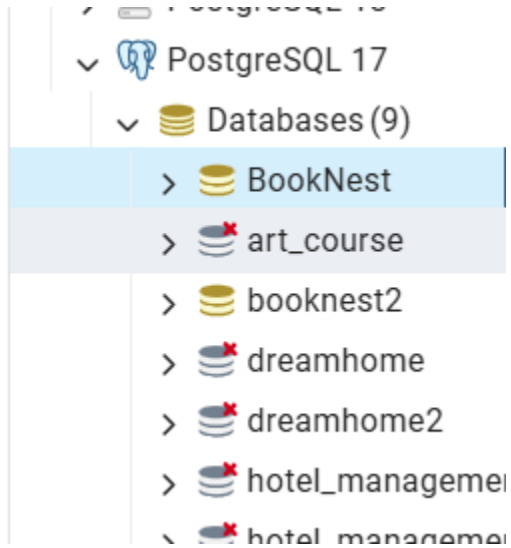


Name: Min Thant Hein
ID: PIUS20230001

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!

-- 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 displays a PostgreSQL client interface. On the left, a tree view shows the database structure: 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, and Tables (2). The 'publisher' table is highlighted under the 'Tables (2)' category. On the right, the 'Query' tab is active, showing the SQL command:

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

 The 'Messages' tab is also visible, showing the output:

```
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

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 (3)

book

member

publisher

Trigger Functions

Types

Views

Subscriptions

BookNest/postgres@PostgreSQL 17

Query

Query History

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

);

);

/*-- 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,

member_id INT NOT NULL,

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

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)

Trigger Functions

Types

Views

Subscriptions

Query

Query History

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

);

-- 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

The screenshot shows a PostgreSQL IDE interface. On the left, a sidebar displays the database schema tree, with 'Tables (5)' selected under the 'public' schema. The main editor window shows the SQL query for creating the 'payment' table, which is highlighted in blue. The query includes a foreign key constraint and a check constraint. Below the query editor, the 'Messages' tab shows the execution result: 'CREATE TABLE' and 'Query returned successfully in 98 msec.'

```
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 -- 6) Review  
64 CREATE TABLE review (  
65     review_id INT PRIMARY KEY,  
66     member_id INT,  
67     book_id INT,
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 98 msec.

-- 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

> 1.3 Sequences

> Tables (6)

> book

> member

> payment

> publisher

> rental

> review

> Trigger Functions

Query Query History

```
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 -- 6) Review  
64 CREATE TABLE review (  
65 review_id INT PRIMARY KEY,  
66 member_id INT,  
67 book_id INT,  
68 rating INT CHECK (rating BETWEEN 1 AND 5),  
69 review_text TEXT,  
70 FOREIGN KEY (member_id) REFERENCES member(member_id)  
71 ON DELETE CASCADE,  
72 FOREIGN KEY (book_id) REFERENCES book(book_id)  
73 ON DELETE CASCADE,  
74 CONSTRAINT unique_review_per_member_book UNIQUE (member_id, book_id)  
75 );  
76  
77
```

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)

The screenshot displays a PostgreSQL client interface. On the left, a schema browser shows the 'BookNest' database structure, with 'public' schemas expanded to show 'book' and 'member' tables. The main window is titled 'BookNest/postgres@PostgreSQL 17' and contains a query editor with the following SQL script:

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

Below the query editor, the 'Messages' tab shows the output: 'ALTER TABLE' and 'Query returned successfully in 53 msec.'

