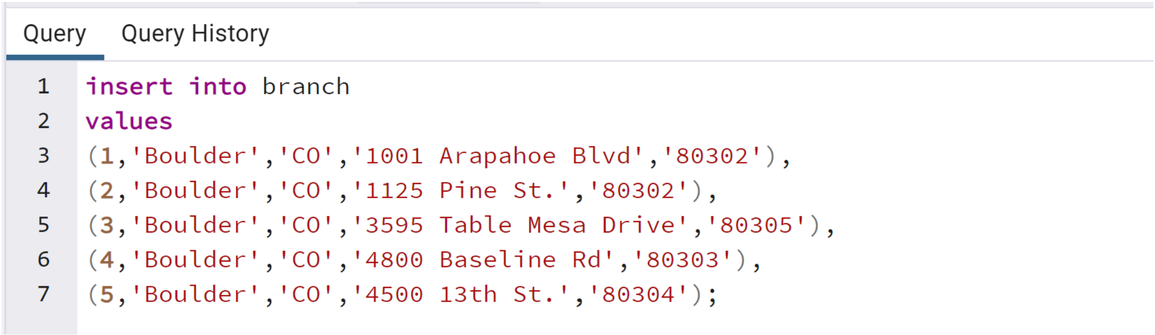
Project: Library Management System hosted on AWS RDS DB.
Goals: Be able to create/drop members, query books by author or title, checkout a book, and return a book.
Skills Used: AWS, Postgres for creating/updating tables, primary/foreign keys, creating and editing functions. Functions are used in this project for abstracting out the SQL code from the user.

Creating the tables.

```
create table branch (
	id integer primary key,
	city text,
	state text,
	street_name text,
	zip_code text
);

create table book (
	id serial primary key,
	title text,
	author text,
	branch_id integer,
	foreign key(branch_id) references branch(id)
);

create table member (
	id serial primary key not null,
	firstname text not null,
	lastname text not null,
	dob date not null,
	city text not null,
	state text not null,
	street_name text not null,
	zip_code text not null,
	date_joined date,
	email text not null,
	branch_id integer not null,
	foreign key(branch_id) references branch(id)
);

create table book_checked_out (
	book_id integer,
	date_out date,
	date_due date,
        date_returned date,
        member_id integer,
        branch_id integer,
        foreign key(member_id) references member(id),
        foreign key(branch_id) references branch(id),
        foreign key(book_id) references book(id)

);
```

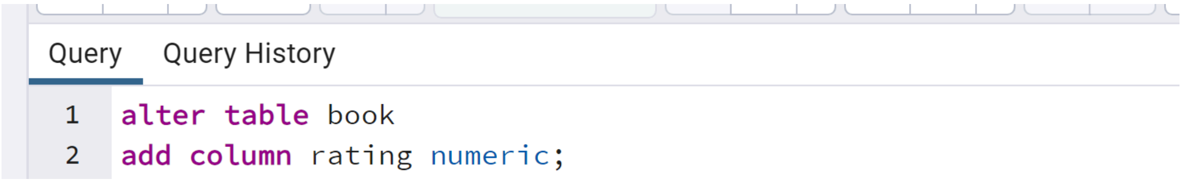
Adding data.

First, insert branch data.



```
insert into branch
values
(1,'Boulder','CO','1001 Arapahoe Blvd','80302'),
(2,'Boulder','CO','1125 Pine St.','80302'),
(3,'Boulder','CO','3595 Table Mesa Drive','80305'),
(4,'Boulder','CO','4800 Baseline Rd','80303'),
(5,'Boulder','CO','4500 13th St.','80304');
```

After looking for book data online, I came across a good data set on kaggle of good books, with average ratings. So, let's add a ratings column to the book table.

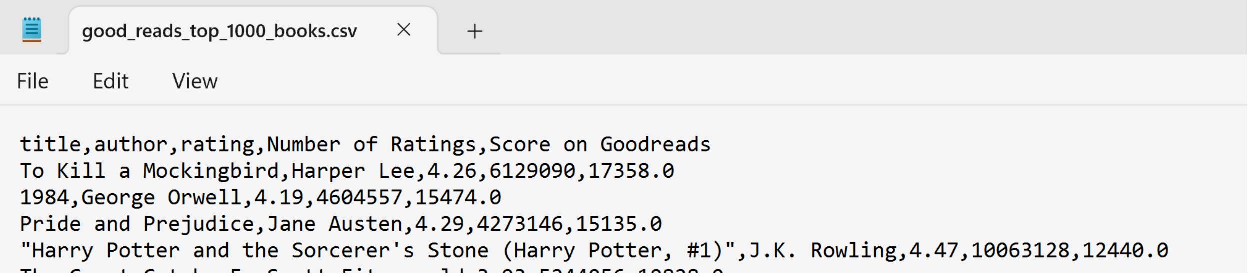


```
alter table book
add column rating numeric;
```

Here is the data:
Data Source: https://www.kaggle.com/datasets/prishasawhney/good-reads-top-1000-books

