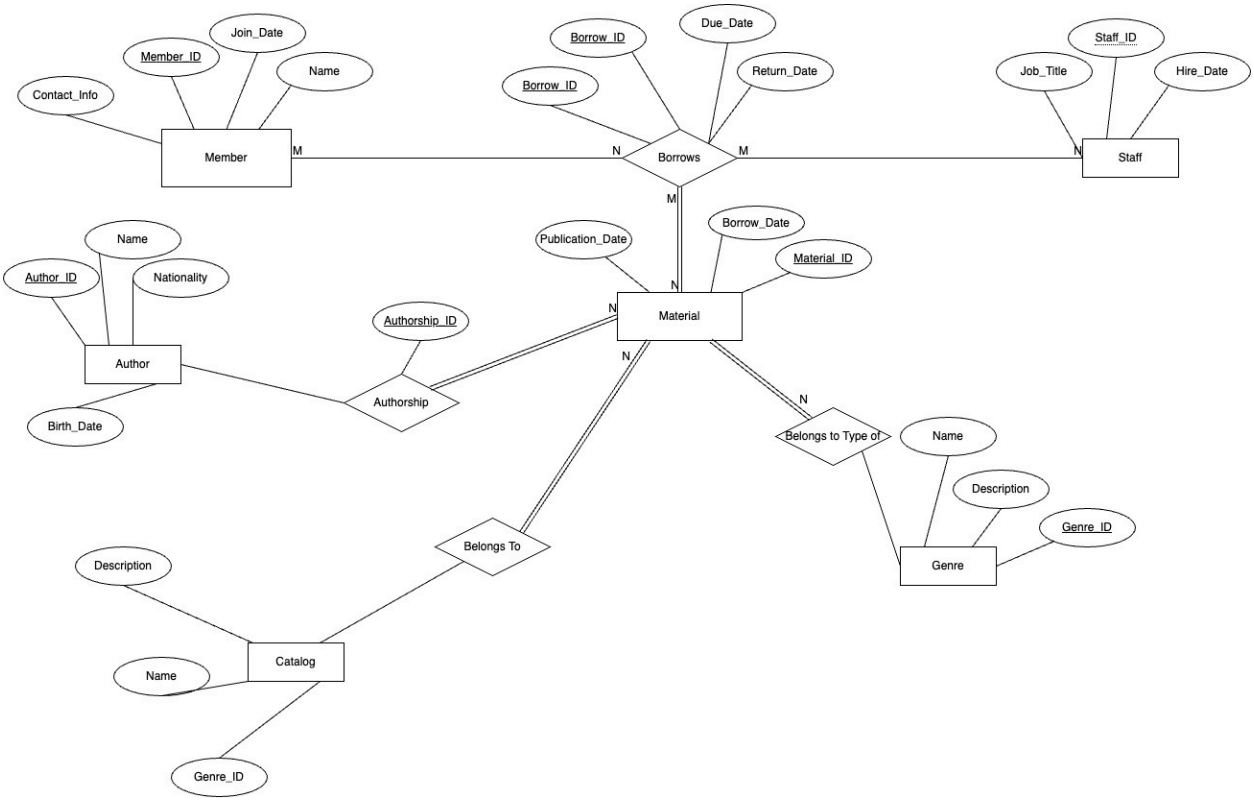
CS 504 – DB Project

ER Diagram



The purpose of this ER diagram is to visually represent the structure of the database that supports the management of library materials, members, staff, and borrowing activities.

Relationships:

1. From the above diagram, we can see Authorship is a relationship which has complete or total participation with Material entity and partial participation with Author entity.
2. The “Material” entity is related to “Author” through “Authorship” relationship with Author_ID and Material_ID attributes are foreign key constraints in Authorship table.
3. The "Material" entity is related to the "Catalog" entity through the "Catalog_ID" attribute, establishing a link between individual materials and their catalog entries.
4. The "Material" entity is related to the "Genre" entity through the "Genre_ID" attribute, associating materials with their respective genres.
5. The "Borrow" entity is related to the "Material" entity through the "Material_ID" attribute, representing the borrowing activity of library materials by members.

Database Implementation

Table creation:

Author Entity:

```
Create table public."Author" (
     "Author_ID" integer NOT NULL ,
    "Name" character varying COLLATE pg_catalog."default",
    "Birth_Date" date,
    "Nationality" character varying COLLATE
pg_catalog."default",
    CONSTRAINT "Author_ID" PRIMARY KEY ("Author_ID")
);
```

Staff Entity

```
Create table public."Staff" (
     "Staff_ID" integer NOT NULL,
    "Name" character varying COLLATE pg_catalog."default",
    "Contact_Info" character varying COLLATE
pg_catalog."default",
    "Job_Title" character varying COLLATE pg_catalog."default",
    "Hire_Date" date,
    CONSTRAINT "Staff_ID" PRIMARY KEY ("Staff_ID")

);
```

Member Entity:

```
Create table public."Member" (
     "Member_ID" integer NOT NULL,
    "Name" character varying COLLATE pg_catalog."default",
    "Contact_Info" character varying COLLATE pg_catalog."default",
    "Join_Date" date,
    CONSTRAINT "Member_ID" PRIMARY KEY ("Member_ID")
);
```

Genre Entity:

```
Create table public."Genre" (
     "Genre_ID" integer NOT NULL,
    "Name" character varying COLLATE pg_catalog."default",
    "Description" character varying COLLATE
pg_catalog."default",
    CONSTRAINT "Genre_ID" PRIMARY KEY ("Genre_ID")
);
```

Catalog Entity:

```
Create table public."Catalog" (
     "Catalog_ID" integer NOT NULL,
    "Name" character varying COLLATE pg_catalog."default",
    "Location" character varying COLLATE pg_catalog."default",
    CONSTRAINT "Catalog_ID" PRIMARY KEY ("Catalog_ID")
);
```

Material Entity:

```
CREATE TABLE public."Material" (
    Material_ID SERIAL PRIMARY KEY,
    Title VARCHAR(255) NOT NULL,
    Publication_Date DATE,
    Catalog_ID INT REFERENCES public."Catalog"("Catalog_ID"),
    Genre_ID INT REFERENCES public."Genre"("Genre_ID")
);
```

Borrow Entity:

```
CREATE TABLE public."Borrow" (
    Borrow_ID SERIAL PRIMARY KEY,
    Material_ID INT REFERENCES public."Material"("material_id"),
    Member_ID INT REFERENCES public."Member"("Member_ID"),
    Staff_ID INT REFERENCES public."Staff"("Staff_ID"),
    Borrow_Date DATE,
    Due_Date DATE,
    Return_Date DATE
);
```

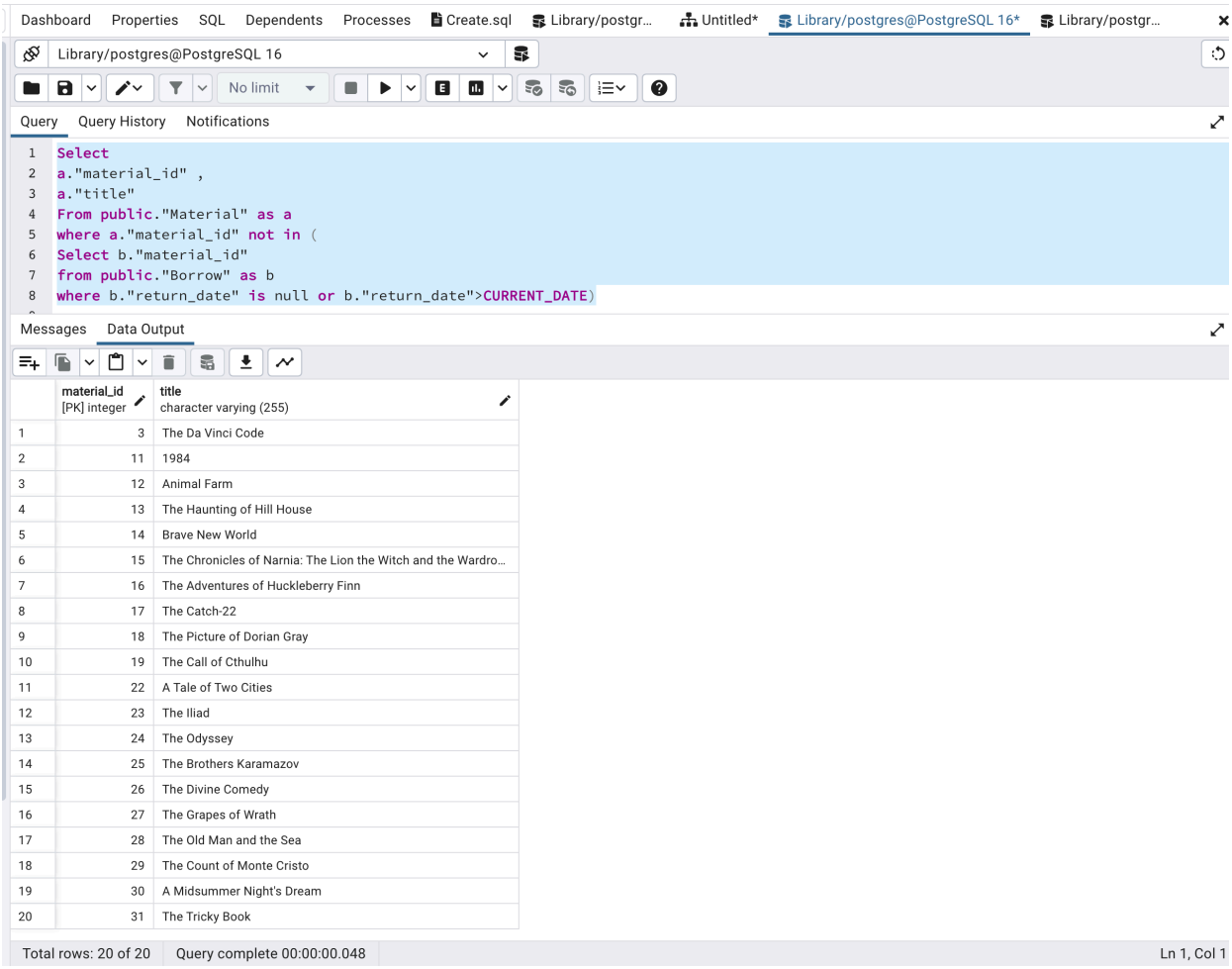
Insertion:

I have used 'Import/Export Data' feature in PgAdmin to insert values to all the tables from the required CSV files.

Queries:

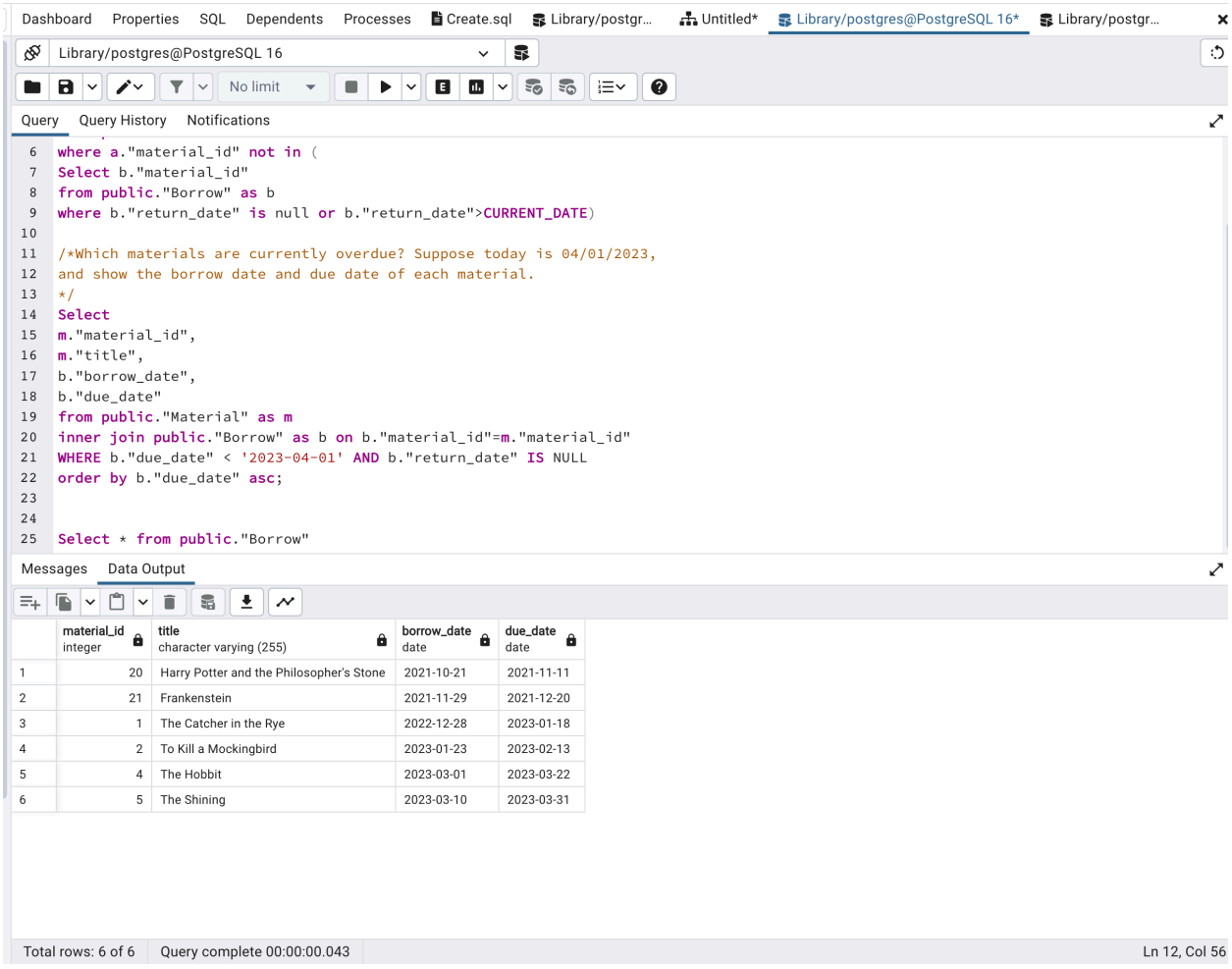
1.Which materials are currently available in the library? If a material is borrowed and not returned, it's not considered as available.

```
Select
a."material_id" ,
a."title"
From public."Material" as a
where a."material_id" not in (
Select b."material_id"
from public."Borrow" as b
where b."return_date" is null or b."return_date">CURRENT_DATE);
```



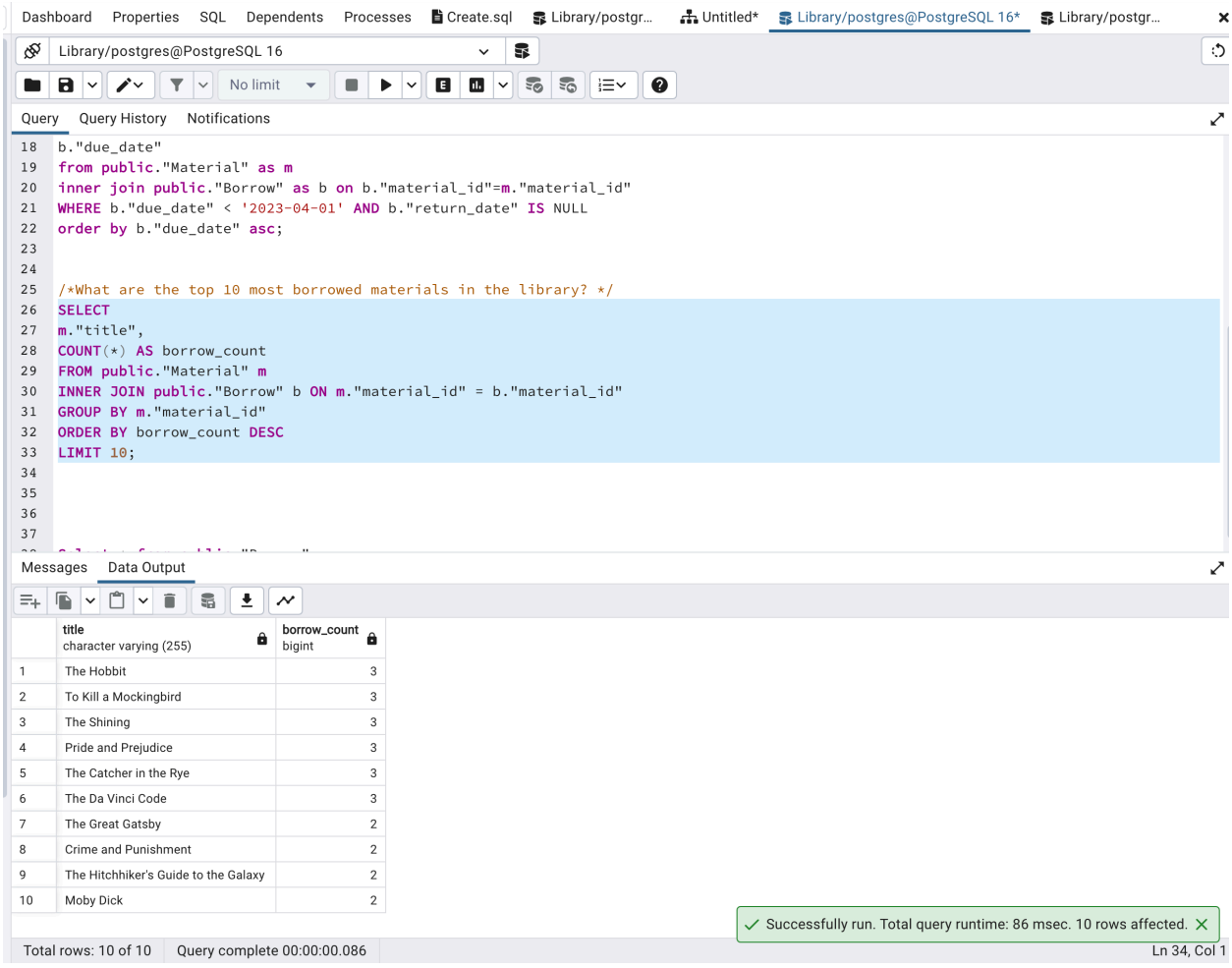
2. Which materials are currently overdue? Suppose today is 04/01/2023, and show the borrow date and due date of each material.

```
Select
m."Material_ID",
m."Title",
b."Borrow_Date",
b."Due_Date"
from public."Material" as m
inner join public."Borrow" as b on
b."Material_ID"=m."Material_ID"
WHERE b."Due_Date" < '2023-04-01' AND b."Return_Date" IS NULL
order by m."Material_ID";
```