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. Knafl', '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_num...
                                     ^
```

SQL state: 42703

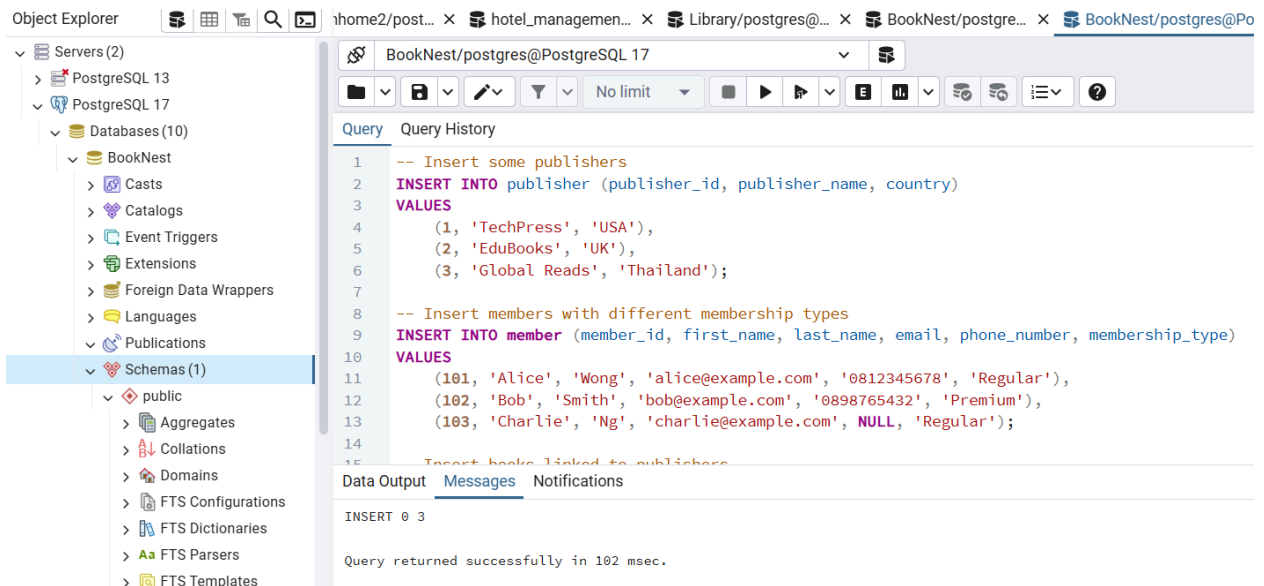
Character: 301

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

The screenshot displays a PostgreSQL client interface. On the left, a tree view shows the database structure: Servers (2) > PostgreSQL 13 > PostgreSQL 17 > Databases (10) > BookNest > Schemas (1) > public. The 'public' schema is selected. The main pane shows the 'Query' tab with the following SQL statement: `1 ALTER TABLE member ADD COLUMN phone_number VARCHAR(20);`. The 'Messages' tab is active, showing the output: `ALTER TABLE` and `Query returned successfully in 59 msec.`. The top toolbar includes icons for file operations, query execution, and other database management functions.

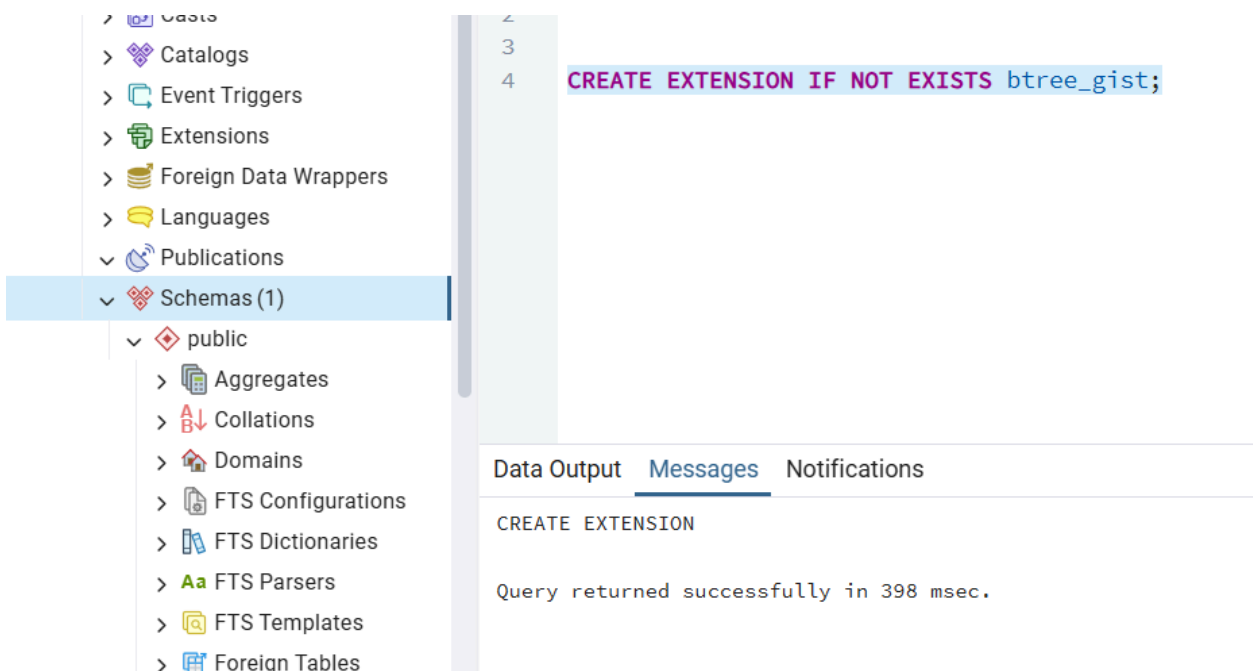
##

Thus, inserting data run successfully!



##

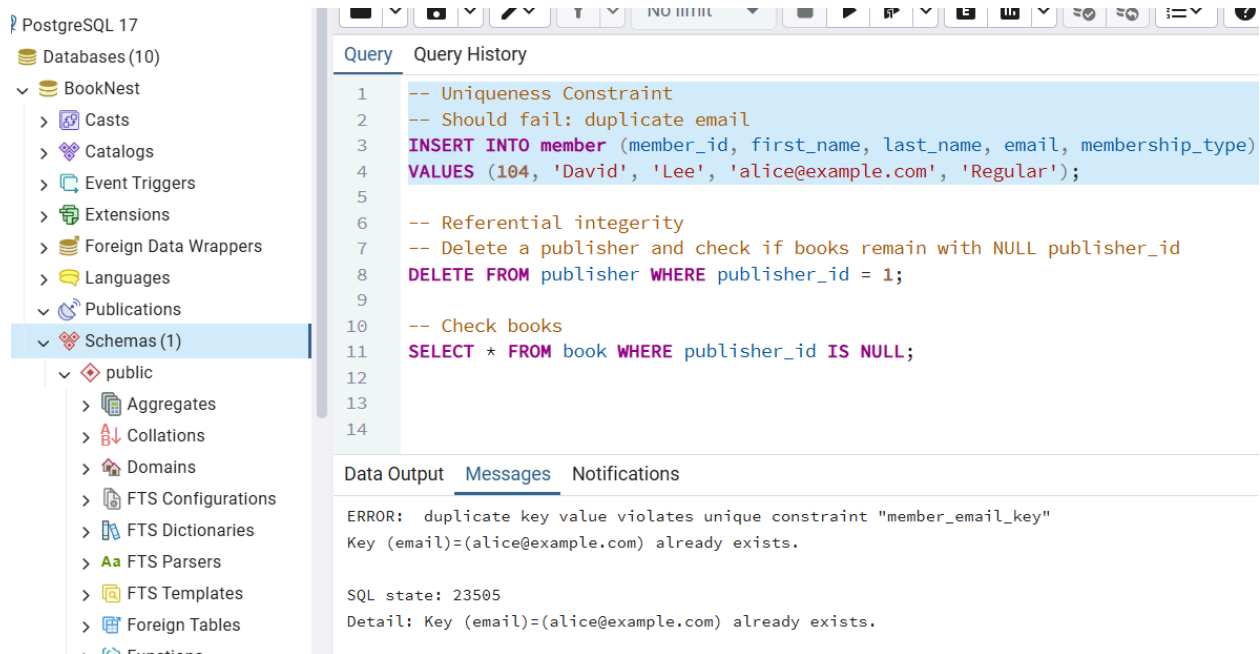
Importing an extension



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.



The screenshot shows the PostgreSQL 17 interface. On the left, the 'Databases (10)' tree is expanded to 'BookNest', then 'Schemas (1)', and finally 'public'. The main window displays a SQL query 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;
12
13
14
```