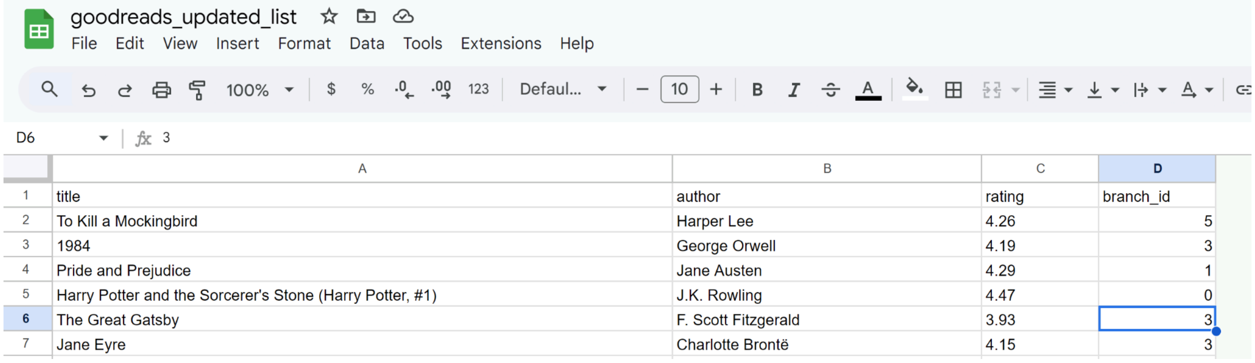
Let's upload this set of books to google sheets, and clean it as needed. Then, we will upload it to the database book table.

Here is a snapshot of the raw data. We change the header names to match the field names of the table.



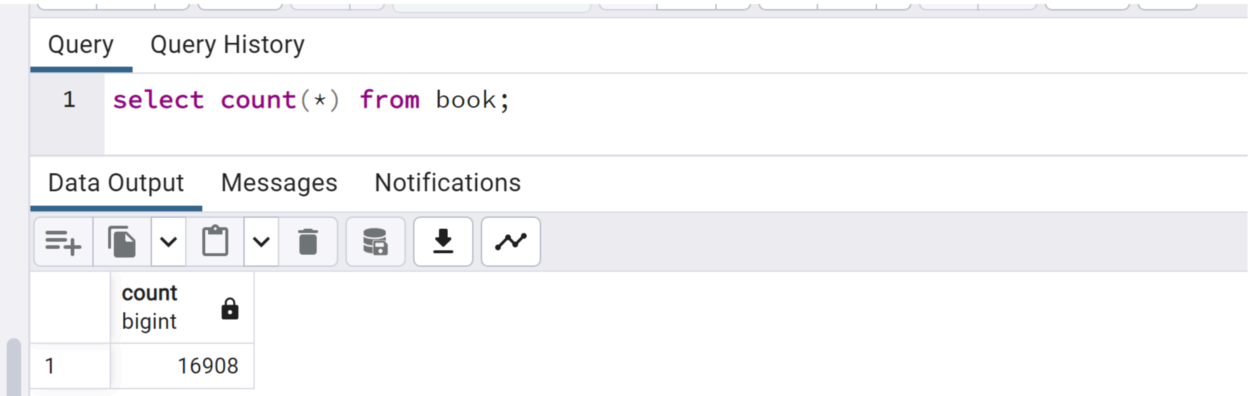
```
title,author,rating,Number of Ratings,Score on Goodreads
To Kill a Mockingbird,Harper Lee,4.26,6129090,17358.0
1984,George Orwell,4.19,4604557,15474.0
Pride and Prejudice,Jane Austen,4.29,4273146,15135.0
"Harry Potter and the Sorcerer's Stone (Harry Potter, #1)",J.K. Rowling,4.47,10063128,12440.0
```

In google sheets, we delete the columns for number of ratings and score on goodreads. We add a column for branch_id and populate with randbetween(0,5). Copy the data set nine times. Then, remove rows with zeros. This will allow us to have different total numbers of copies of some of the titles.



	A	B	C	D
1	title	author	rating	branch_id
2	To Kill a Mockingbird	Harper Lee	4.26	5
3	1984	George Orwell	4.19	3
4	Pride and Prejudice	Jane Austen	4.29	1
5	Harry Potter and the Sorcerer's Stone (Harry Potter, #1)	J.K. Rowling	4.47	0
6	The Great Gatsby	F. Scott Fitzgerald	3.93	3
7	Jane Eyre	Charlotte Brontë	4.15	3