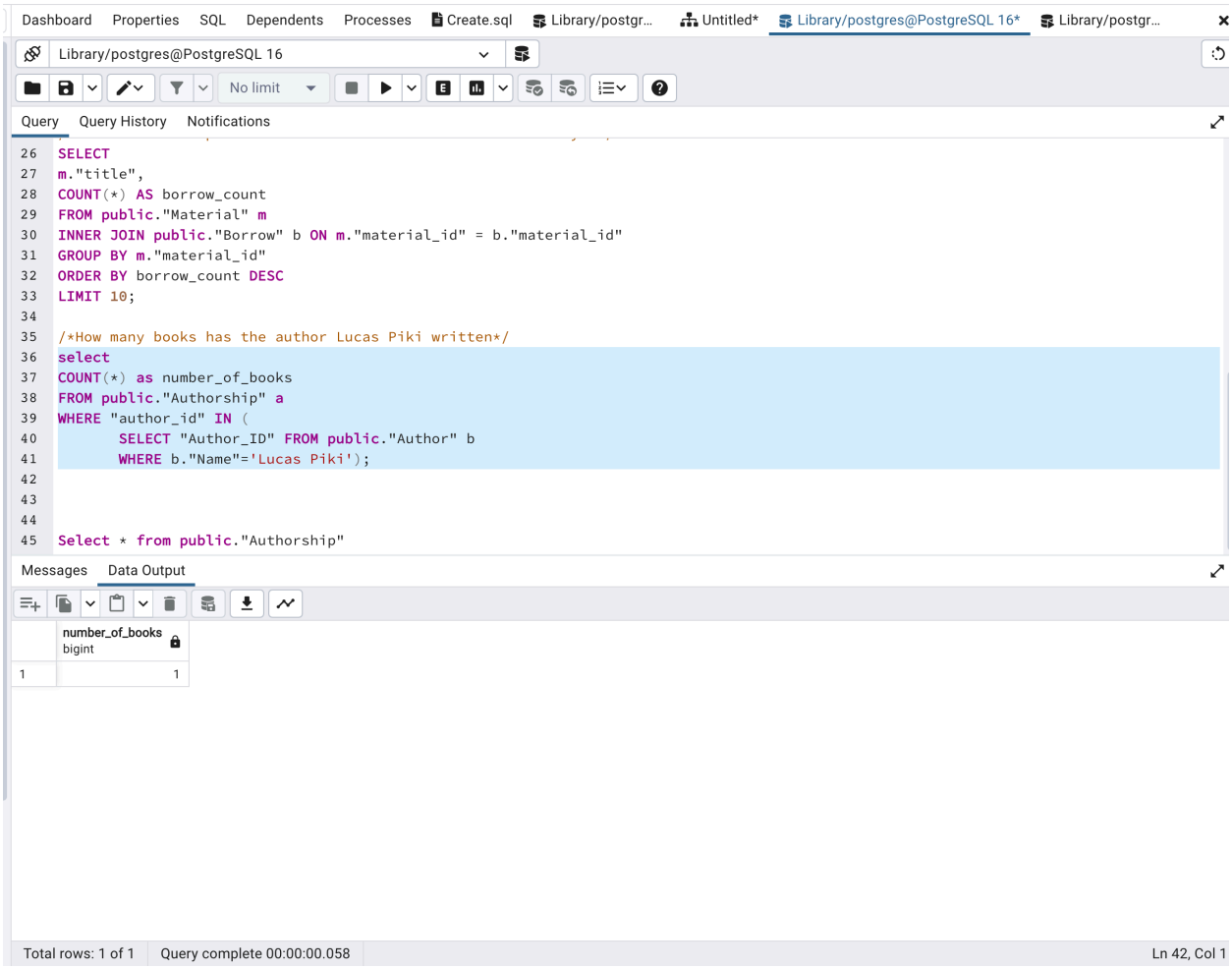
Q3. What are the top 10 most borrowed materials in the library? Show the title of each material and order them based on their available counts.

```
SELECT
m."title",
COUNT(*) AS borrow_count
FROM public."Material" m
INNER JOIN public."Borrow" b ON m."material_id" =
b."material_id"
GROUP BY m."material_id"
ORDER BY borrow_count DESC
LIMIT 10;
```



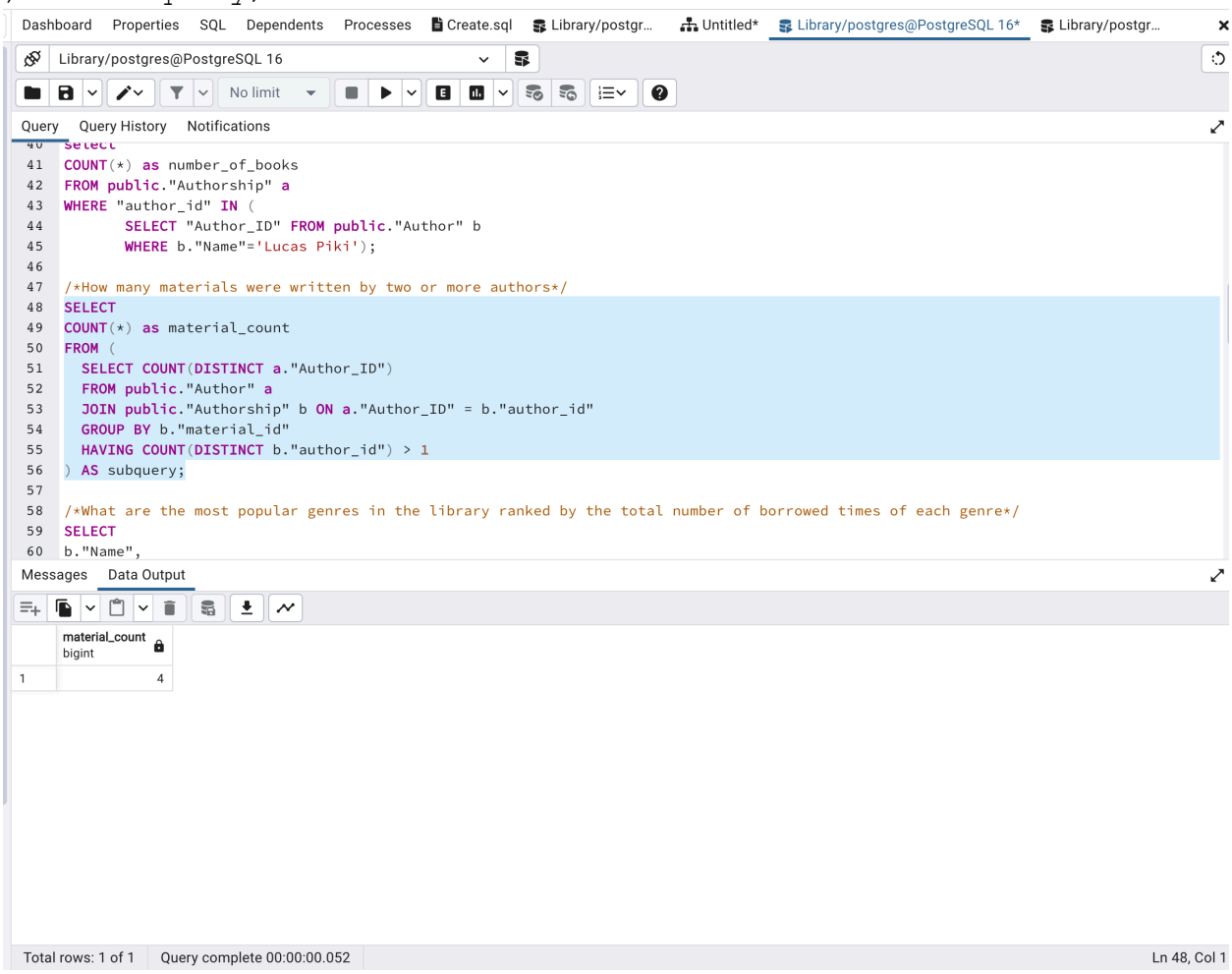
Q4. How many materials has the author Lucas Piki written?

```
select
COUNT(*) as number_of_books
FROM public."Authorship" a
WHERE "author_id" IN (
       SELECT "Author_ID" FROM public."Author" b
       WHERE b."Name"='Lucas Piki');
```



