SQL ASSIGNMENT

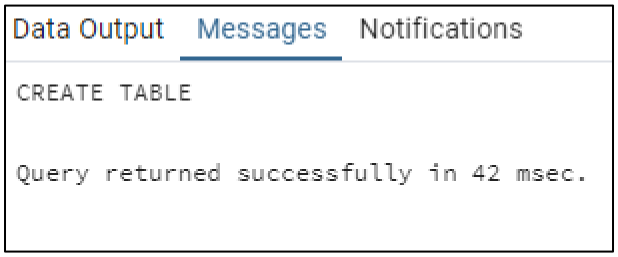
CHAPTER1

1. Write a query to create a simple table **Movies** including columns **movie_id, movie_name, release_year, rental_rate**.

   QUERY:

   ```
   CREATE TABLE Movies (
   movie_id INT,
   movie_name varchar (255),
   release_year INT,
   rental_rate INT
   );
   ```

   OUTPUT:

   Data Output Messages Notifications

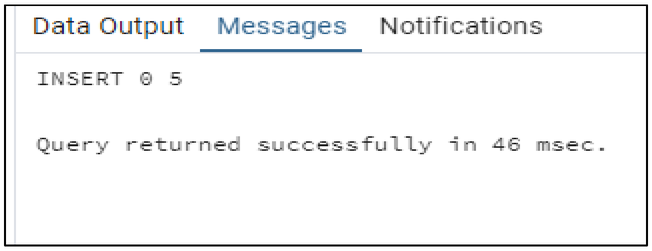
   ```
   CREATE TABLE

   Query returned successfully in 42 msec.
   ```

2. Write a query to insert 5 records with your own value into the table **Movies** against each column. The **release_year** must be 2016, 2017, 2022, 2023, 2025.

   QUERY:

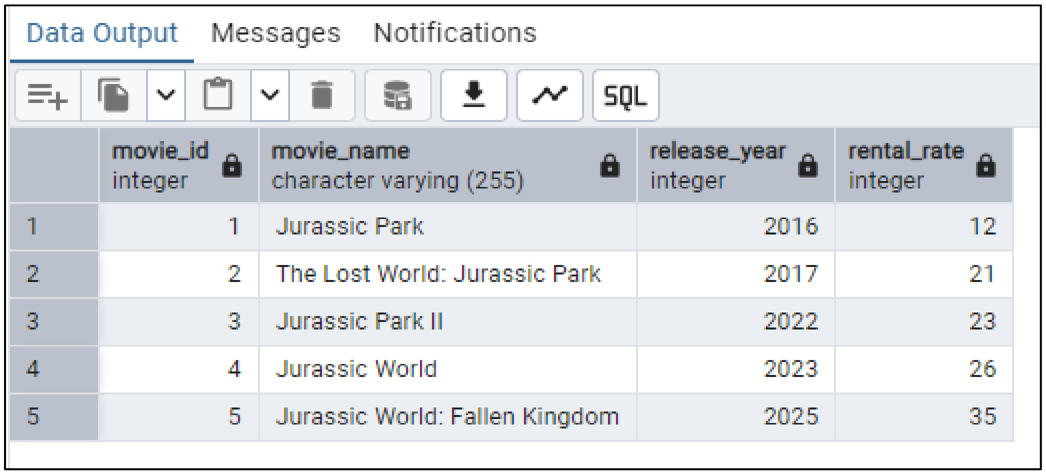
   ```
   INSERT INTO Movies
   VALUES
   (1, 'Jurassic Park', 2016, 12.34),
   (2, 'The Lost World: Jurassic Park', 2017, 21.23),
   (3, 'Jurassic Park II', 2022, 23.45),
   (4, 'Jurassic World', 2023, 25.69),
   (5, 'Jurassic World: Fallen Kingdom', 2025, 34.56);
   ```

   OUTPUT:

   Data Output Messages Notifications

   ```
   INSERT 0 5

   Query returned successfully in 46 msec.
   ```


Data Output Messages Notifications

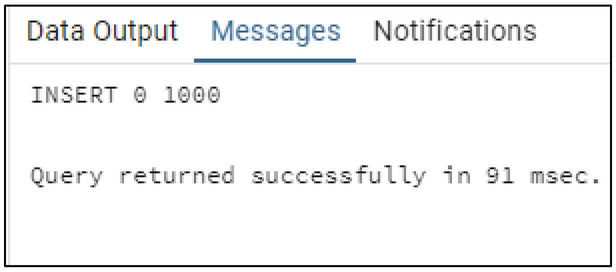
	movie_id integer	movie_name character varying (255)	release_year integer	rental_rate integer
1	1	Jurassic Park	2016	12
2	2	The Lost World: Jurassic Park	2017	21
3	3	Jurassic Park II	2022	23
4	4	Jurassic World	2023	26
5	5	Jurassic World: Fallen Kingdom	2025	35

3. Write a query to insert rows from **film** table to **Movies** table.

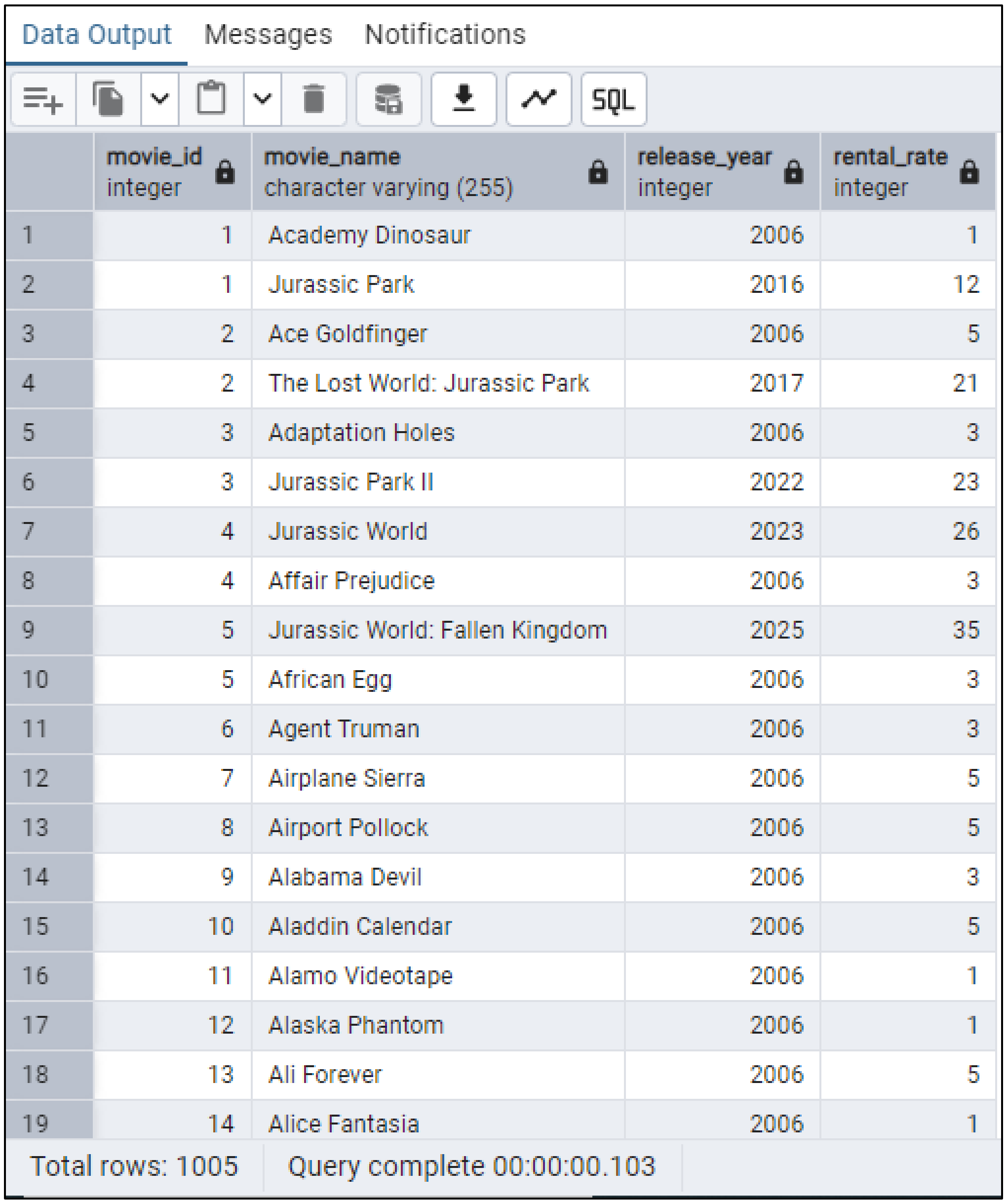
QUERY:

**INSERT INTO Movies (movie_id, movie_name, release_year, rental_rate)
SELECT film_id, title, release_year, rental_rate FROM film;**

OUTPUT:



```
INSERT 0 1000

Query returned successfully in 91 msec.
```


Data Output Messages Notifications

	movie_id integer	movie_name character varying (255)	release_year integer	rental_rate integer
1	1	Academy Dinosaur	2006	1
2	1	Jurassic Park	2016	12
3	2	Ace Goldfinger	2006	5
4	2	The Lost World: Jurassic Park	2017	21
5	3	Adaptation Holes	2006	3
6	3	Jurassic Park II	2022	23
7	4	Jurassic World	2023	26
8	4	Affair Prejudice	2006	3
9	5	Jurassic World: Fallen Kingdom	2025	35
10	5	African Egg	2006	3
11	6	Agent Truman	2006	3
12	7	Airplane Sierra	2006	5
13	8	Airport Pollock	2006	5
14	9	Alabama Devil	2006	3
15	10	Aladdin Calendar	2006	5
16	11	Alamo Videotape	2006	1
17	12	Alaska Phantom	2006	1
18	13	Ali Forever	2006	5
19	14	Alice Fantasia	2006	1

Total rows: 1005 Query complete 00:00:00.103

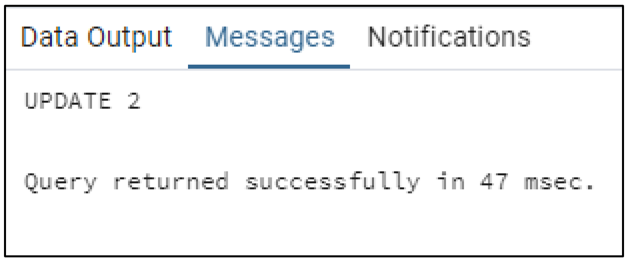
4. Write a query to update the **rental_rate** to 9999 for those **Movies** whose **release_year** is greater than 2022.