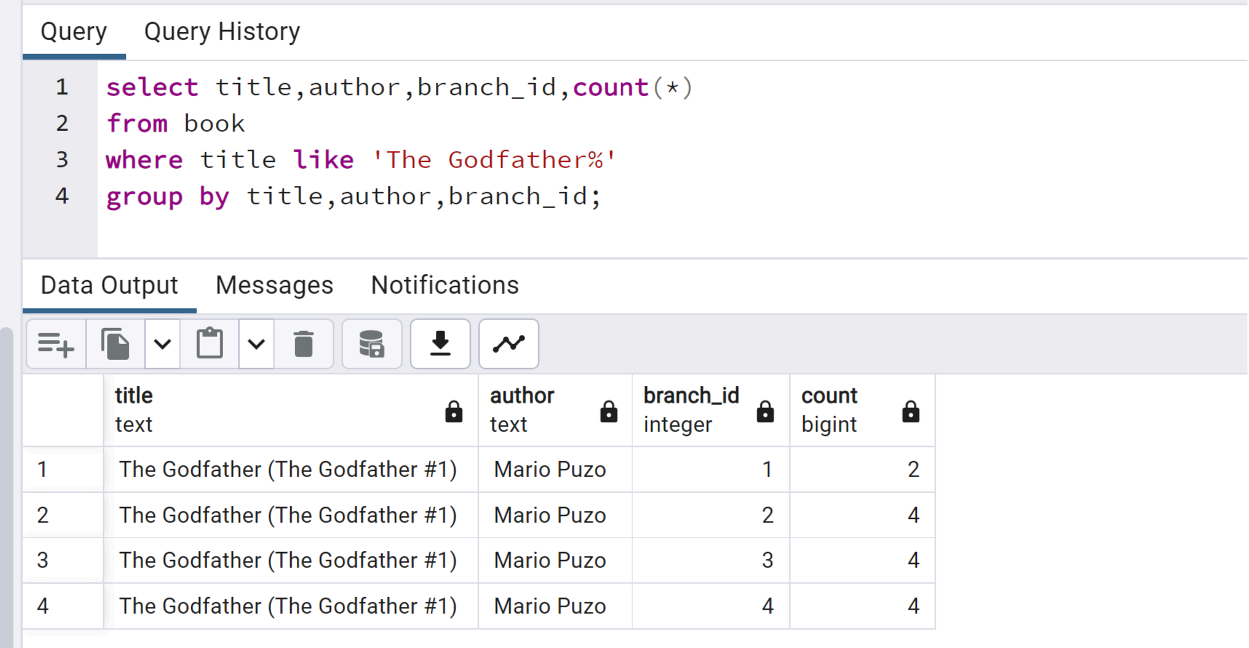
We remove entries with non UTF8 characters, and copy twice into the server DB using the \copy command from inside PSQL.



```
select count(*) from book;
```

	count bigint
1	16908

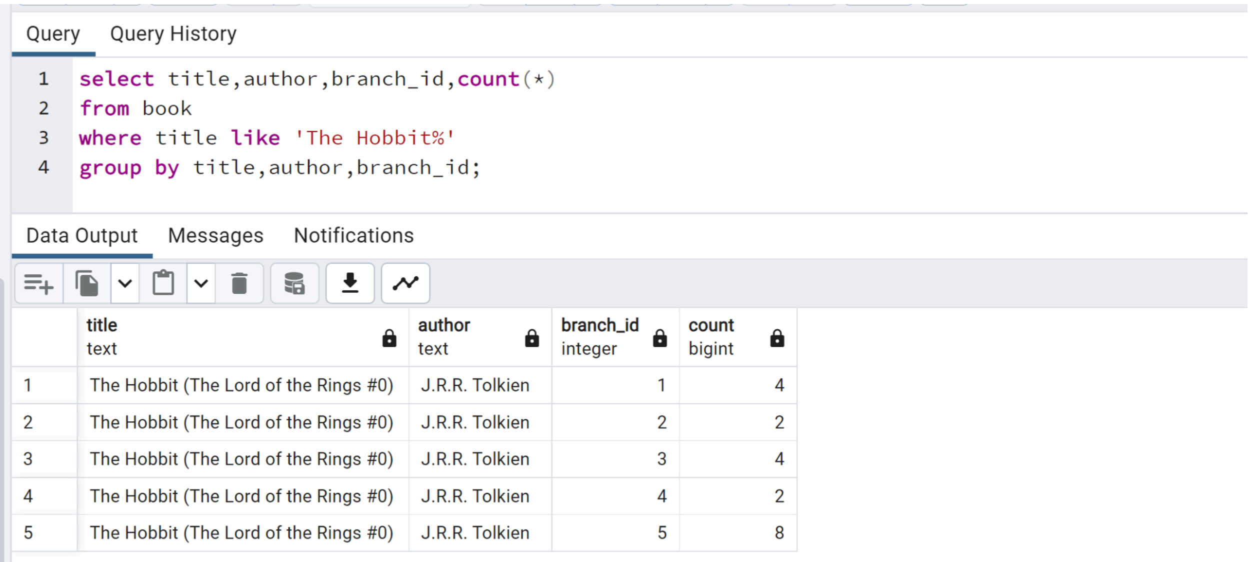
Let’s find out what branches have ‘The Godfather’, and how many copies of each.



```sql
select title,author,branch_id,count(*)
from book
where title like 'The Godfather%'
group by title,author,branch_id;
```

	title text	author text	branch_id integer	count bigint
1	The Godfather (The Godfather #1)	Mario Puzo	1	2
2	The Godfather (The Godfather #1)	Mario Puzo	2	4
3	The Godfather (The Godfather #1)	Mario Puzo	3	4
4	The Godfather (The Godfather #1)	Mario Puzo	4	4

So, there are four copies at branches 2,3,4, two copies at branch 1, and no copies at branch 5. Below is a similar query. We have 8 copies of ‘The Hobbit’ at branch 5.