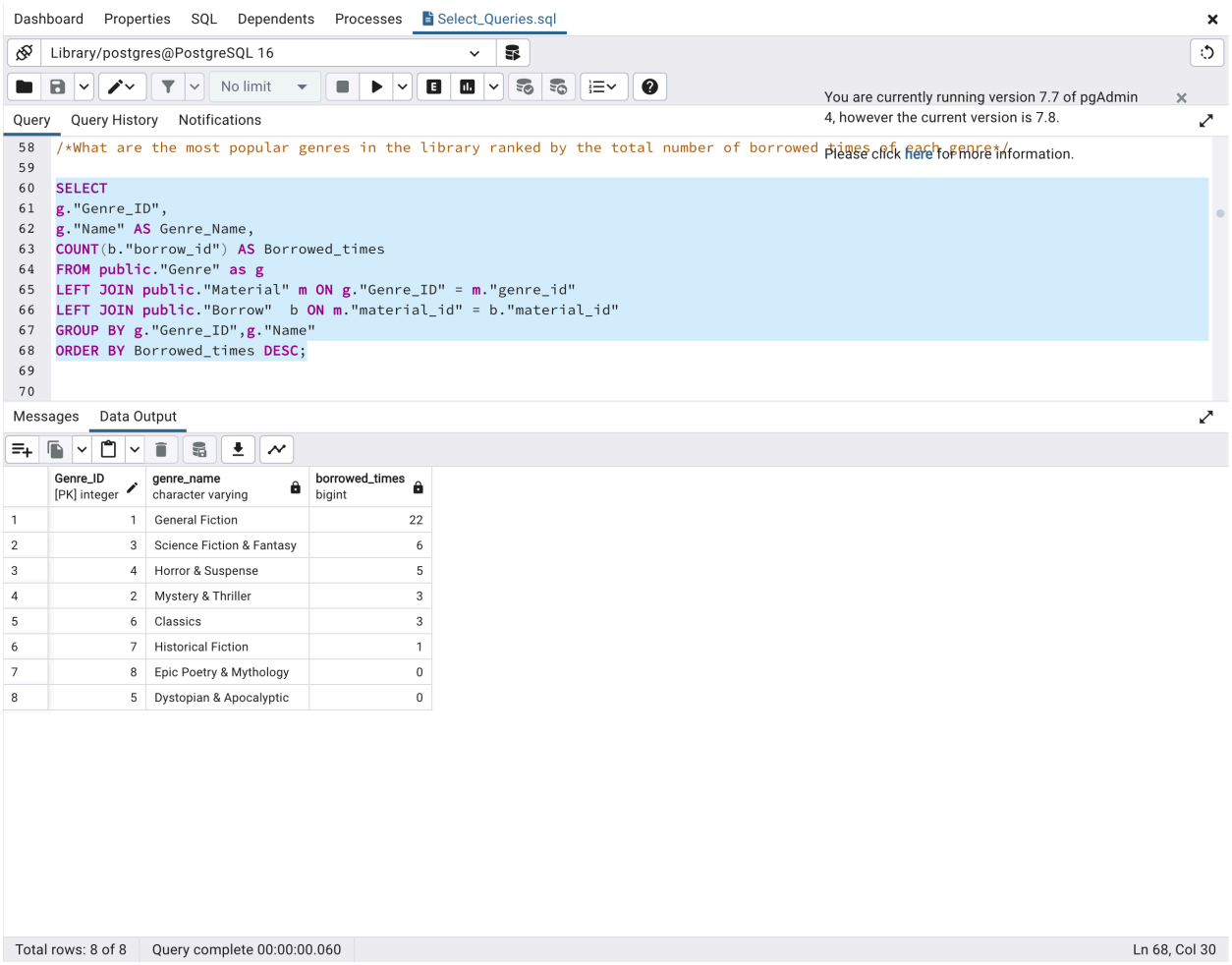
Q5. How many materials were written by two or more authors?

```
SELECT
COUNT(*) as material_count
FROM (
  SELECT COUNT(DISTINCT a."Author_ID")
  FROM public."Author" a
  JOIN public."Authorship" b ON a."Author_ID" = b."author_id"
  GROUP BY b."material_id"
  HAVING COUNT(DISTINCT b."author_id") > 1
) AS subquery;
```



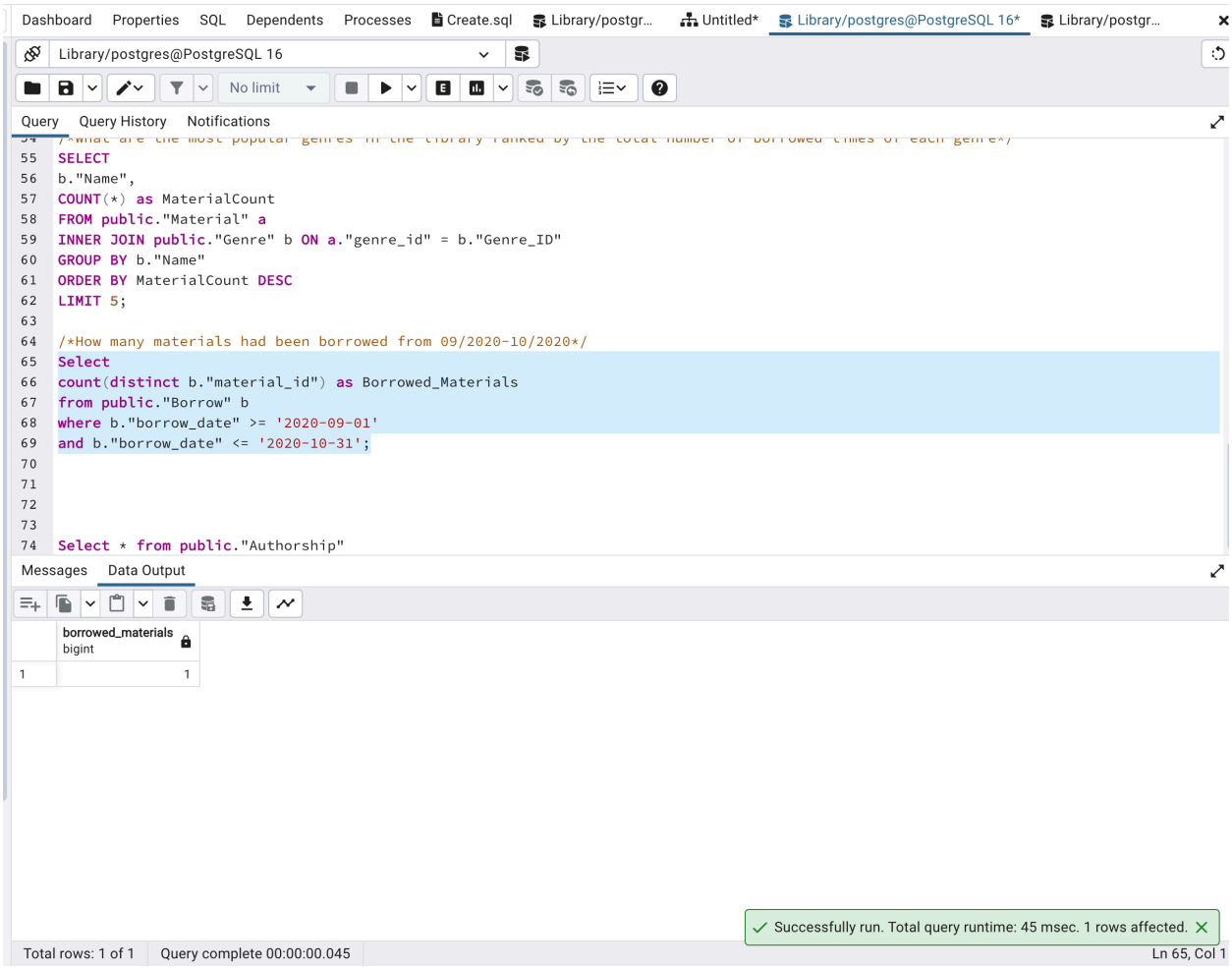
Q6. What are the most popular genres in the library ranked by the total number of borrowed times of each genre?

```
SELECT
g."Genre_ID",
g."Name" AS Genre_Name,
COUNT(b."borrow_id") AS Borrowed_times
FROM public."Genre" as g
LEFT JOIN public."Material" m ON g."Genre_ID" = m."genre_id"
LEFT JOIN public."Borrow"  b ON m."material_id" =
b."material_id"
GROUP BY g."Genre_ID",g."Name"
ORDER BY Borrowed_times DESC;
```