Below the query editor, the 'Messages' tab is active, showing an error message:

```
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

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

```

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

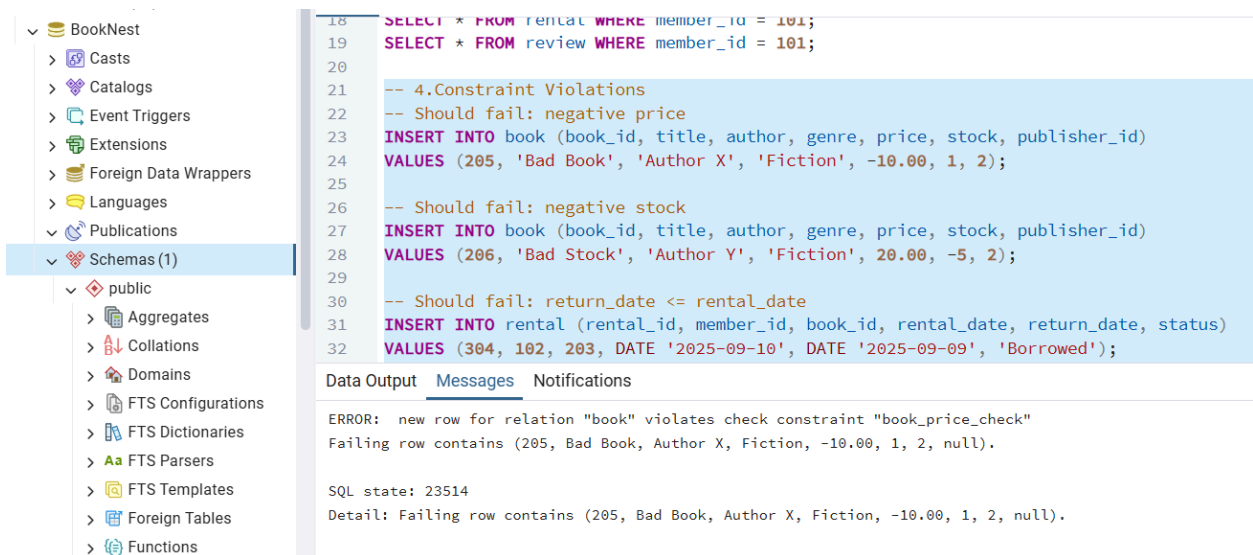
review_id [PK] integer	member_id integer	book_id integer	rating integer	review_text text
------------------------	-------------------	-----------------	----------------	------------------