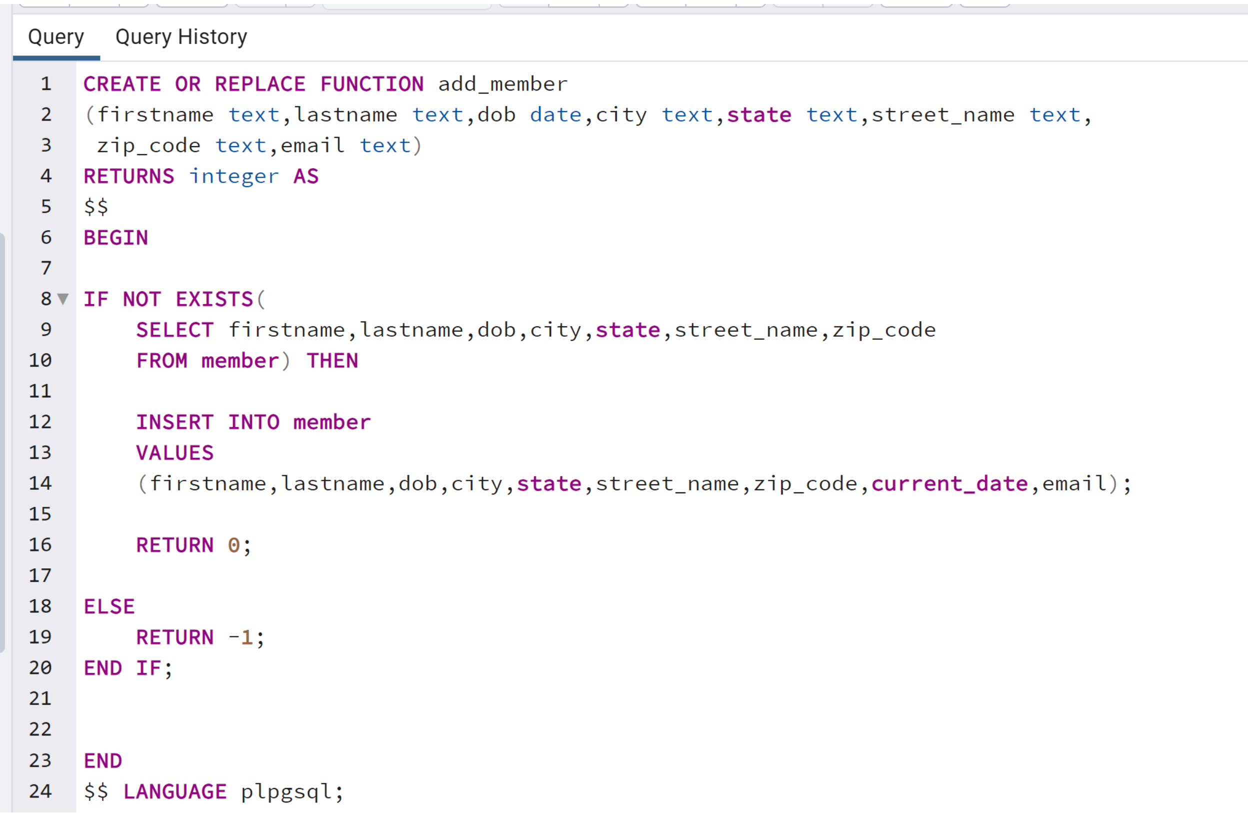


```sql
select title,author,branch_id,count(*)
from book
where title like 'The Hobbit%'
group by title,author,branch_id;
```

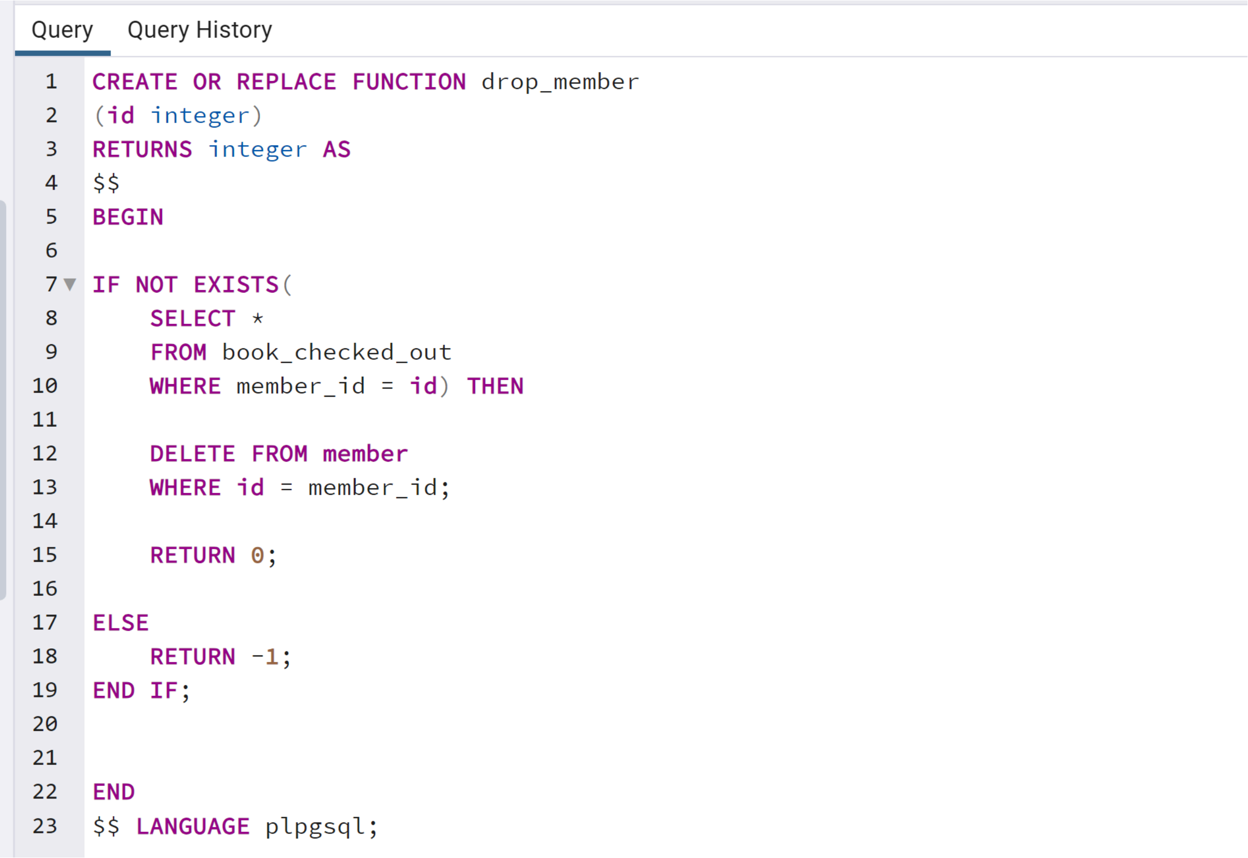
	title text	author text	branch_id integer	count bigint
1	The Hobbit (The Lord of the Rings #0)	J.R.R. Tolkien	1	4
2	The Hobbit (The Lord of the Rings #0)	J.R.R. Tolkien	2	2
3	The Hobbit (The Lord of the Rings #0)	J.R.R. Tolkien	3	4
4	The Hobbit (The Lord of the Rings #0)	J.R.R. Tolkien	4	2
5	The Hobbit (The Lord of the Rings #0)	J.R.R. Tolkien	5	8