QUERY:

UPDATE Movies SET rental_rate=9999 WHERE release_year>2022;

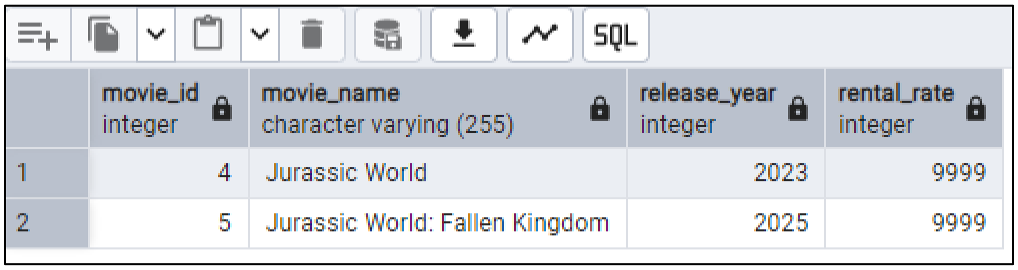
SELECT * FROM Movies WHERE rental_rate = 9999;

OUTPUT:



Data Output | Messages | Notifications

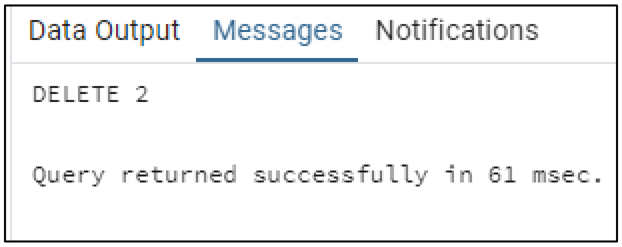
```
UPDATE 2

Query returned successfully in 47 msec.
```



	movie_id integer	movie_name character varying (255)	release_year integer	rental_rate integer
1	4	Jurassic World	2023	9999
2	5	Jurassic World: Fallen Kingdom	2025	9999

5. Delete the rows from the **Movies** table where the **rental_rate** is 9999.
QUERY:
DELETE from MOVIES WHERE rental_rate=9999;

OUTPUT:



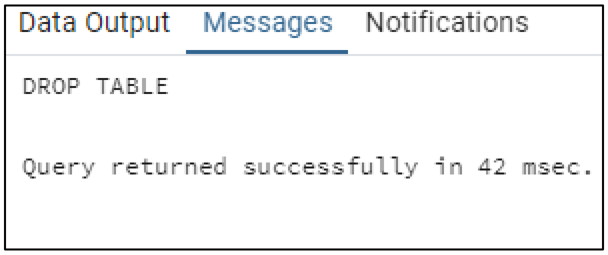
Data Output | Messages | Notifications

```
DELETE 2

Query returned successfully in 61 msec.
```

6. Drop the table **Movies**.
QUERY:

DROP TABLE Movies;

OUTPUT:



Data Output | Messages | Notifications

```
DROP TABLE

Query returned successfully in 42 msec.
```

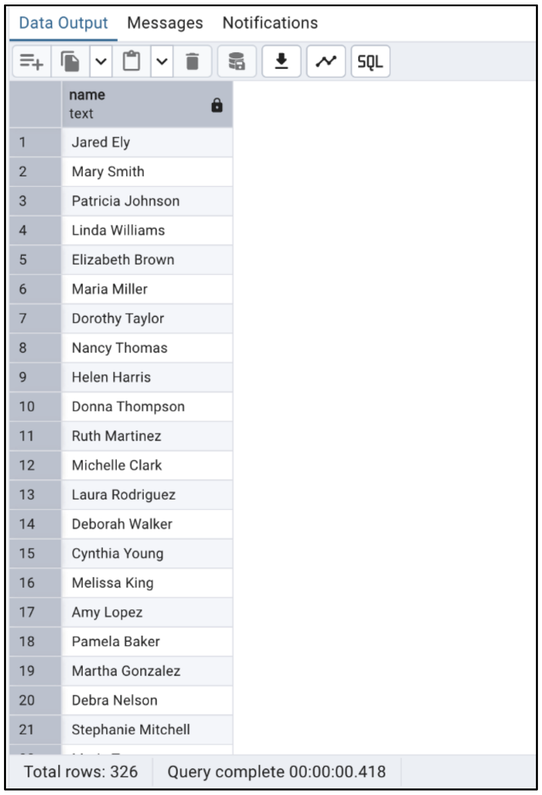
CHAPTER 2

1. Display all the names of the customers with store_id = 1.

QUERY:

select CONCAT(first_name,' ',last_name) AS NAME
from customer where store_id = 1;

OUTPUT:



	name text
1	Jared Ely
2	Mary Smith
3	Patricia Johnson
4	Linda Williams
5	Elizabeth Brown
6	Maria Miller
7	Dorothy Taylor
8	Nancy Thomas
9	Helen Harris
10	Donna Thompson
11	Ruth Martinez
12	Michelle Clark
13	Laura Rodriguez
14	Deborah Walker
15	Cynthia Young
16	Melissa King
17	Amy Lopez
18	Pamela Baker
19	Martha Gonzalez
20	Debra Nelson
21	Stephanie Mitchell

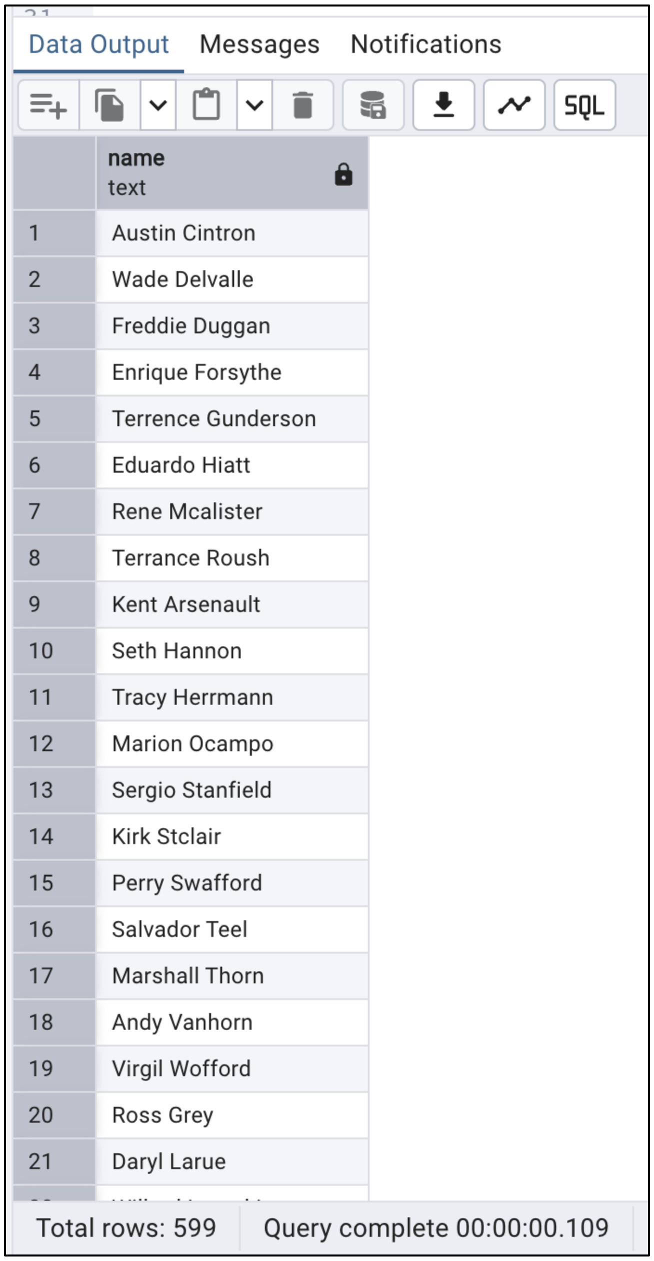
Total rows: 326 Query complete 00:00:00.418

2. Display all the customer names ordered in descending order of their address.

QUERY:

SELECT CONCAT(first_name,' ',last_name) AS NAME
FROM customer ORDER BY address_id DESC;

OUTPUT:


Data Output Messages Notifications

	name text
1	Austin Cintron
2	Wade Delvalle
3	Freddie Duggan
4	Enrique Forsythe
5	Terrence Gunderson
6	Eduardo Hiatt
7	Rene Mcalister
8	Terrance Roush
9	Kent Arsenault
10	Seth Hannon
11	Tracy Herrmann
12	Marion Ocampo
13	Sergio Stanfield
14	Kirk Stclair
15	Perry Swafford
16	Salvador Teel
17	Marshall Thorn
18	Andy Vanhorn
19	Virgil Wofford
20	Ross Grey
21	Daryl Larue

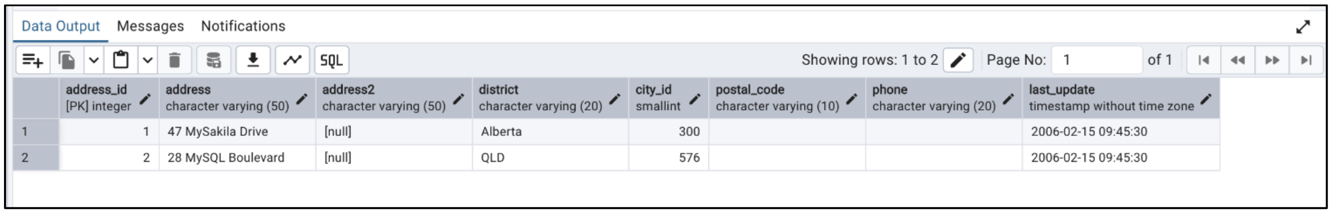
Total rows: 599 Query complete 00:00:00.109

