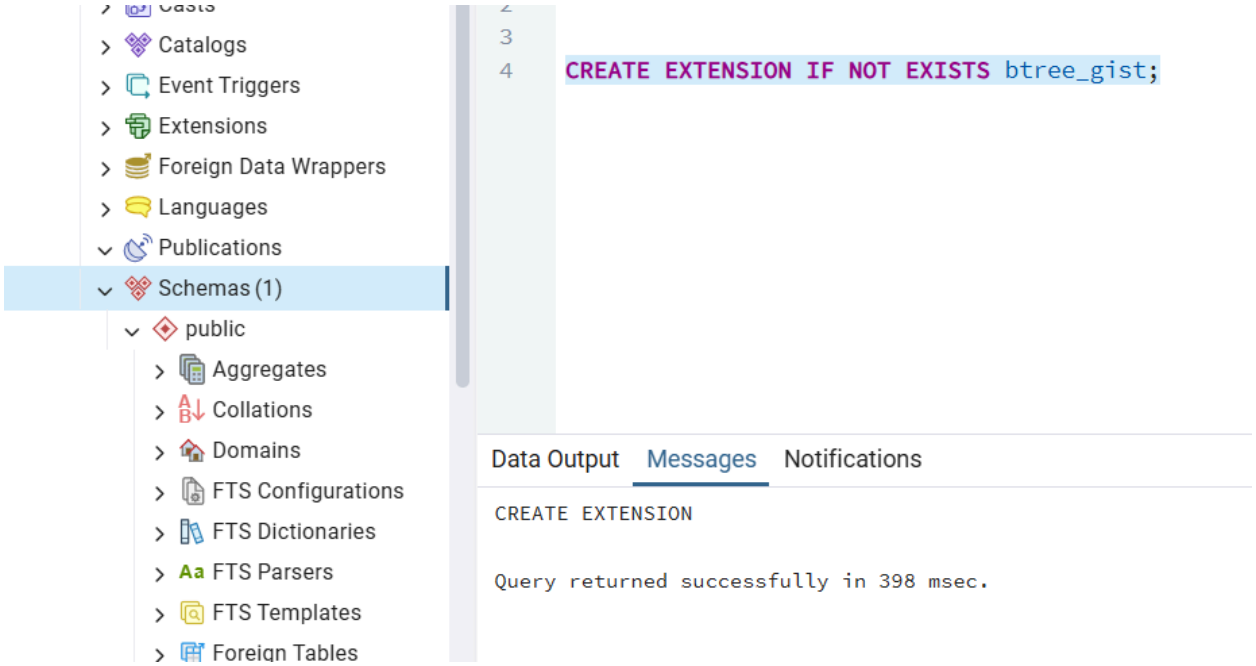
##

Thus, inserting data run successfully!