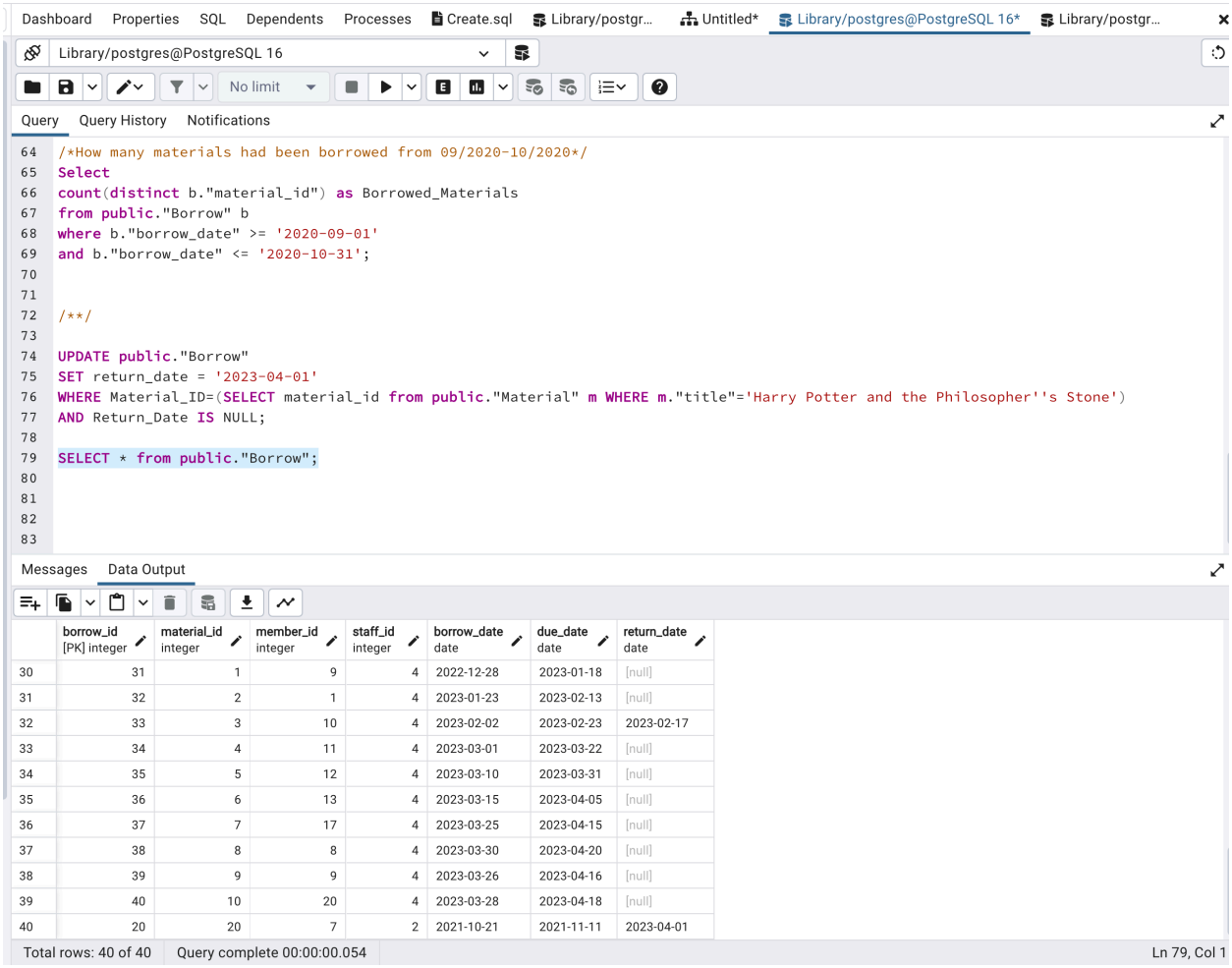
Q7. How many materials had been borrowed from 09/2020-10/2020?

```
Select
count(distinct b."Material_ID") as Borrowed_Materials
from public."Borrow" b
where b."Borrow_Date" >= '2020-09-01'
and b."Borrow_Date" <= '2020-10-31';
```



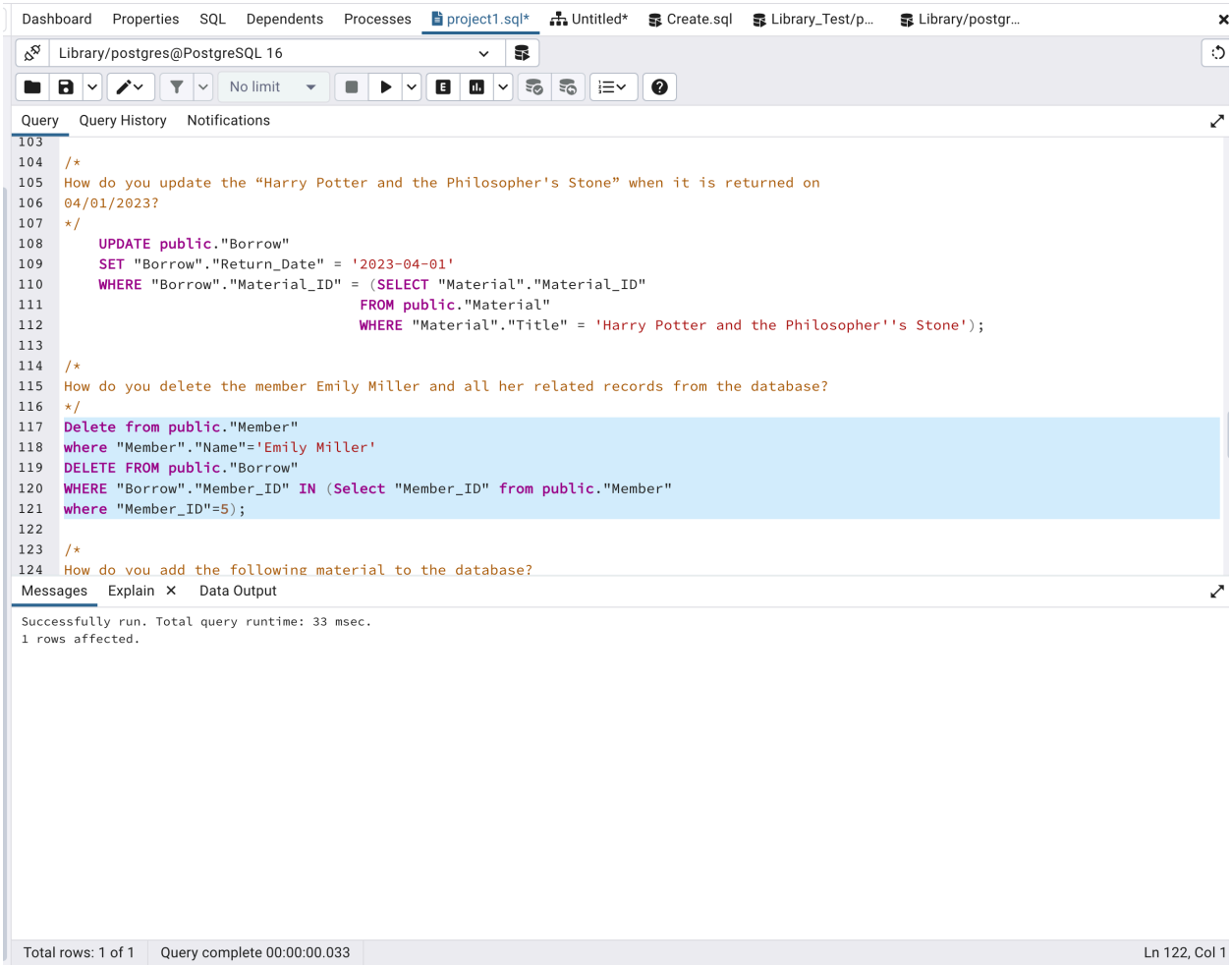
Q8. How do you update the “Harry Potter and the Philosopher's Stone” when it is returned on 04/01/2023?

```
UPDATE public."Borrow"
SET return_date = '2023-04-01'
WHERE Material_ID=(SELECT material_id from public."Material" m
WHERE m."title"='Harry Potter and the Philosopher''s Stone')
AND Return_Date IS NULL;

SELECT * from public."Borrow";
```



Q9. How do you delete the member Emily Miller and all her related records from the database?