3. Display the addresses whose phone number is empty.

<u>**QUERY**</u>:

SELECT *
FROM address
WHERE phone = ' ';

OUTPUT:



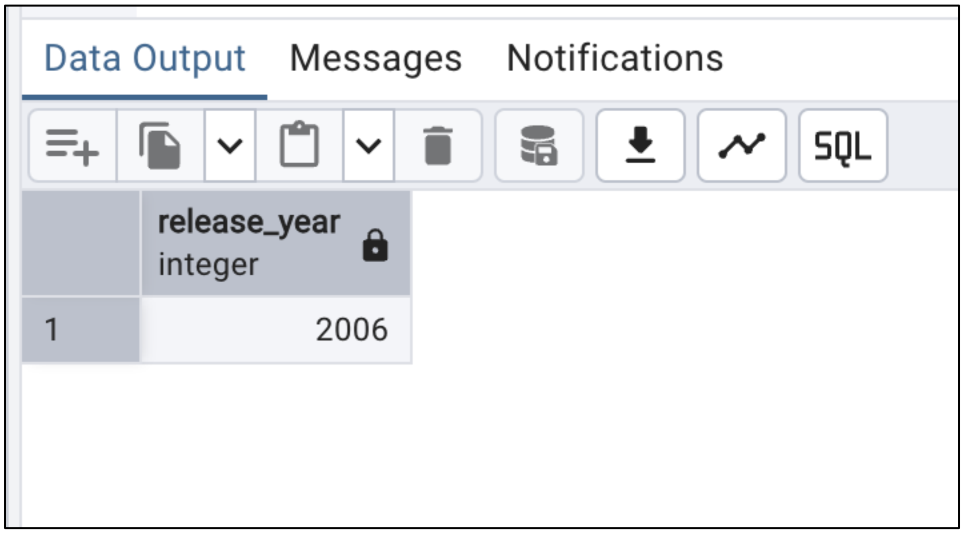
	address_id [PK] integer	address character varying (50)	address2 character varying (50)	district character varying (20)	city_id smallint	postal_code character varying (10)	phone character varying (20)	last_update timestamp without time zone
1	1	47 MySakila Drive	[null]	Alberta	300			2006-02-15 09:45:30
2	2	28 MySQL Boulevard	[null]	QLD	576			2006-02-15 09:45:30

4. **Display all the distinct years of the movie released from the films.**

QUERY:

```
SELECT DISTINCT(release_year)
FROM film;
```

OUTPUT:



	release_year integer
1	2006

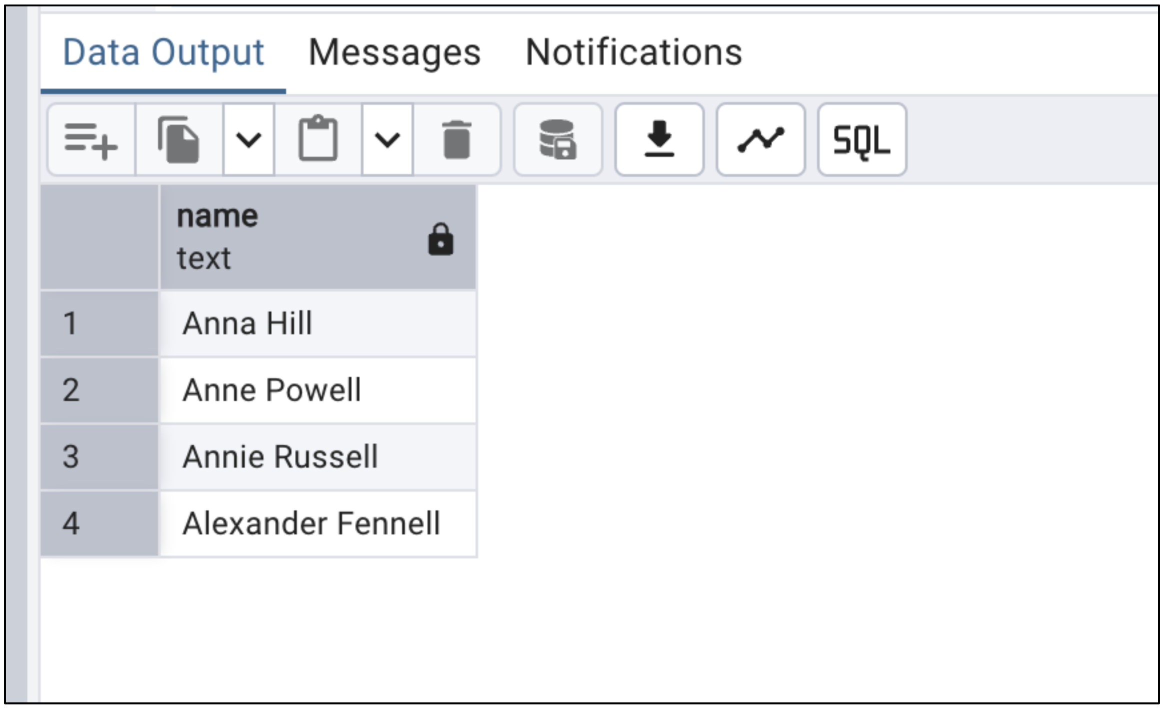
5. Display the names of the customers whose first names start with A and last name ends with ‘l’.

QUERY:

```
SELECT CONCAT (first_name,' ',last_name) AS NAME
```

FROM customer
WHERE first_name LIKE 'A%' AND last_name LIKE '%l';

OUTPUT:



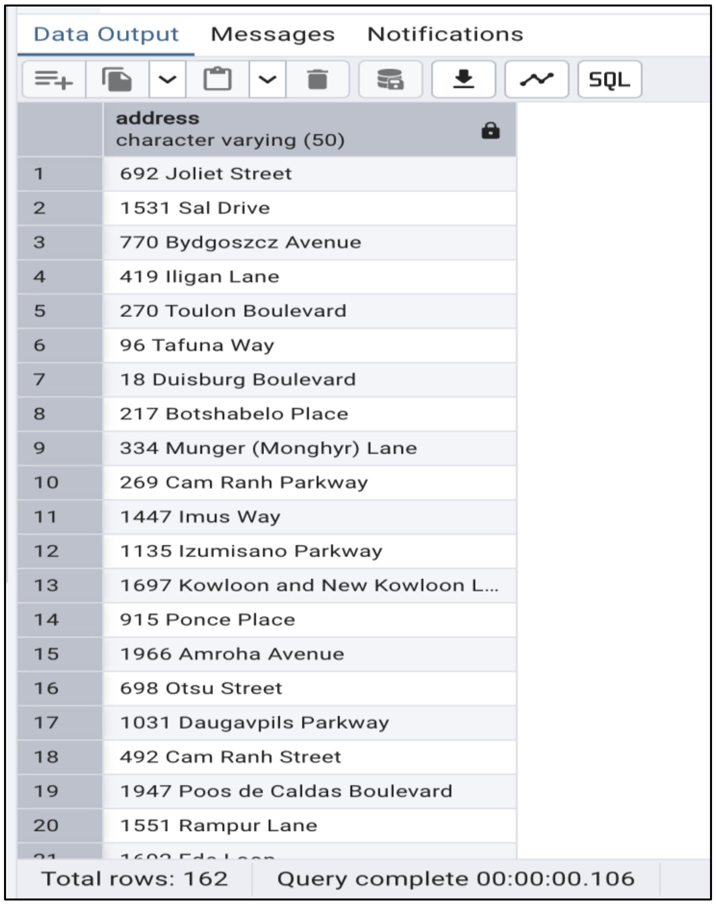
	name text
1	Anna Hill
2	Anne Powell
3	Annie Russell
4	Alexander Fennell

6. Display the address whose city id is in between 30 and 190.

QUERY:

SELECT address from address WHERE city_id BETWEEN 30 AND 190;

OUTPUT:



Data Output Messages Notifications

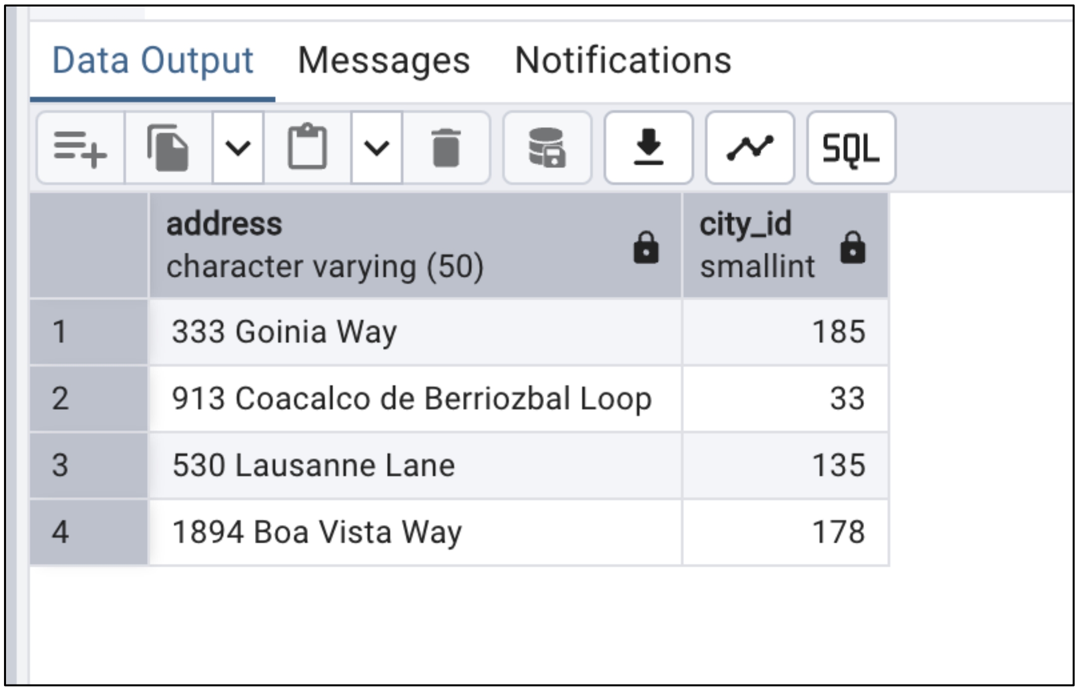
	address character varying (50)
1	692 Joliet Street
2	1531 Sal Drive
3	770 Bydgoszcz Avenue
4	419 Iligan Lane
5	270 Toulon Boulevard
6	96 Tafuna Way
7	18 Duisburg Boulevard
8	217 Botshabelo Place
9	334 Munger (Monghyr) Lane
10	269 Cam Ranh Parkway
11	1447 Imus Way
12	1135 Izumisano Parkway
13	1697 Kowloon and New Kowloon L...
14	915 Ponce Place
15	1966 Amroha Avenue
16	698 Otsu Street
17	1031 Daugavpils Parkway
18	492 Cam Ranh Street
19	1947 Poos de Caldas Boulevard
20	1551 Rampur Lane