#

##

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.



The screenshot shows a database IDE interface. On the left is a tree view of the database structure, with 'Schemas (1)' expanded to show the 'public' schema. The main area displays SQL queries and their execution results.

```
18 SELECT * FROM rental WHERE member_id = 101;
19 SELECT * FROM review WHERE member_id = 101;
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 a PostgreSQL client interface with a left-hand sidebar displaying a database schema. The main window is divided into a 'Query' editor and a 'Data Output' table.

Database Schema (Left Sidebar):

- 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

SQL Query (Main Editor):

```
30 -- Should fail: return_date <= rental_date
31 INSERT INTO rental (rental_id, member_id, book_id, r
32 VALUES (304, 102, 203, DATE '2025-09-10', DATE '2025
33
34 -- Should fail: same book rented twice on same date
35 INSERT INTO rental (rental_id, member_id, book_id, r
36 VALUES (305, 102, 202, DATE '2025-09-02', DATE '2025
37
38 --5.Cascading Delete for Payment
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
```

Data Output Table:

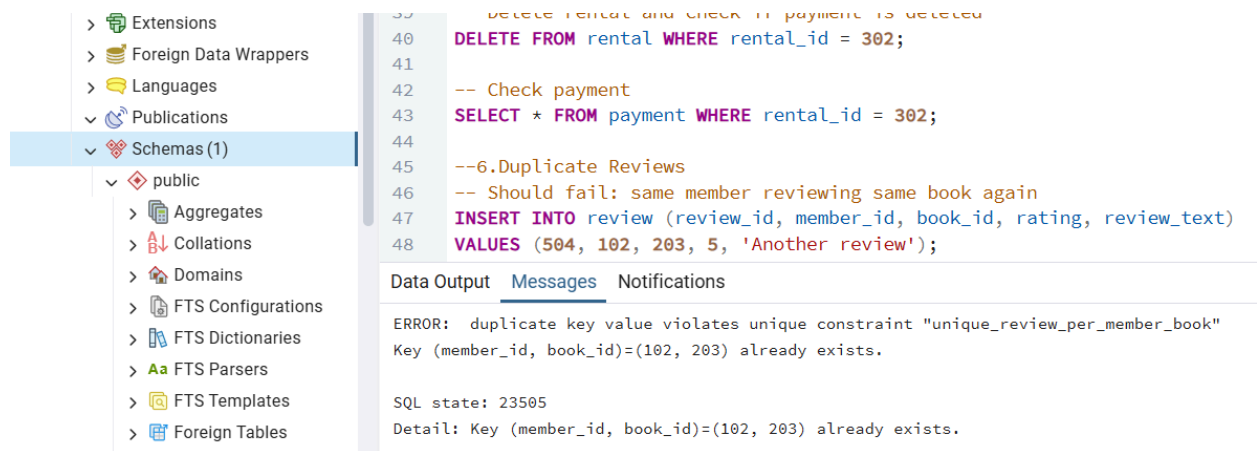
payment_id	rental_id	amount	payment_date
[PK] integer	integer	numeric (6,2)	date

#

##

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 $\text{return_date} \leq \text{rental_date}$,
 - 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 management interface. On the left, a tree view displays the database structure, including Schemas (1) and public. The main area shows SQL execution results. The SQL code includes a DELETE statement for rental_id 302, a SELECT statement for payment, and an INSERT statement for a review. The execution results show an error: "ERROR: duplicate key value violates unique constraint 'unique_review_per_member_book' Key (member_id, book_id)=(102, 203) already exists." The SQL state is 23505 and the detail is "Key (member_id, book_id)=(102, 203) already exists."

```
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');
```

Data Output Messages Notifications

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.

##