Creating functions.

Create functions to facilitate abstraction for the following transactions:
-Adding a member
-Removing a member
-Adding a book
-Removing a book
-Checking out a book
-Returning a book

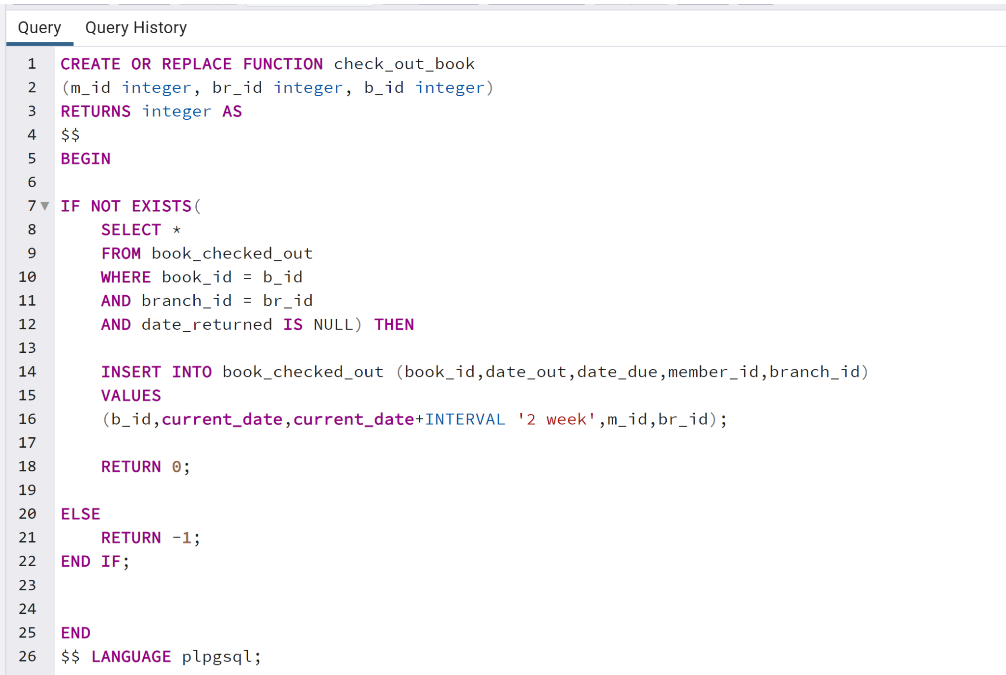


```
CREATE OR REPLACE FUNCTION add_member
(firstname text,lastname text,dob date,city text,state text,street_name text,
 zip_code text,email text)
RETURNS integer AS
$$
BEGIN

IF NOT EXISTS(
    SELECT firstname,lastname,dob,city,state,street_name,zip_code
    FROM member) THEN

    INSERT INTO member
    VALUES
    (firstname,lastname,dob,city,state,street_name,zip_code,current_date,email);

    RETURN 0;

ELSE
    RETURN -1;
END IF;


END
$$ LANGUAGE plpgsql;
```