Total rows: 162 Query complete 00:00:00.106

7. Display the address whose city id is in between 30 and 190 and whose district is Texas.

QUERY:
SELECT address, city_id from address WHERE (city_id BETWEEN 30 and 190) AND DISTRICT='Texas';

OUTPUT:


Data Output Messages Notifications

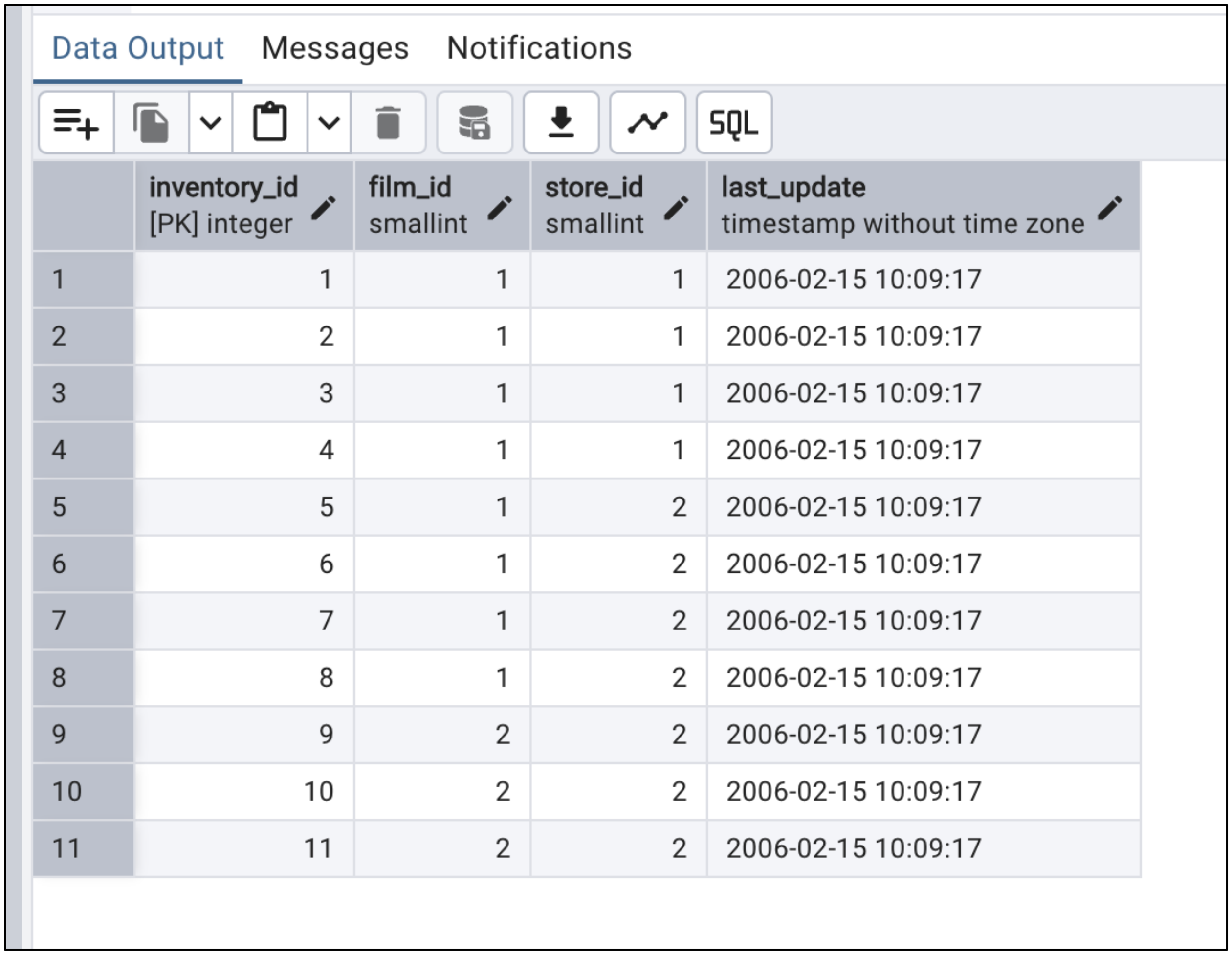
	address character varying (50)	city_id smallint
1	333 Goinia Way	185
2	913 Coacalco de Berriozbal Loop	33
3	530 Lausanne Lane	135
4	1894 Boa Vista Way	178

8. Display all the inventories whose film id is 1 or 2.

   QUERY:

   ```
   SELECT * FROM inventory
   WHERE film_id = 1 OR film_id = 2;
   ```

   OUTPUT:



Data Output Messages Notifications

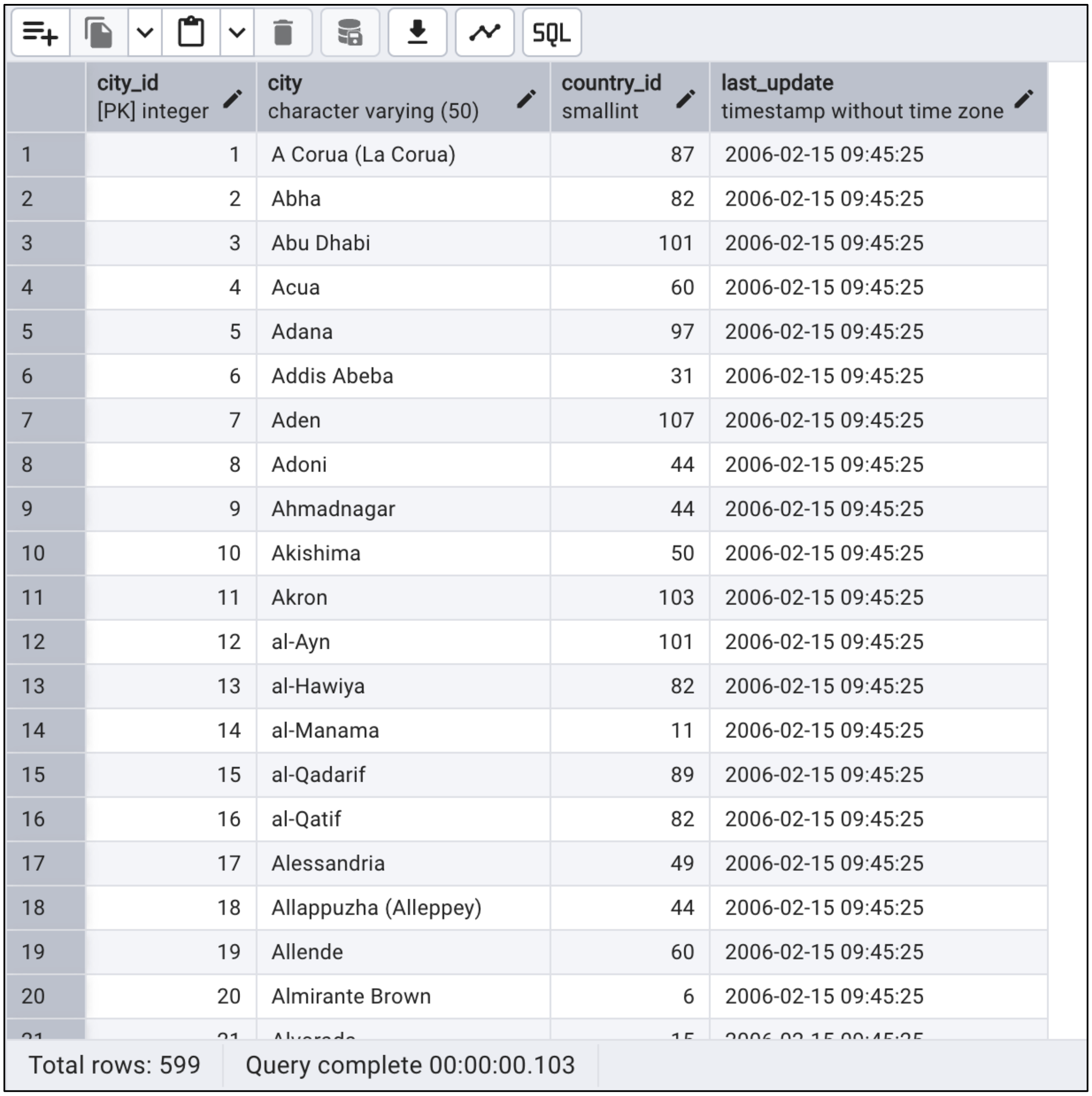
	inventory_id [PK] integer	film_id smallint	store_id smallint	last_update timestamp without time zone
1	1	1	1	2006-02-15 10:09:17
2	2	1	1	2006-02-15 10:09:17
3	3	1	1	2006-02-15 10:09:17
4	4	1	1	2006-02-15 10:09:17
5	5	1	2	2006-02-15 10:09:17
6	6	1	2	2006-02-15 10:09:17
7	7	1	2	2006-02-15 10:09:17
8	8	1	2	2006-02-15 10:09:17
9	9	2	2	2006-02-15 10:09:17
10	10	2	2	2006-02-15 10:09:17
11	11	2	2	2006-02-15 10:09:17