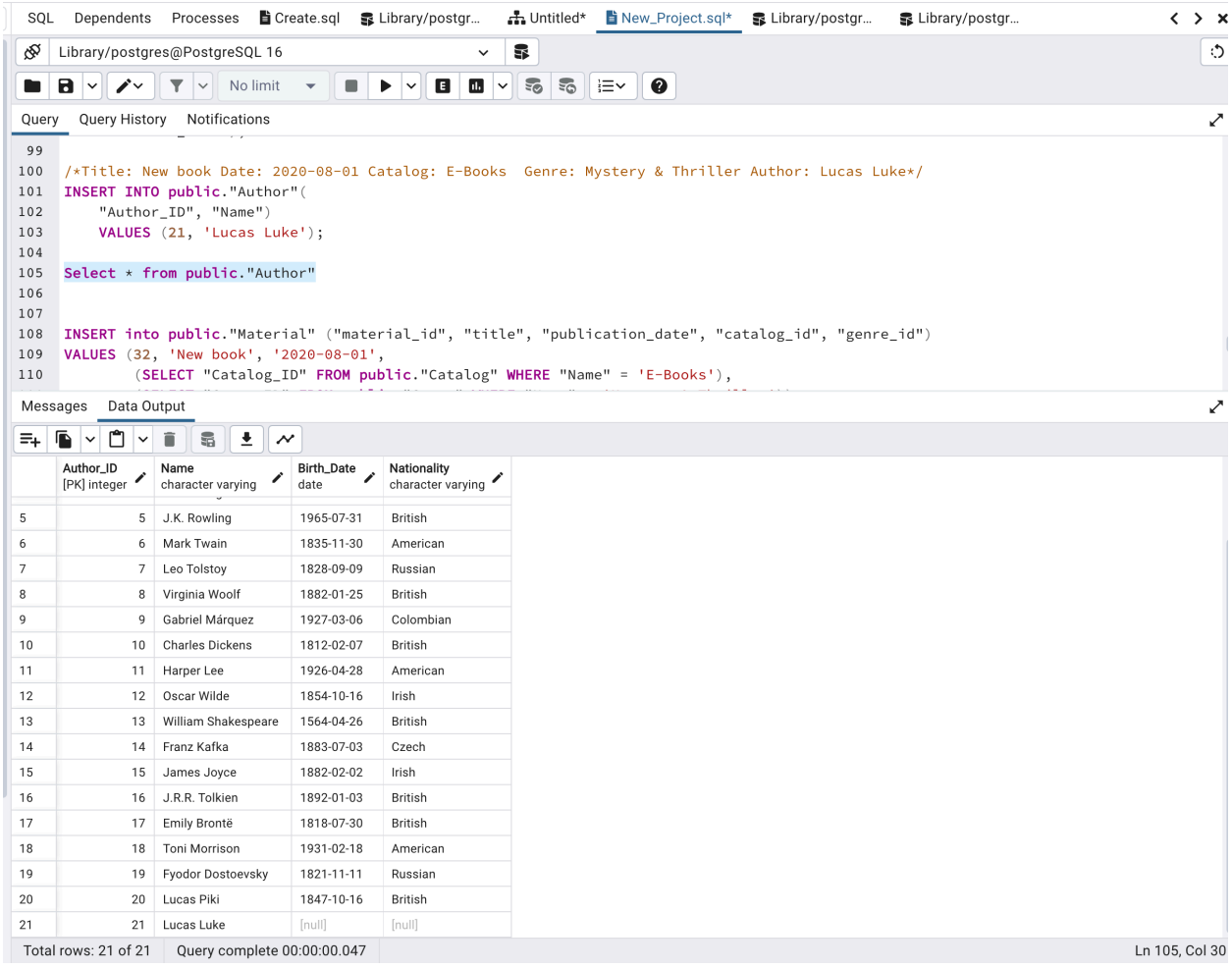
```
Delete from public."Member"
where "Member"."Name"='Emily Miller'
DELETE FROM public."Borrow"
WHERE "Borrow"."Member_ID" IN (Select "Member_ID" from
public."Member"
where "Member_ID"=5);
```



Q10. How do you add the following material to the database? Title: New book Date: 2020-08-01 Catalog: E-Books13 Genre: Mystery & Thriller Author: Lucas Luke

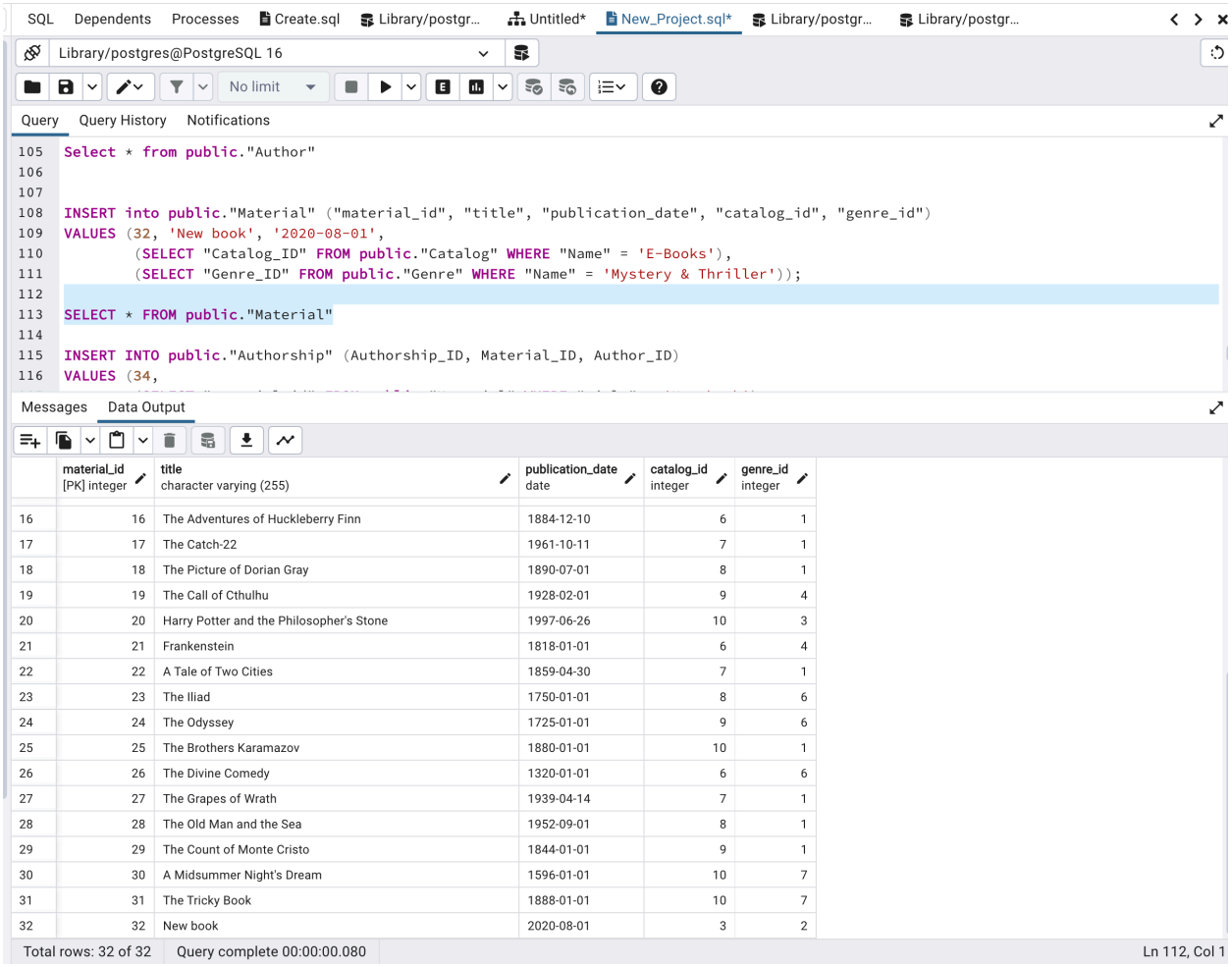
First I have loaded Author name into the author table

```
INSERT INTO public."Author"(
     "Author_ID", "Name")
     VALUES (21, 'Lucas Luke');

Select * from public."Author"
```