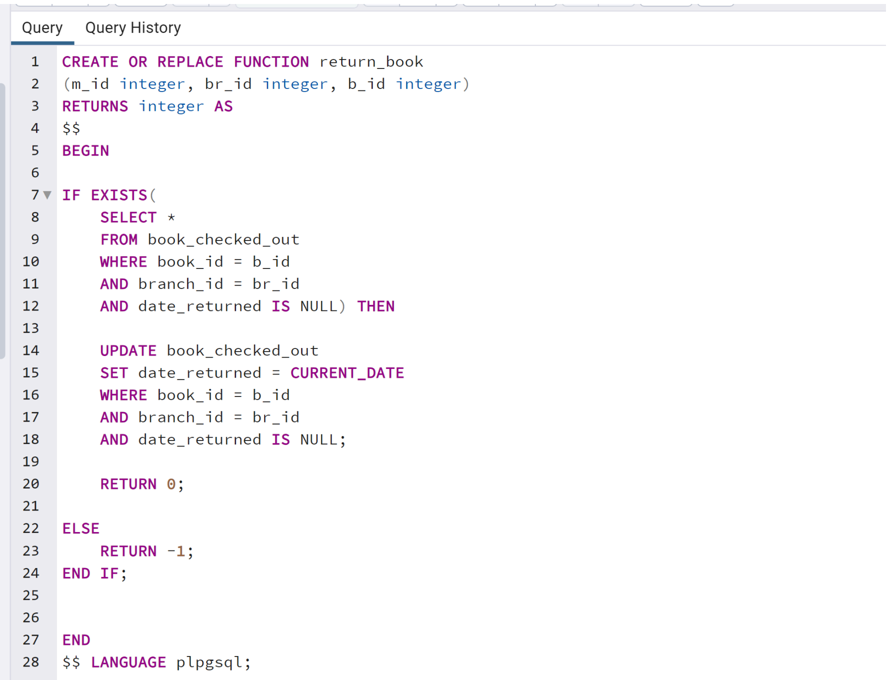
```
CREATE OR REPLACE FUNCTION drop_member
(id integer)
RETURNS integer AS
$$
BEGIN

IF NOT EXISTS(
    SELECT *
    FROM book_checked_out
    WHERE member_id = id) THEN

    DELETE FROM member
    WHERE id = member_id;

    RETURN 0;

ELSE
    RETURN -1;
END IF;


END
$$ LANGUAGE plpgsql;
```

We will require that a member return any due books before terminating their membership.
To be extra safe, the library staff member must remove the member only by their id.



```
CREATE OR REPLACE FUNCTION check_out_book
(m_id integer, br_id integer, b_id integer)
RETURNS integer AS
$$
BEGIN

IF NOT EXISTS(
    SELECT *
    FROM book_checked_out
    WHERE book_id = b_id
    AND branch_id = br_id
    AND date_returned IS NULL) THEN

    INSERT INTO book_checked_out (book_id,date_out,date_due,member_id,branch_id)
    VALUES
    (b_id,current_date,current_date+INTERVAL '2 week',m_id,br_id);

    RETURN 0;

ELSE
    RETURN -1;
END IF;


END
$$ LANGUAGE plpgsql;
```

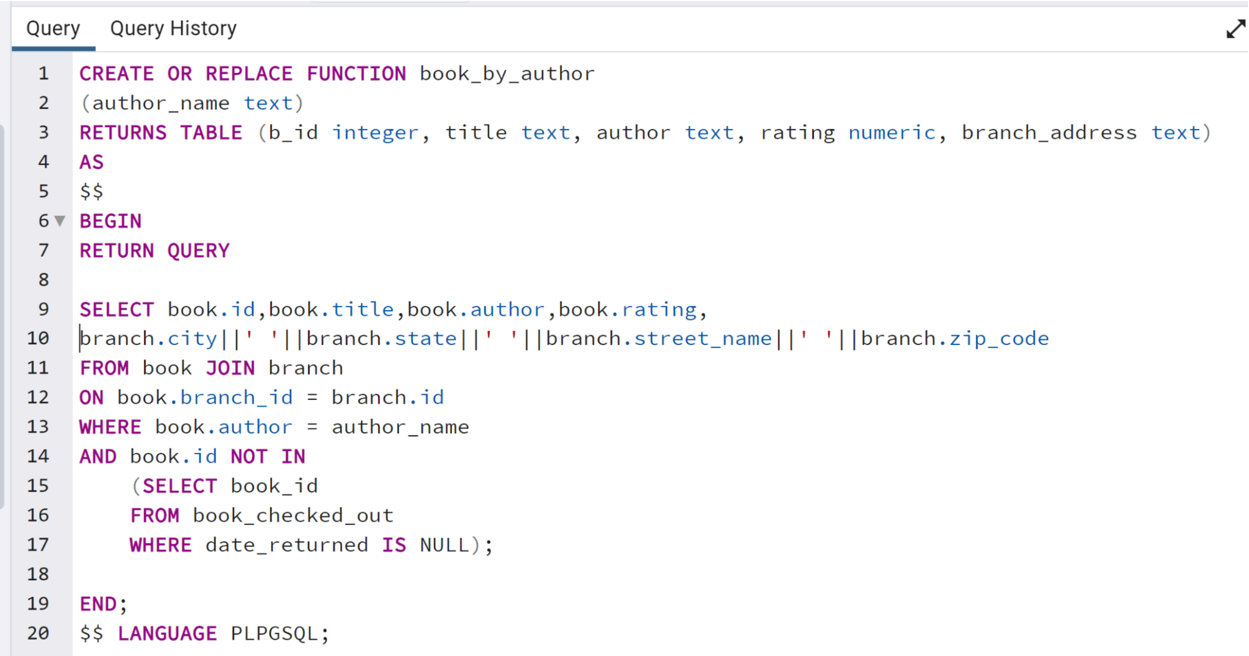
The transaction will execute only if that specific book (b_id) from the specific branch (branch_id) Is not currently checked out.