9. Display all the records whose city_id is not 200.

QUERY:

```
SELECT * FROM city
 WHERE city_id != 200;
```

OUTPUT:



	city_id [PK] integer	city character varying (50)	country_id smallint	last_update timestamp without time zone
1	1	A Corua (La Corua)	87	2006-02-15 09:45:25
2	2	Abha	82	2006-02-15 09:45:25
3	3	Abu Dhabi	101	2006-02-15 09:45:25
4	4	Acua	60	2006-02-15 09:45:25
5	5	Adana	97	2006-02-15 09:45:25
6	6	Addis Abeba	31	2006-02-15 09:45:25
7	7	Aden	107	2006-02-15 09:45:25
8	8	Adoni	44	2006-02-15 09:45:25
9	9	Ahmadnagar	44	2006-02-15 09:45:25
10	10	Akishima	50	2006-02-15 09:45:25
11	11	Akron	103	2006-02-15 09:45:25
12	12	al-Ayn	101	2006-02-15 09:45:25
13	13	al-Hawiya	82	2006-02-15 09:45:25
14	14	al-Manama	11	2006-02-15 09:45:25
15	15	al-Qadarif	89	2006-02-15 09:45:25
16	16	al-Qatif	82	2006-02-15 09:45:25
17	17	Alessandria	49	2006-02-15 09:45:25
18	18	Allappuzha (Alleppey)	44	2006-02-15 09:45:25
19	19	Allende	60	2006-02-15 09:45:25
20	20	Almirante Brown	6	2006-02-15 09:45:25
21	21	Alvarado	15	2006-02-15 09:45:25

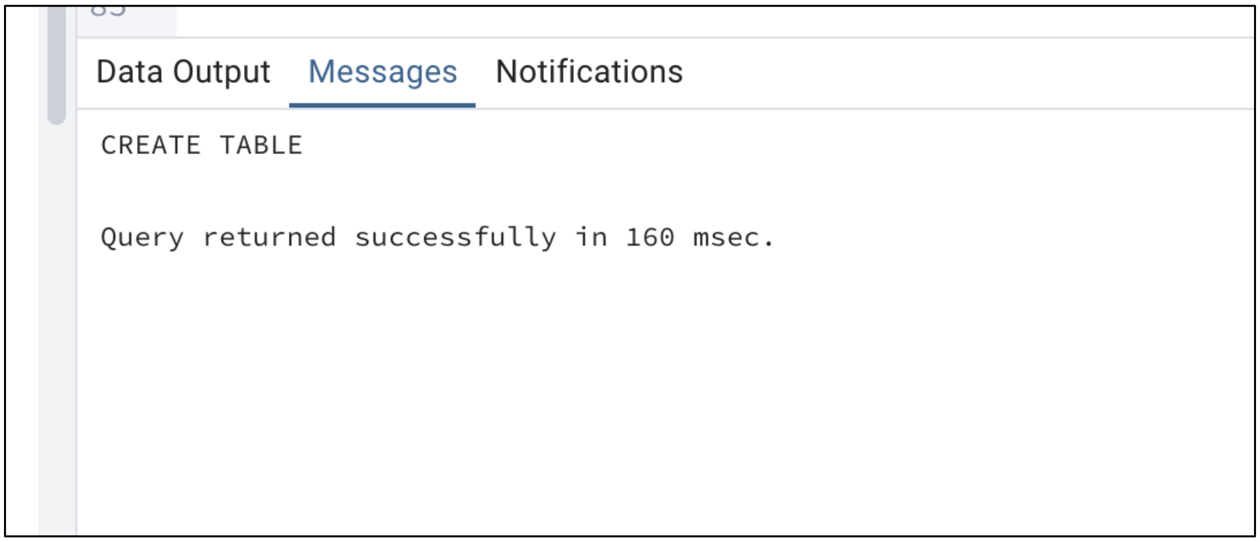
Total rows: 599 Query complete 00:00:00.103

10. Write a query to create a table Movies and set NOT NULL and PRIMARY KEY constraints for movie_name and movie_id.

QUERY:

```
CREATE TABLE Movies

(

movie_id integer NOT NULL,

 movie_name varchar(255) NOT NULL,

  PRIMARY KEY(movie_id,movie_name )

 );
```

OUTPUT:



```
CREATE TABLE

Query returned successfully in 160 msec.
```

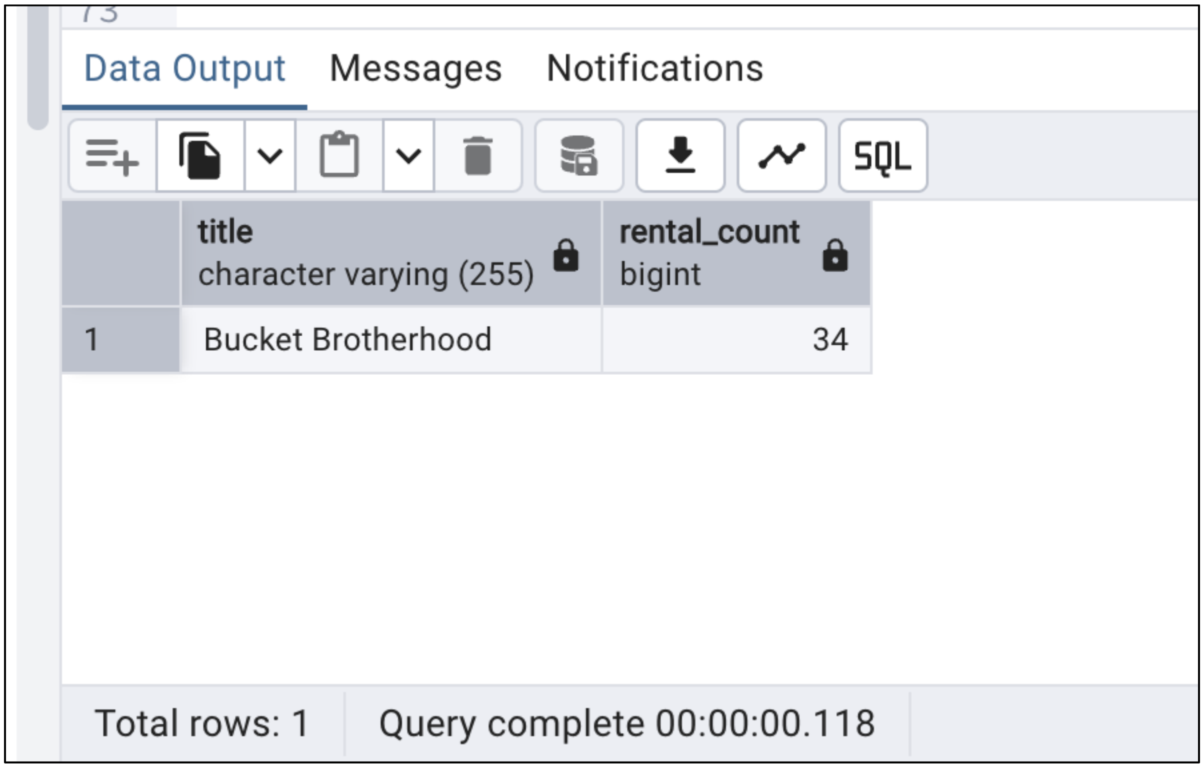
CHAPTER 3

1. What is the movie(s) that was rented the most.

   QUERY:

```
SELECT DISTINCT title, COUNT(rental_id) AS rental_count
FROM film
JOIN inventory ON  film.film_id = inventory.film_id
JOIN rental ON  inventory.inventory_id = rental.inventory_id
GROUP BY film.title
ORDER BY rental_count DESC
LIMIT 1;
```

   OUTPUT:



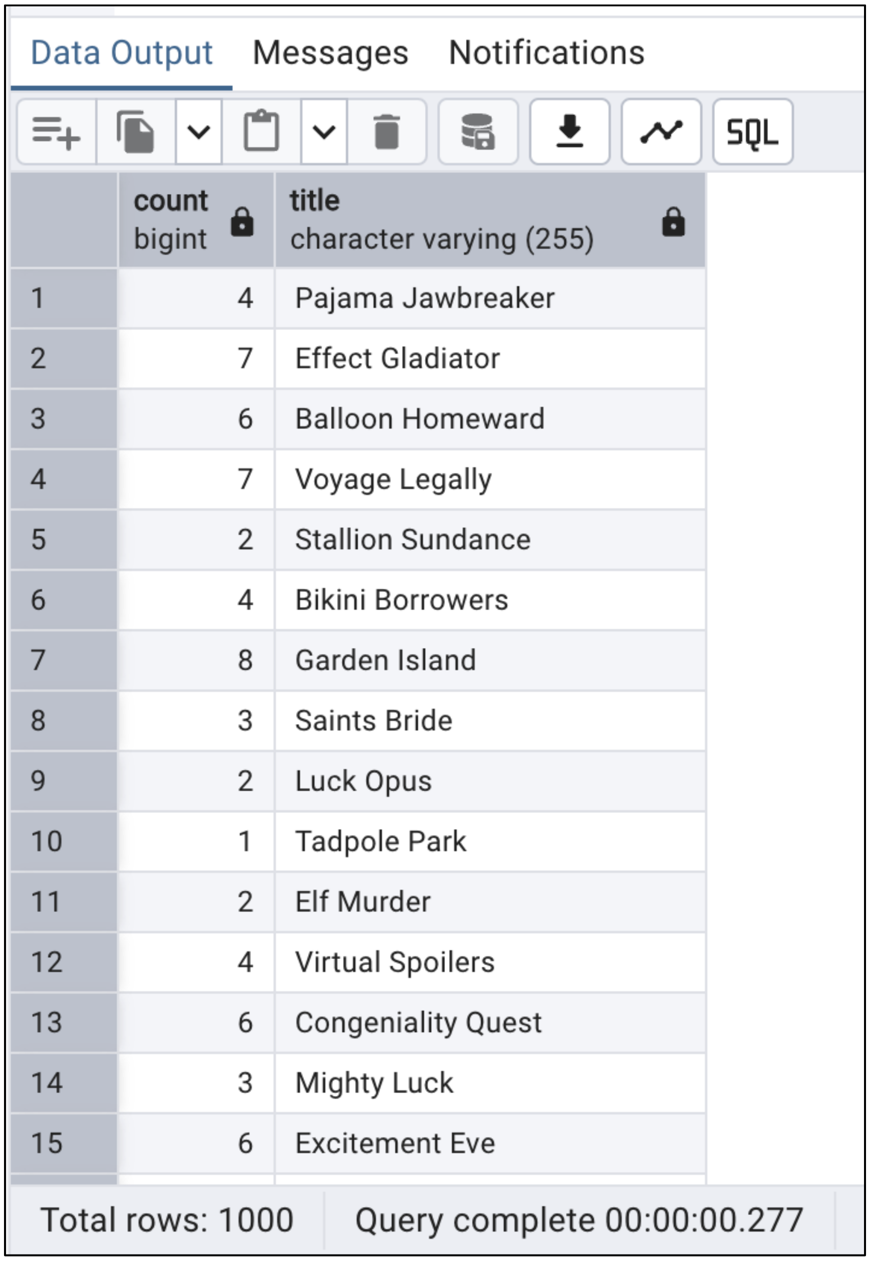
	title character varying (255)	rental_count bigint
1	Bucket Brotherhood	34

Total rows: 1   Query complete 00:00:00.118

2. Display each movie and the number of times it got rented.

QUERY:
Select title, Count(title) From film Group by title

OUTPUT:



	count bigint	title character varying (255)
1	4	Pajama Jawbreaker
2	7	Effect Gladiator
3	6	Balloon Homeward
4	7	Voyage Legally
5	2	Stallion Sundance
6	4	Bikini Borrowers
7	8	Garden Island
8	3	Saints Bride
9	2	Luck Opus
10	1	Tadpole Park
11	2	Elf Murder
12	4	Virtual Spoilers
13	6	Congeniality Quest
14	3	Mighty Luck
15	6	Excitement Eve

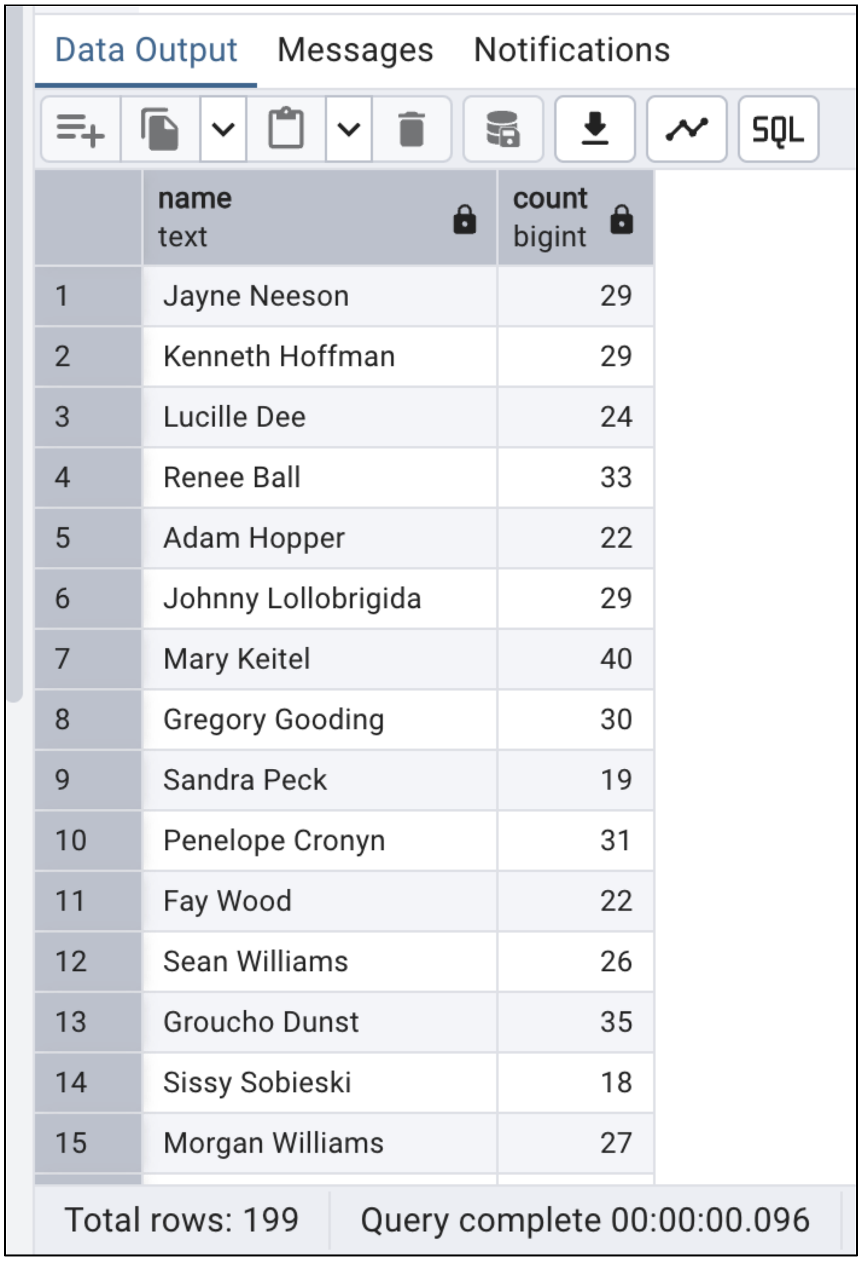
Total rows: 1000 Query complete 00:00:00.277

3. Show the number of movies each actor acted in.

QUERY:

select concat(a.first_name, ' ' ,a.last_name) as Name, Count(f.film_id)
from actor a, film_actor f
where a.actor_id = f.actor_id
group by Name;

OUTPUT:



Data Output Messages Notifications

	name text	count bigint
1	Jayne Neeson	29
2	Kenneth Hoffman	29
3	Lucille Dee	24
4	Renee Ball	33
5	Adam Hopper	22
6	Johnny Lollobrigida	29
7	Mary Keitel	40
8	Gregory Gooding	30
9	Sandra Peck	19
10	Penelope Cronyn	31
11	Fay Wood	22
12	Sean Williams	26
13	Groucho Dunst	35
14	Sissy Sobieski	18
15	Morgan Williams	27

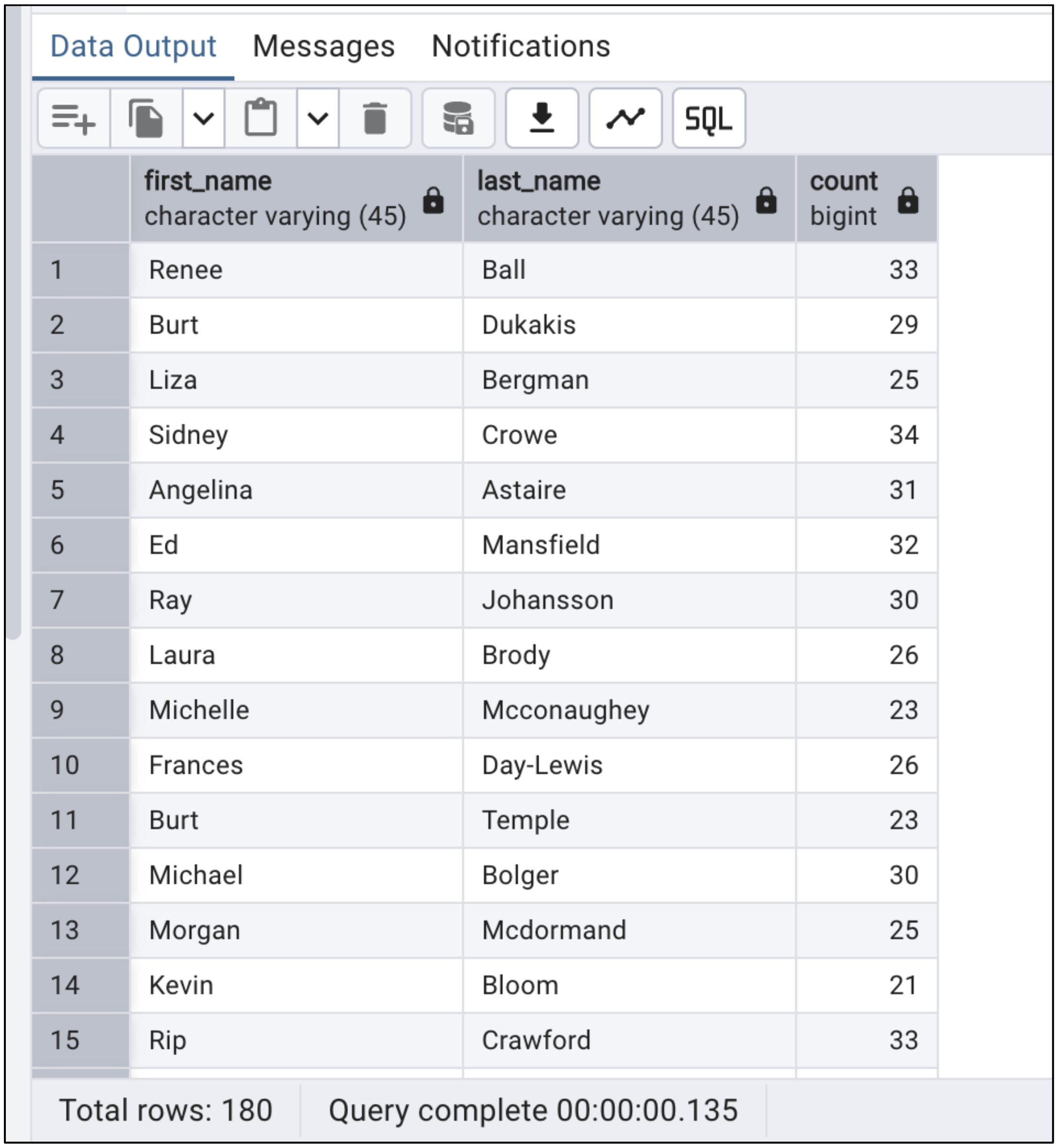
Total rows: 199 Query complete 00:00:00.096