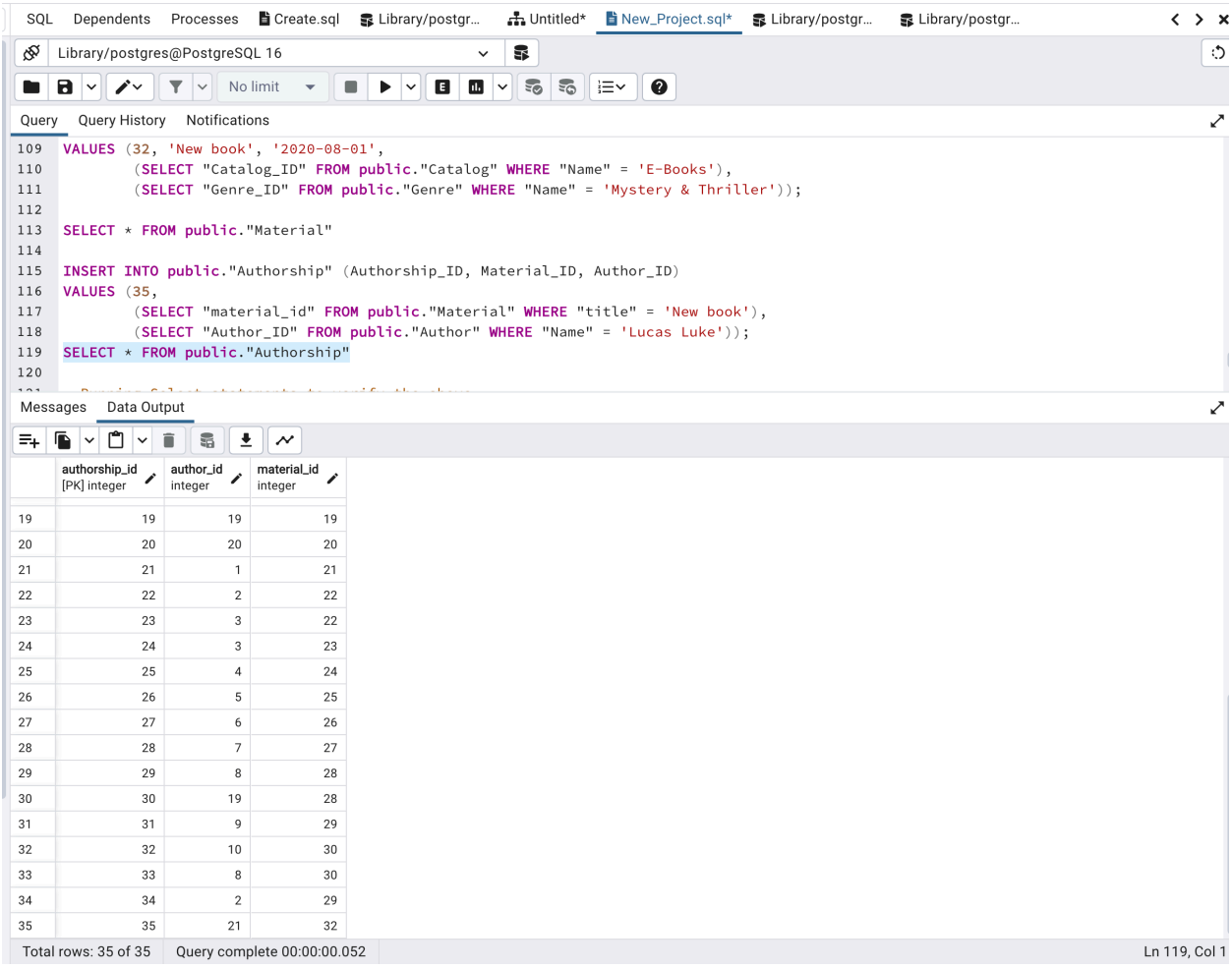
As we can see from the above screenshot “Lucas Luke” has been updated as 21st record in Author Table.

```
INSERT into public."Material" ("material_id", "title",
"publication_date", "catalog_id", "genre_id")
VALUES (32, 'New book', '2020-08-01',
         (SELECT "Catalog_ID" FROM public."Catalog" WHERE
"Name" = 'E-Books'),
         (SELECT "Genre_ID" FROM public."Genre" WHERE "Name" =
'Mystery & Thriller'));

SELECT * FROM public."Material"
```



From the above screenshot we can see that we have inserted 32nd row with title as ‘New Book’ with Publication Date="2020-08-01"

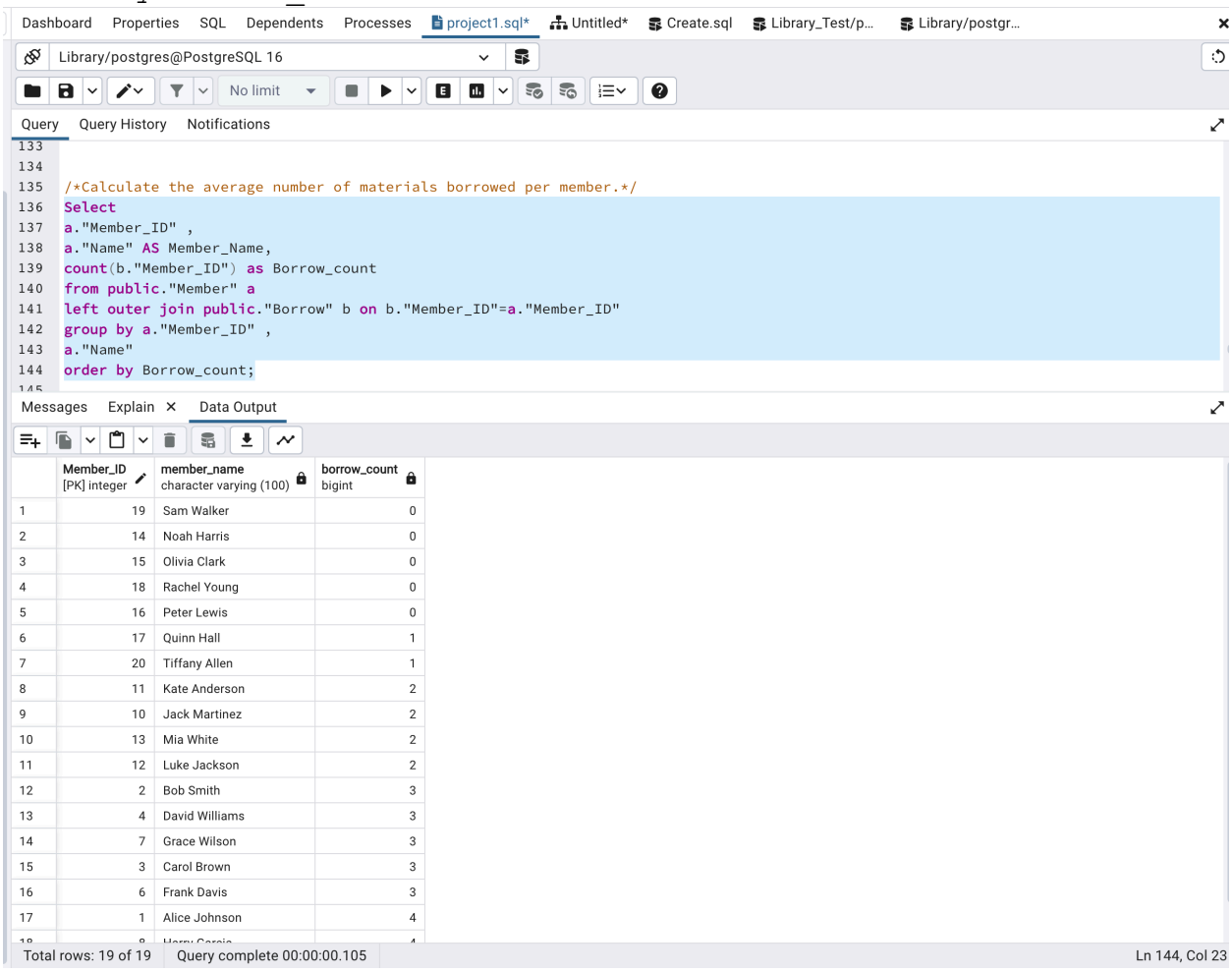
```
INSERT INTO public."Authorship" (Authorship_ID, Material_ID,
Author_ID)
VALUES (35,
          (SELECT "material_id" FROM public."Material" WHERE
"title" = 'New book'),
          (SELECT "Author_ID" FROM public."Author" WHERE "Name"
= 'Lucas Luke'));
SELECT * FROM public."Authorship"
```