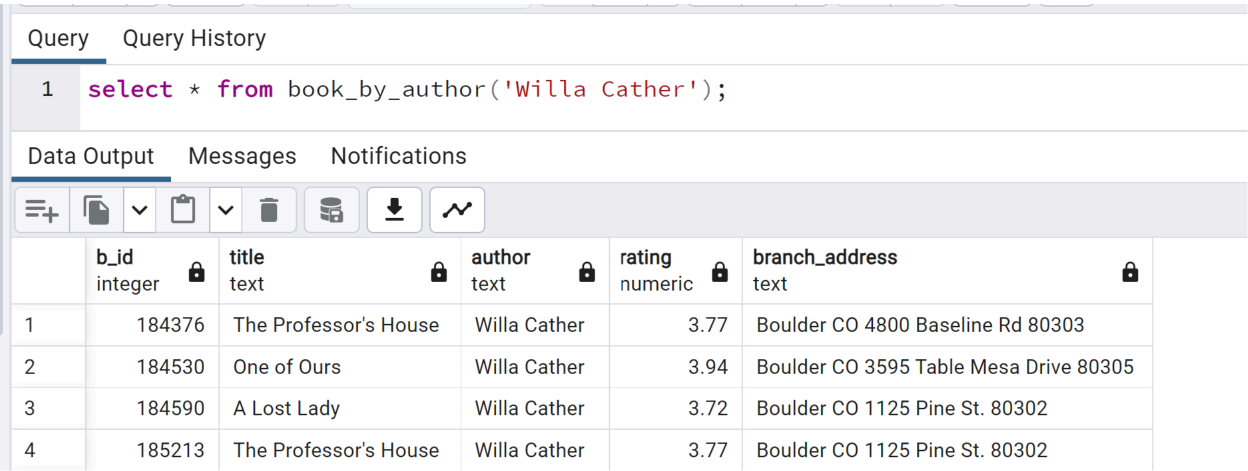
```
CREATE OR REPLACE FUNCTION return_book
(m_id integer, br_id integer, b_id integer)
RETURNS integer AS
$$
BEGIN

IF EXISTS(
    SELECT *
    FROM book_checked_out
    WHERE book_id = b_id
    AND branch_id = br_id
    AND date_returned IS NULL) THEN

    UPDATE book_checked_out
    SET date_returned = CURRENT_DATE
    WHERE book_id = b_id
    AND branch_id = br_id
    AND date_returned IS NULL;

    RETURN 0;

ELSE
    RETURN -1;
END IF;


END
$$ LANGUAGE plpgsql;
```

The reason we include date_returned is null in the update statement is that we want to keep a history of check out and returns for the same item.

After testing these functions, let's create several query functions for the user. These functions will return results sets for searching available books by author or by title.



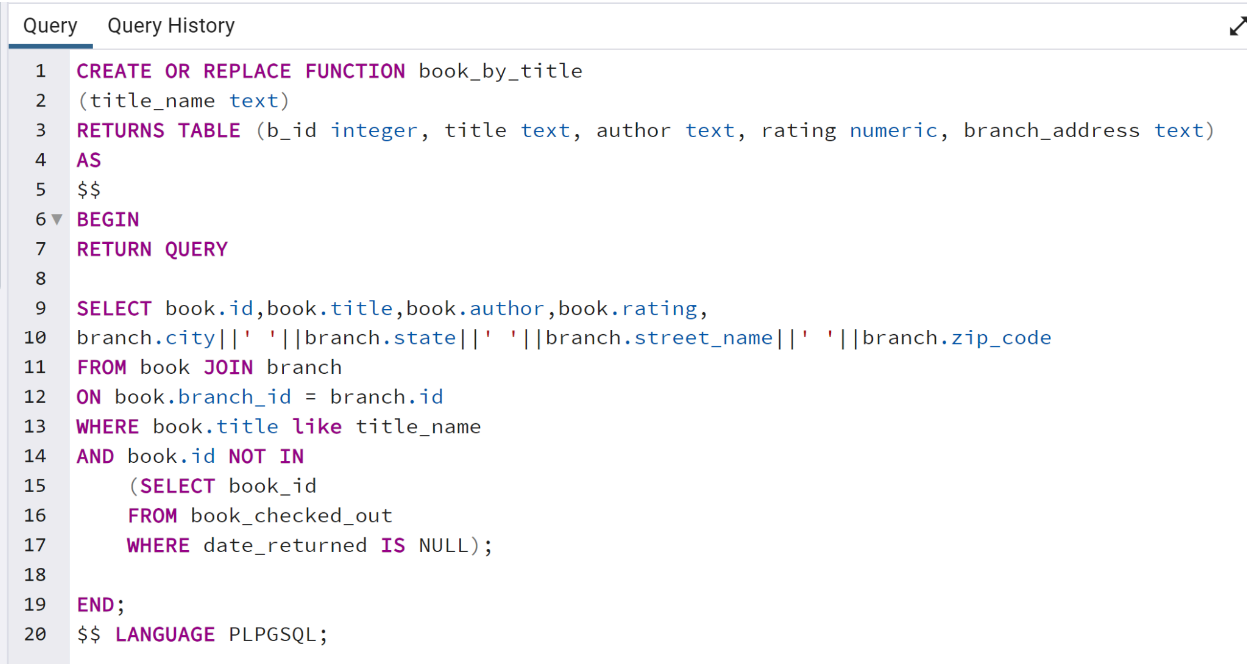
```
CREATE OR REPLACE FUNCTION book_by_author
(author_name text)
RETURNS TABLE (b_id integer, title text, author text, rating numeric, branch_address text)
AS
$$
BEGIN
RETURN QUERY

SELECT book.id,book.title,book.author,book.rating,
branch.city||' '||branch.state||' '||branch.street_name||' '||branch.zip_code
FROM book JOIN branch
ON book.branch_id = branch.id
WHERE book.author = author_name
AND book.id NOT IN
    (SELECT book_id
    FROM book_checked_out
    WHERE date_returned IS NULL);

END;
$$ LANGUAGE PLPGSQL;
```