4. Display the names of the actors that acted in more than 20 movies.

QUERY:

```
select concat(a.first_name, ' ' ,a.last_name) as Name, Count(f.film_id)
from actor a, film_actor f
where a.actor_id = f.actor_id
group by Name
Having Count(f.film_id) > 20
```

OUTPUT:



	first_name character varying (45)	last_name character varying (45)	count bigint
1	Renee	Ball	33
2	Burt	Dukakis	29
3	Liza	Bergman	25
4	Sidney	Crowe	34
5	Angelina	Astaire	31
6	Ed	Mansfield	32
7	Ray	Johansson	30
8	Laura	Brody	26
9	Michelle	Mcconaughey	23
10	Frances	Day-Lewis	26
11	Burt	Temple	23
12	Michael	Bolger	30
13	Morgan	Mcdormand	25
14	Kevin	Bloom	21
15	Rip	Crawford	33

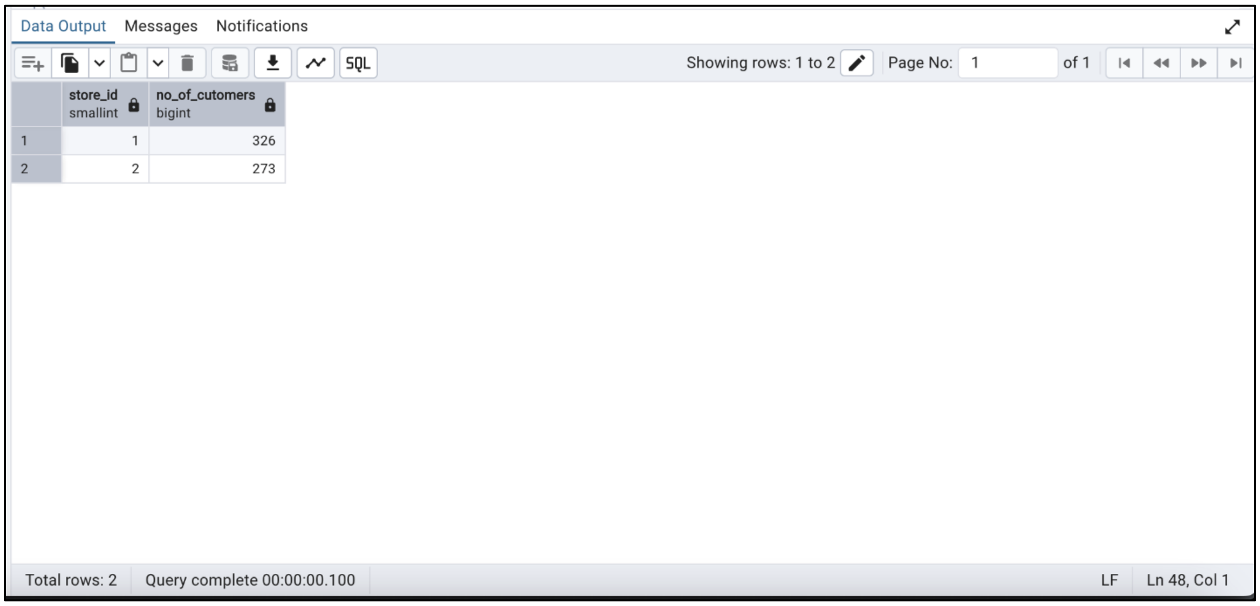
Total rows: 180 Query complete 00:00:00.135

5. For each store, display the number of customers that are members of that store.

QUERY

Select store_id, Count(customer_id)
from customer group by store_id

OUTPUT:



Data Output Messages Notifications

Showing rows: 1 to 2 Page No: 1 of 1

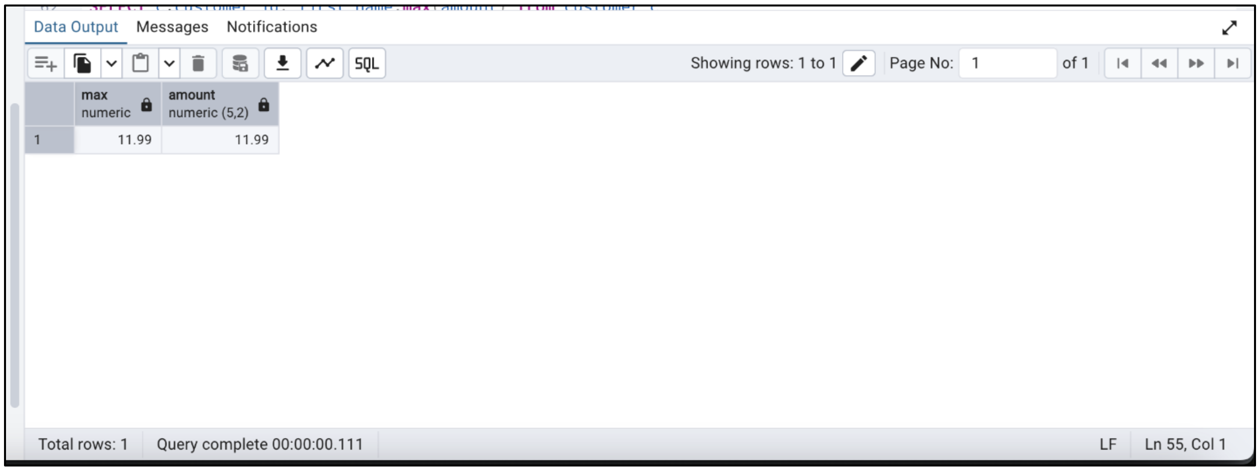
	store_id smallint	no_of_cutomers bigint
1	1	326
2	2	273

Total rows: 2 Query complete 00:00:00.100 LF Ln 48, Col 1

6. What is the highest total_payment done.

QUERY:
select MAX(amount) from payment;

OUTPUT:


Data Output Messages Notifications

Showing rows: 1 to 1 Page No: 1 of 1

	max numeric	amount numeric (5,2)
1	11.99	11.99

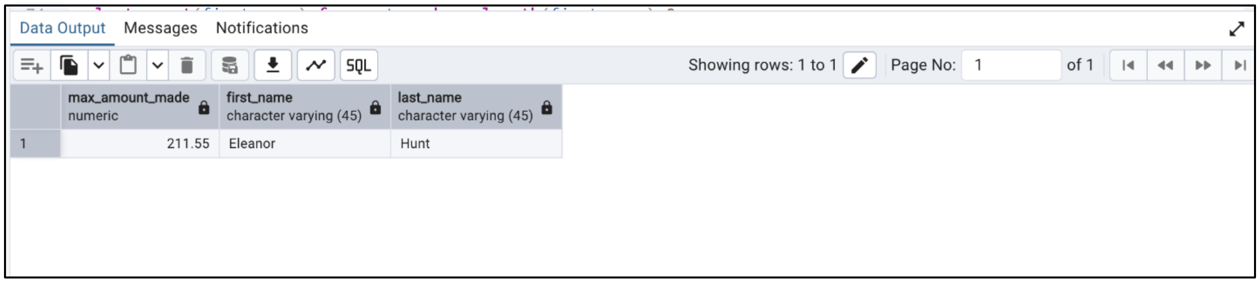
Total rows: 1 Query complete 00:00:00.111 LF Ln 55, Col 1

7. What is the name of the customer who made the highest total payments.

QUERY:

select Concat(c.first_name, ' ',c.last_name) as Name, SUM(p.amount)
from customer as c, payment p
where c.customer_id = p.customer_id
group by Name
Order by SUM(p.amount) desc
limit 1;

OUTPUT:


Data Output Messages Notifications

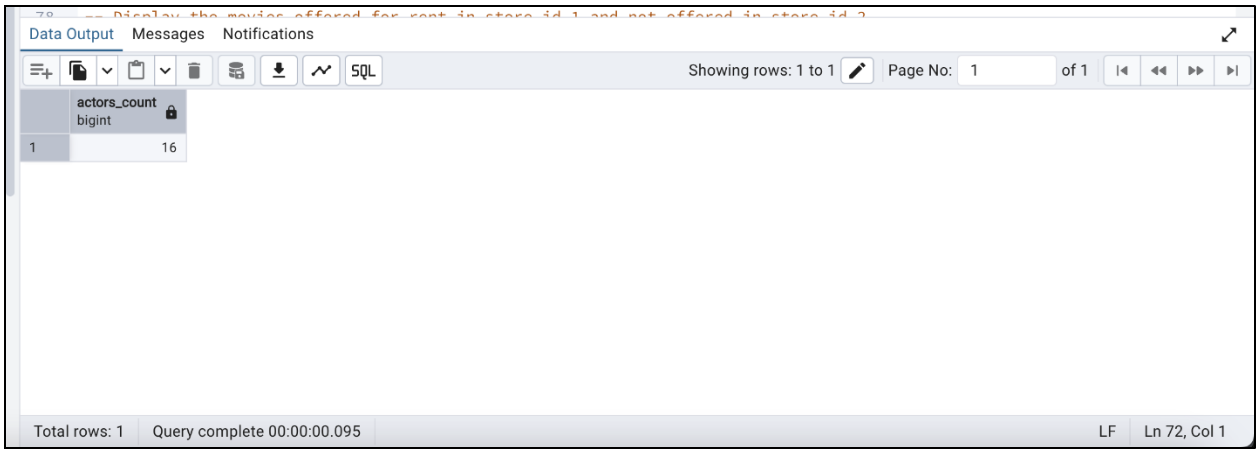
Showing rows: 1 to 1 Page No: 1 of 1

	max_amount_made numeric	first_name character varying (45)	last_name character varying (45)
1	211.55	Eleanor	Hunt