As we can see last row has been updated with all the three id's

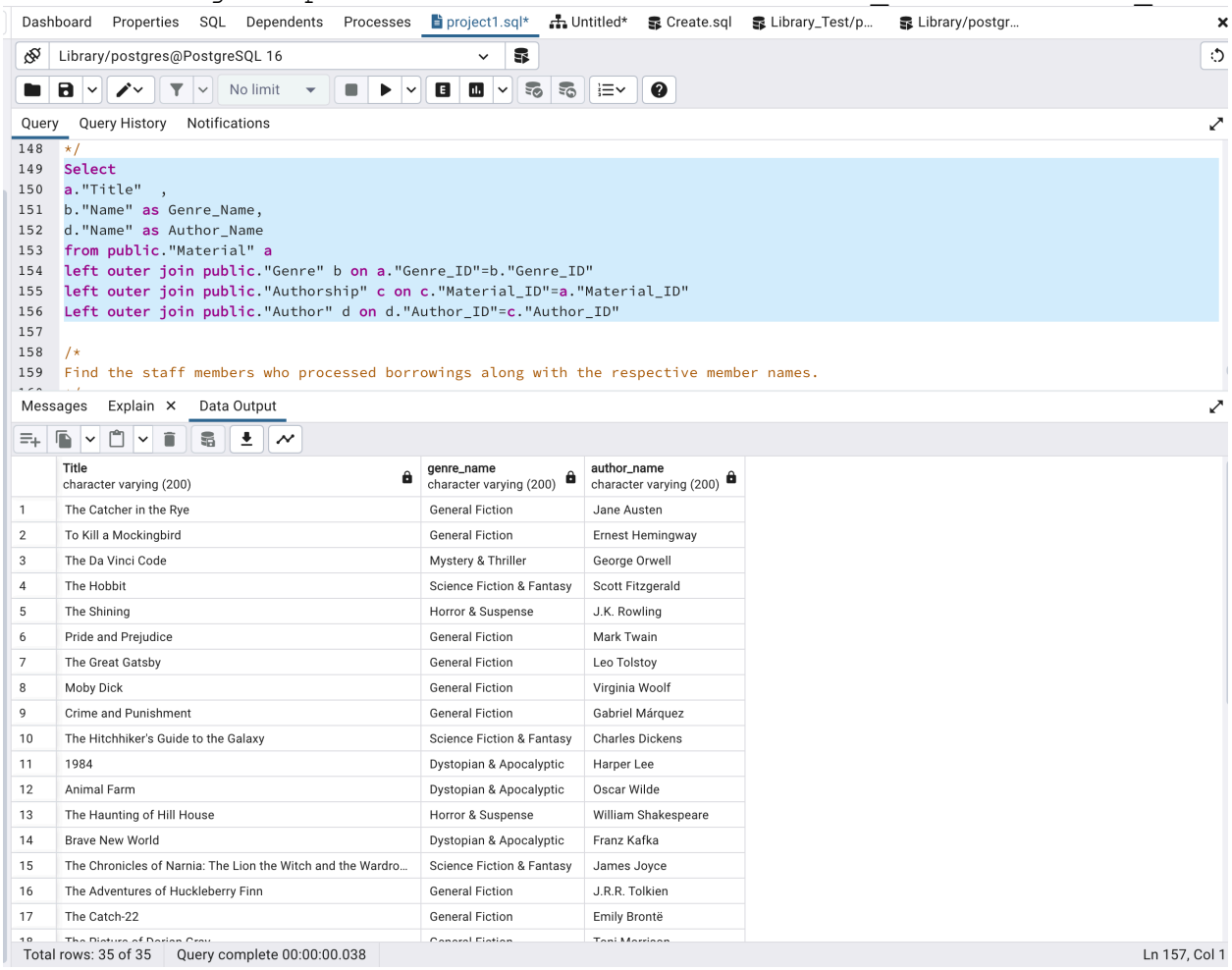
Q11. Calculate the average number of materials borrowed per member.

```
Select
a."Member_ID" ,
a."Name" AS Member_Name,
count(b."Member_ID") as Borrow_count
from public."Member" a
left outer join public."Borrow" b on b."Member_ID"=a."Member_ID"
group by a."Member_ID" ,
a."Name"
order by Borrow_count;
```



Q12. List materials with their genres and the names of authors who created them.

```
Select
a."Title"  ,
b."Name" as Genre_Name,
d."Name" as Author_Name
from public."Material" a
left outer join public."Genre" b on a."Genre_ID"=b."Genre_ID"
left outer join public."Authorship" c on
c."Material_ID"=a."Material_ID"
Left outer join public."Author" d on d."Author_ID"=c."Author_ID"
```



Future Work:

1. Alert staff about overdue materials on a daily-basis?

To alert staff about overdue materials , we have to create table called as alert_staff

```
CREATE TABLE Alert_staff (
    Allert_ID SERIAL PRIMARY KEY,
    Material_ID INTEGER,
    Member_ID INTEGER,
    Staff_ID INTEGER,
    Alert_Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Created a stored procedure to generate alert and insert into alert_staff table, so that we will get to know when we run select query for alert staff table.

```
CREATE OR REPLACE FUNCTION generate_overdue_alert()
RETURNS VOID AS $$
BEGIN
    INSERT INTO Alert_staff (Material_ID, Member_ID, Staff_ID)
    SELECT b."material_id",b."member_id",b."staff_id"
    FROM public."Borrow" b
    WHERE b."return_date" IS NULL AND b."due_date" <
CURRENT_DATE;
END;
$$ LANGUAGE plpgsql;
```

Using pg_notify(it is asynchronous feature of pg admin) we have created a function called schedule daily alert

```
CREATE OR REPLACE FUNCTION schedule_daily_alerts()
RETURNS TRIGGER AS $$
BEGIN
    PERFORM pg_notify('alert_staff', 'generate_overdue_alert');
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Created trigger which it inserts data into alert_staff table whenever there is overdue of materials

```
CREATE TRIGGER daily_notification_alert
AFTER INSERT ON public."Alert_staff"
EXECUTE FUNCTION schedule_daily_notifications();
```

2. Automatically deactivate the membership based on the member's overdue occurrence (>=three times). And reactivate the membership once the member pays the overdue fee.

We have added new column into member to get membership_status

```
ALTER TABLE public."Member"
ADD COLUMN membership_status BOOLEAN DEFAULT TRUE ;
```

We have created new table Payments which will store the payment information for each member.

```
CREATE TABLE public."payments" (
    payment_id SERIAL PRIMARY KEY,
    member_id INT REFERENCES public."Member"("Member_ID"),
    payment_date DATE,
    payment_amount DECIMAL(10, 2)
);
```

The update_membership_function is a trigger function that will automatically run after an insert or update operation on the member and payments tables.

```
CREATE OR REPLACE FUNCTION update_membership_status() RETURNS
TRIGGER AS $$
BEGIN

    IF NEW.overdue_occurrences >= 3 THEN
        UPDATE members SET membership_status = FALSE WHERE id =
NEW.id;
    END IF;

    IF TG_TABLE_NAME = 'payments' THEN
```

```
        UPDATE members SET membership_status = TRUE WHERE id =
NEW.member_id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Created two triggers to check the status and payment

```
CREATE TRIGGER check_membership_status
AFTER INSERT OR UPDATE ON public."Member"
FOR EACH ROW EXECUTE FUNCTION update_membership_status();

CREATE TRIGGER check_payment
AFTER INSERT ON public."payments"
FOR EACH ROW EXECUTE FUNCTION update_membership_status();
```

INSTRUCTIONS TO RUN THE CODE

1. Create Database in Pg Admin .
2. Create all the tables using create statements mentioned in the table queries section.
3. Now Insert values using 'Import/Export Data' wizard for the standalone tables which do not have any constraints like 'Author','Genre','Catalog','Member','Staff' and later insert into 'Material' ,'Borrow' and 'Authorship' table as they have foreign key constraints . Follow this sequence respectively.
4. Now Execute the queries as mentioned in Queries section.