##

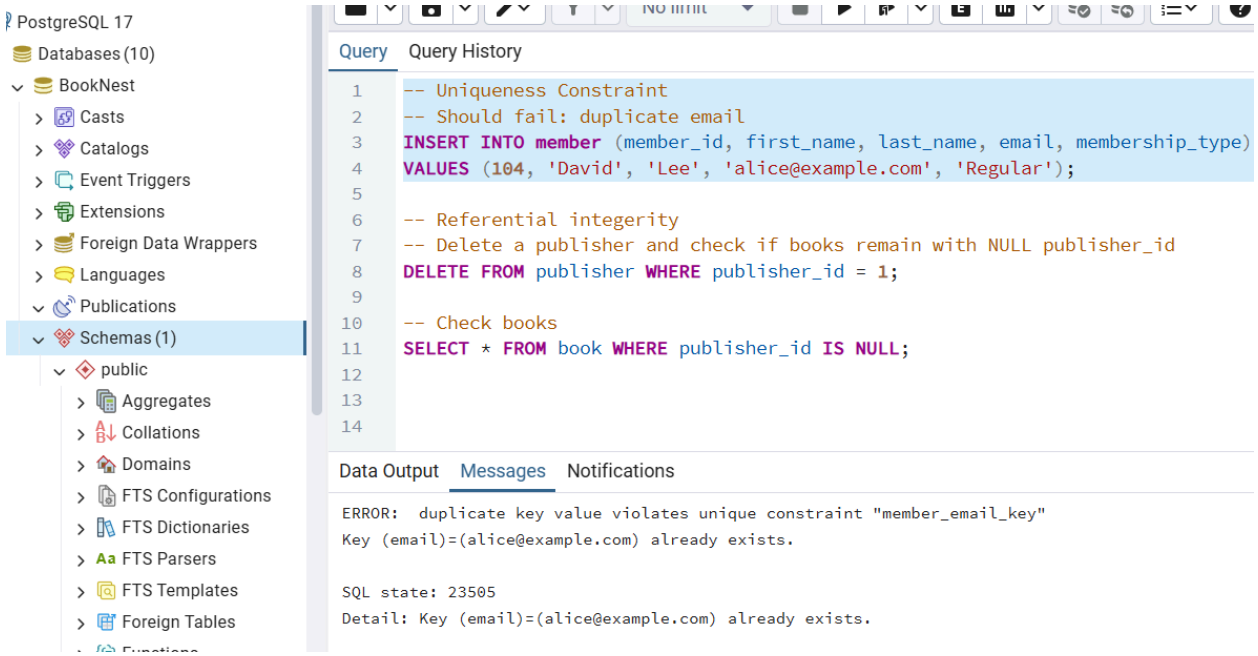
Importing an extension



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

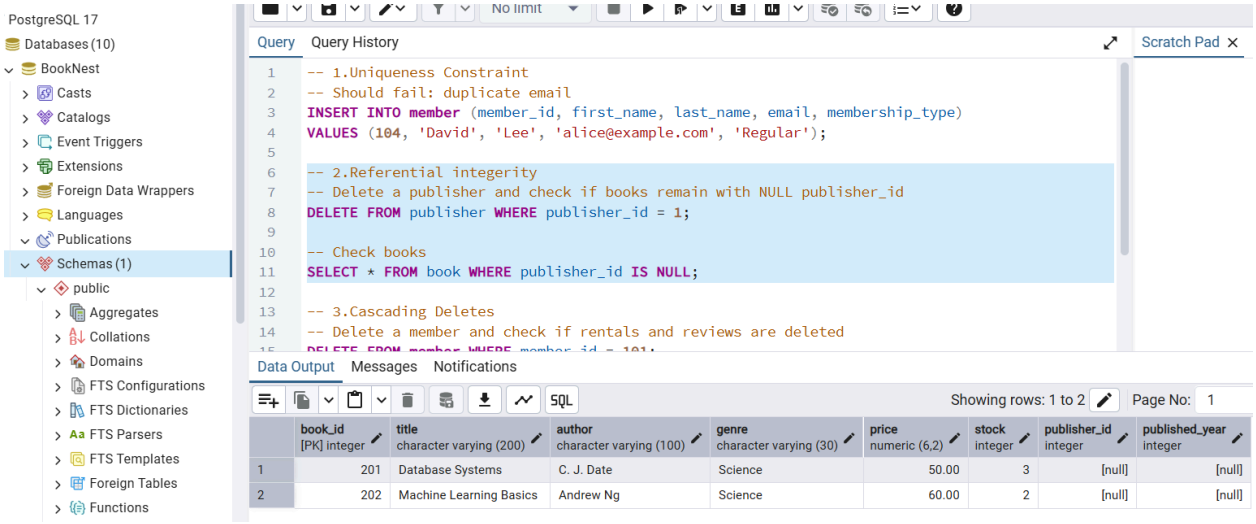
Check uniqueness: Try inserting a member with a duplicate email → should fail.