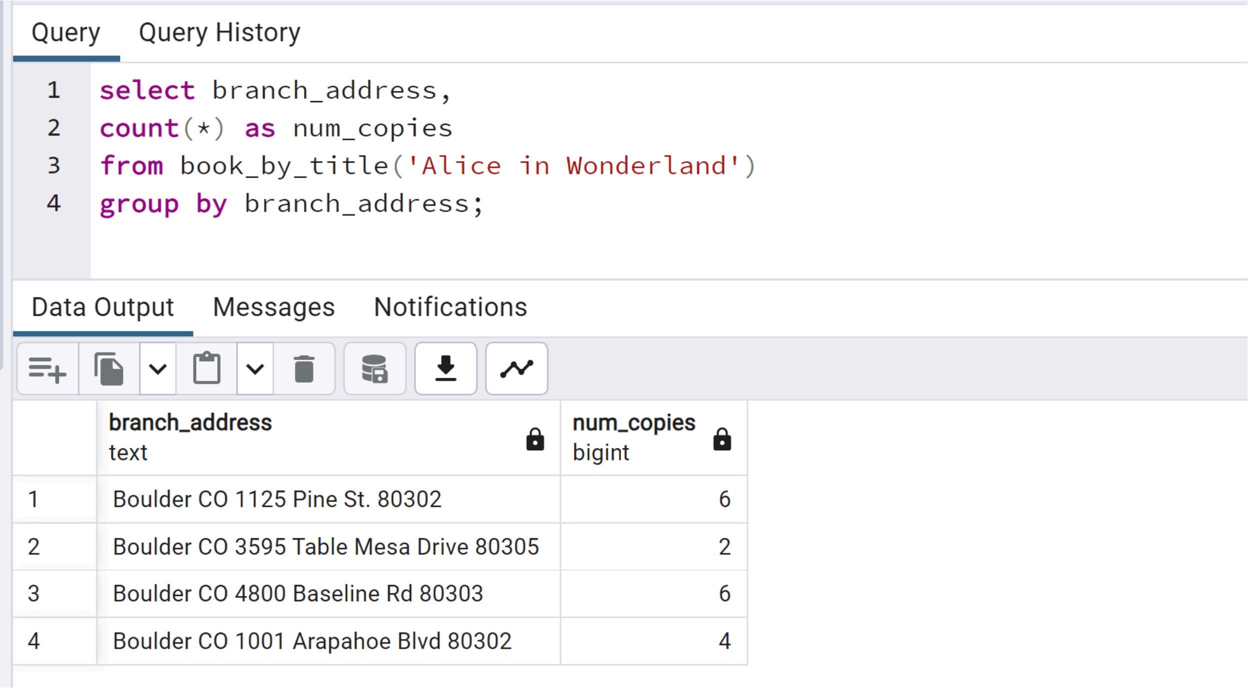
```
select * from book_by_author('Willa Cather');
```

	b_id integer	title text	author text	rating numeric	branch_address text
1	184376	The Professor's House	Willa Cather	3.77	Boulder CO 4800 Baseline Rd 80303
2	184530	One of Ours	Willa Cather	3.94	Boulder CO 3595 Table Mesa Drive 80305
3	184590	A Lost Lady	Willa Cather	3.72	Boulder CO 1125 Pine St. 80302
4	185213	The Professor's House	Willa Cather	3.77	Boulder CO 1125 Pine St. 80302

And here is an almost identical function for searching by title.



```sql
CREATE OR REPLACE FUNCTION book_by_title
(title_name text)
RETURNS TABLE (b_id integer, title text, author text, rating numeric, branch_address text)
AS
$$
BEGIN
RETURN QUERY

SELECT book.id,book.title,book.author,book.rating,
branch.city||' '||branch.state||' '||branch.street_name||' '||branch.zip_code
FROM book JOIN branch
ON book.branch_id = branch.id
WHERE book.title like title_name
AND book.id NOT IN
    (SELECT book_id
    FROM book_checked_out
    WHERE date_returned IS NULL);

END;
$$ LANGUAGE PLPGSQL;
```



```sql
select branch_address,
count(*) as num_copies
from book_by_title('Alice in Wonderland')
group by branch_address;
```

	branch_address text	num_copies bigint
1	Boulder CO 1125 Pine St. 80302	6
2	Boulder CO 3595 Table Mesa Drive 80305	2
3	Boulder CO 4800 Baseline Rd 80303	6
4	Boulder CO 1001 Arapahoe Blvd 80302	4