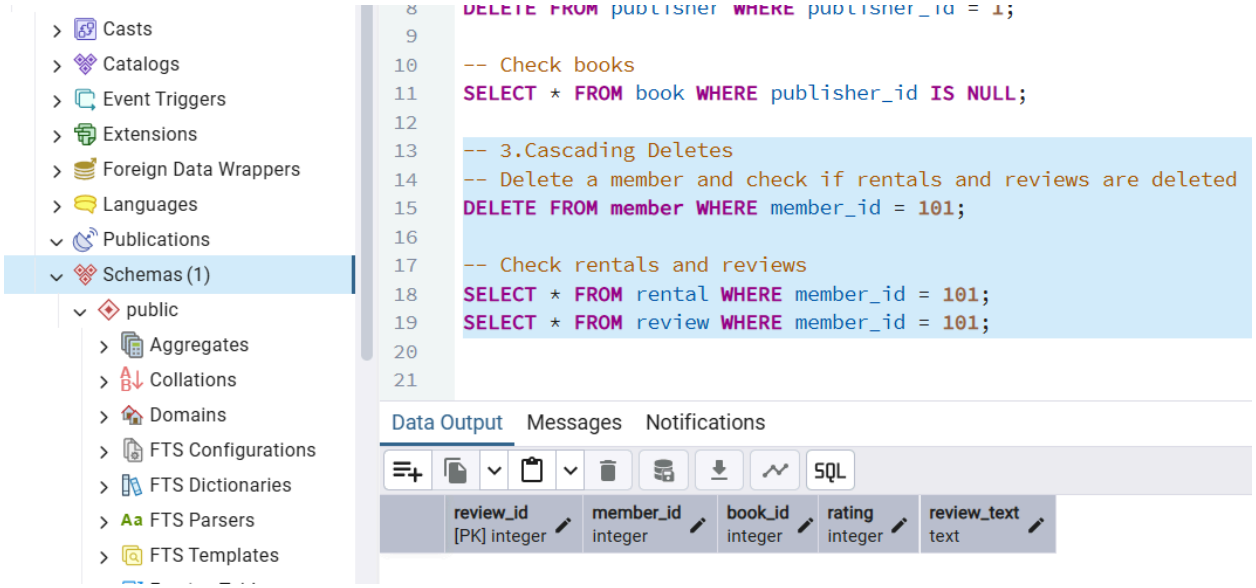
#

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
Scratch Pad
-- 1.Uniqueness Constraint
-- Should fail: duplicate email
INSERT INTO member (member_id, first_name, last_name, email, membership_type)
VALUES (104, 'David', 'Lee', 'alice@example.com', 'Regular');
-- 2.Referential integerity
-- Delete a publisher and check if books remain with NULL publisher_id
DELETE FROM publisher WHERE publisher_id = 1;
-- Check books
SELECT * FROM book WHERE publisher_id IS NULL;
-- 3.Cascading Deletes
-- Delete a member and check if rentals and reviews are deleted
Data Output
Messages
Notifications
Showing rows: 1 to 2
Page No: 1
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]

#

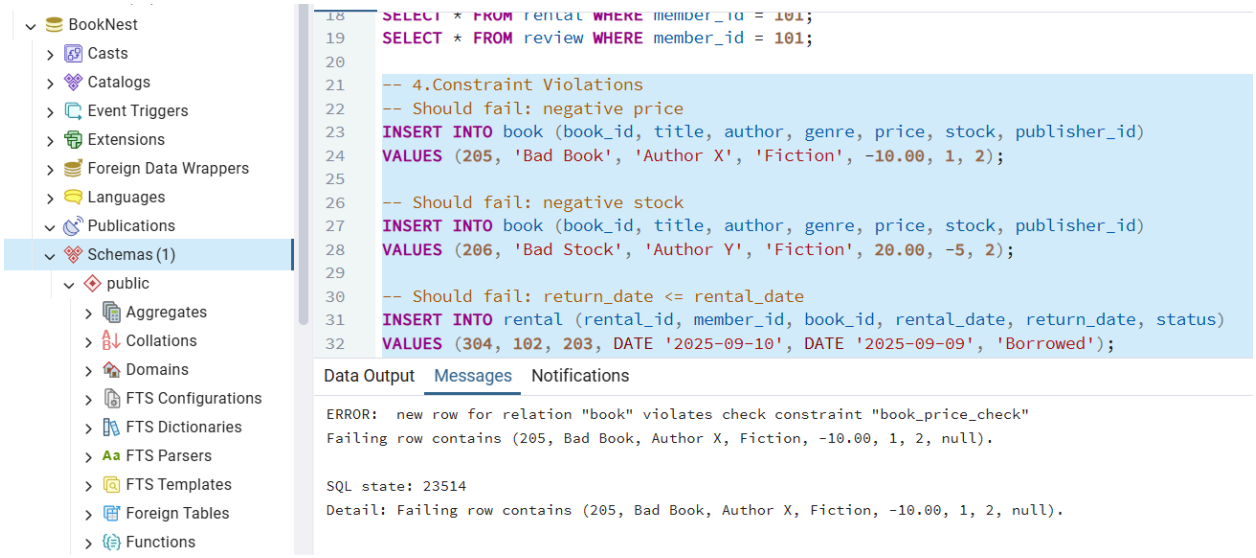

Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
DELETE FROM publisher WHERE publisher_id = 1;
-- Check books
SELECT * FROM book WHERE publisher_id IS NULL;
-- 3.Cascading Deletes
-- Delete a member and check if rentals and reviews are deleted
DELETE FROM member WHERE member_id = 101;
-- Check rentals and reviews
SELECT * FROM rental WHERE member_id = 101;
SELECT * FROM review WHERE member_id = 101;
Data Output
Messages
Notifications
review_id [PK] integer
member_id integer
book_id integer
rating integer
review_text text

#

##

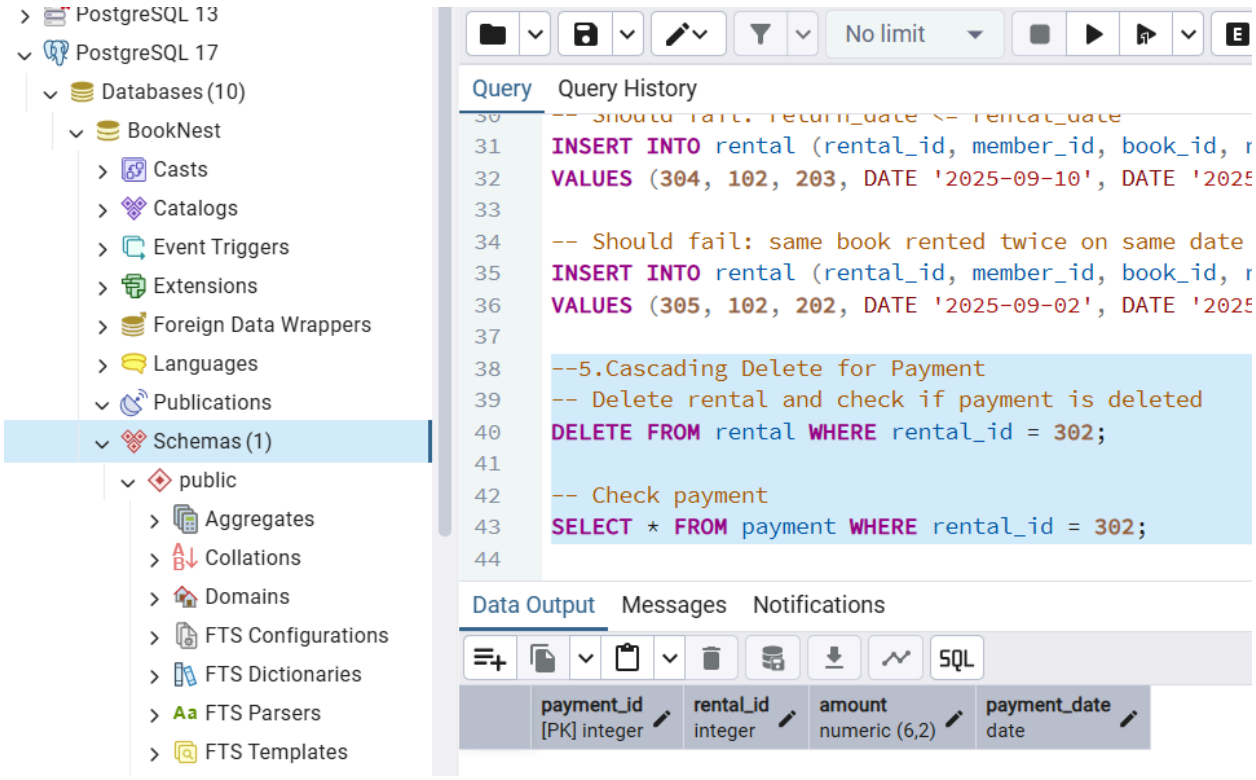
Test constraints:

- 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.



#

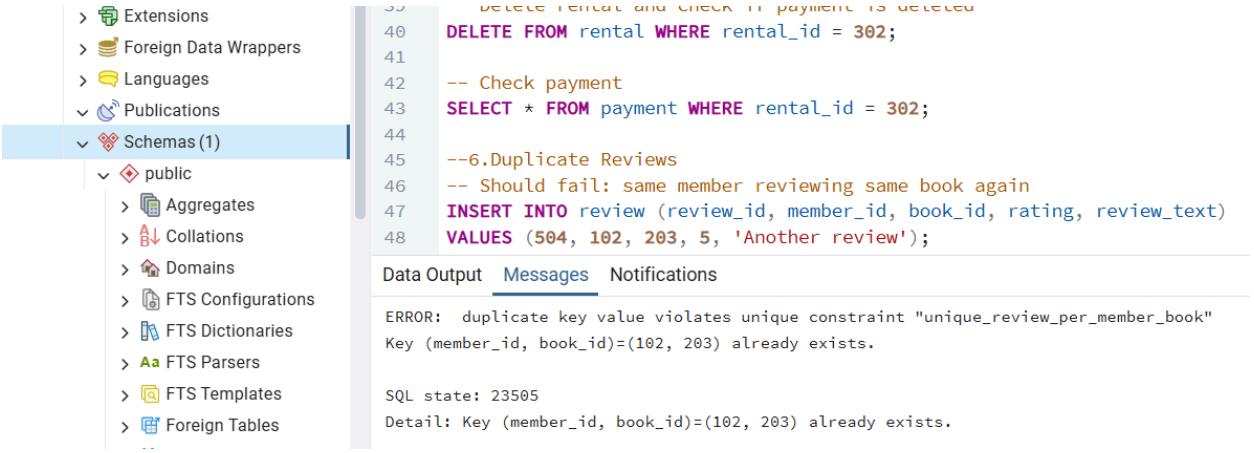
Test cascading deletes: Delete a rental and confirm its payment is deleted automatically.



#
##

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.



##