8. How many actors have 8 letters only in their first_names.

QUERY:
SELECT COUNT(first_name) AS actors_count from actor where length(first_name)=8

OUTPUT:



```
actors_count
bigint
1        16
```

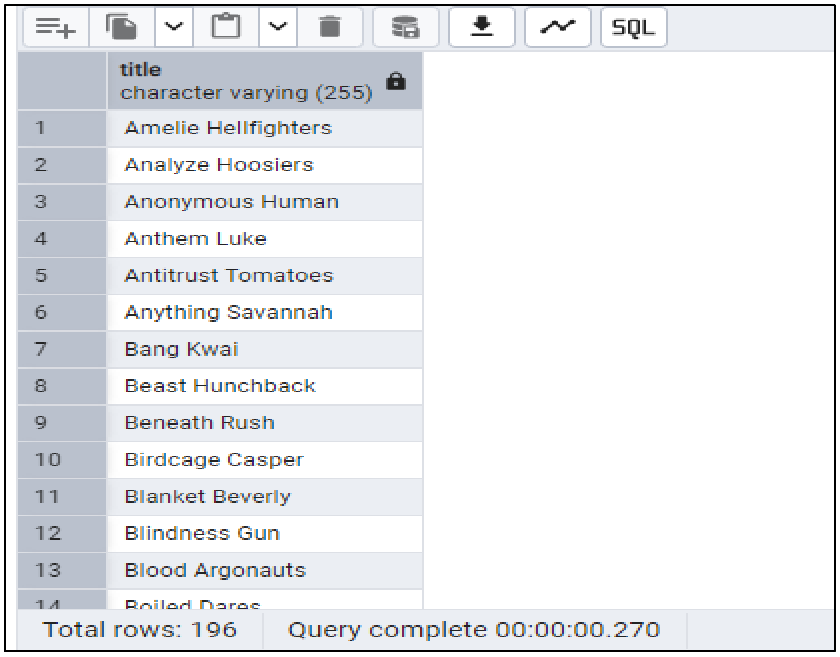
Total rows: 1   Query complete 00:00:00.095

9. Display the movies offered for rent in store_id 1 and not offered in store_id 2.

QUERY:

```
SELECT DISTINCT(f.title), i.store_id
FROM film f
JOIN inventory i ON f.film_id = i.film_id
WHERE i.store_id = 1
AND f.title NOT IN (
SELECT DISTINCT(f.title)
FROM film f  JOIN  inventory i ON f.film_id = i.film_id
WHERE i.store_id = 2 )
ORDER BY f.title;
```

OUTPUT:



	title character varying (255)
1	Amelie Hellfighters
2	Analyze Hoosiers
3	Anonymous Human
4	Anthem Luke
5	Antitrust Tomatoes
6	Anything Savannah
7	Bang Kwai
8	Beast Hunchback
9	Beneath Rush
10	Birdcage Casper
11	Blanket Beverly
12	Blindness Gun
13	Blood Argonauts
14	Boiled Dares

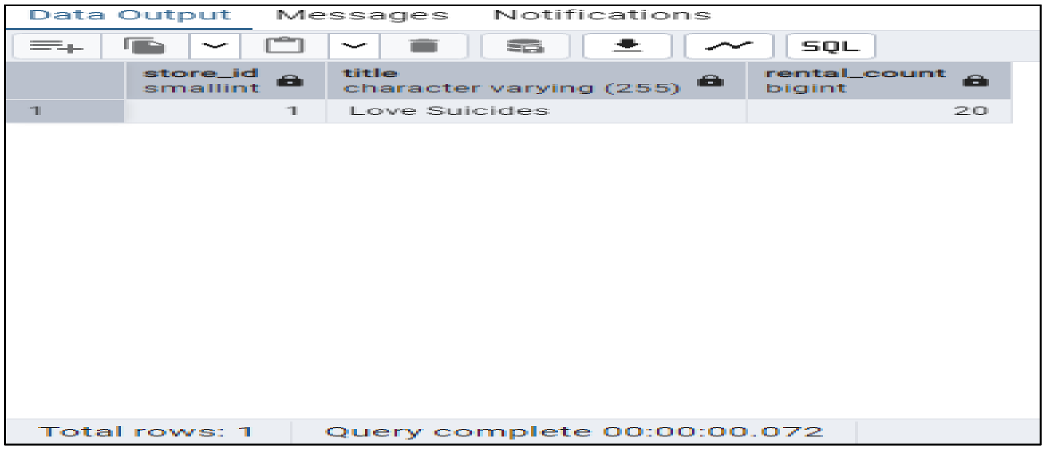
Total rows: 196 Query complete 00:00:00.270

10. Display the movie title for the most rented movie in the store with store_id 1.

QUERY:

SELECT DISTINCT i.store_id,title, COUNT (rental_id) AS rental_count
FROM film f
JOIN inventory i ON f.film_id = i.film_id
JOIN rental r ON i.inventory_id = r.inventory_id
GROUP BY f.title, i.store_id
HAVING i.store_id = 1
ORDER BY rental_count DESC
LIMIT 1;

OUTPUT:



	store_id smallint	title character varying (255)	rental_count bigint
1	1	Love Suicides	20

Total rows: 1 Query complete 00:00:00.072

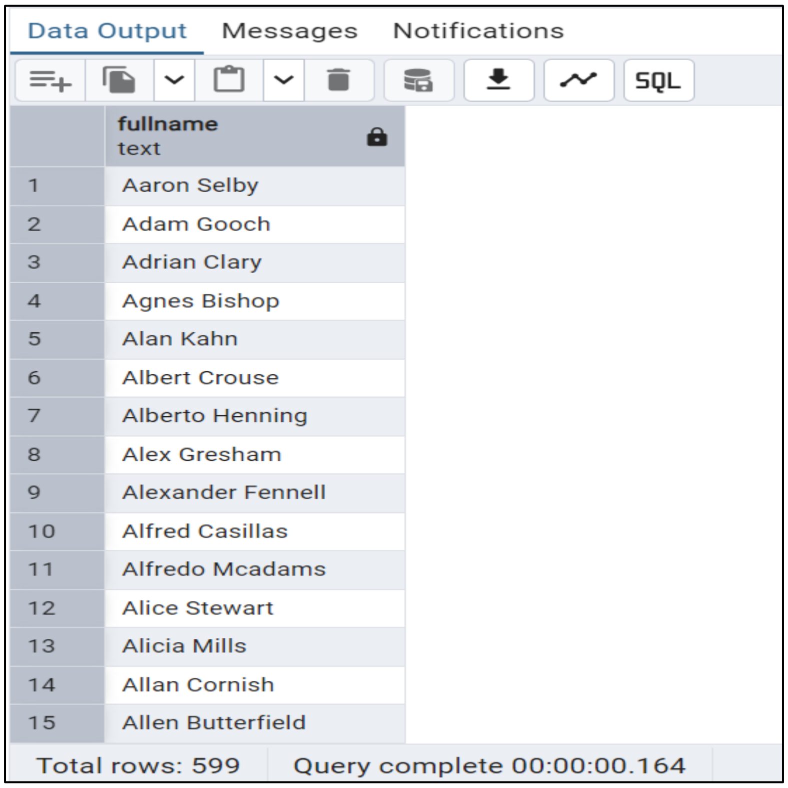
CHAPTER 4

1. Find the names of the **customers** had bought DVD for rent for more than **5 days**?

   QUERY:

   ```
   SELECT distinct CONCAT (first_name,' ',last_name) as FullName
   FROM (SELECT customer_id,
   (return_date::date - rental_date::date) AS days_diff from rental) A
   LEFT JOIN customer B on A.customer_id = B.customer_id
   WHERE days_diff > 5
   ORDER BY FullName
   ```

   OUTPUT:



	fullname text
1	Aaron Selby
2	Adam Gooch
3	Adrian Clary
4	Agnes Bishop
5	Alan Kahn
6	Albert Crouse
7	Alberto Henning
8	Alex Gresham
9	Alexander Fennell
10	Alfred Casillas
11	Alfredo Mcadams
12	Alice Stewart
13	Alicia Mills
14	Allan Cornish
15	Allen Butterfield

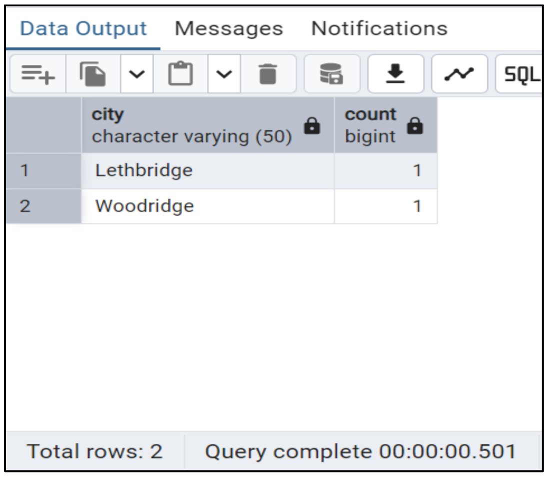
Total rows: 599 Query complete 00:00:00.164

2. Find the **city** with maximum number of **Staff**?

QUERY:

```
SELECT DISTINCT(c.city),
COUNT(s.staff_id)
 FROM staff s JOIN address a
ON s.address_id = a.address_id
 JOIN city c ON a.city_id = c.city_id
 GROUP BY c.city;
```

OUTPUT:



	city character varying (50)	count bigint
1	Lethbridge	1
2	Woodridge	1

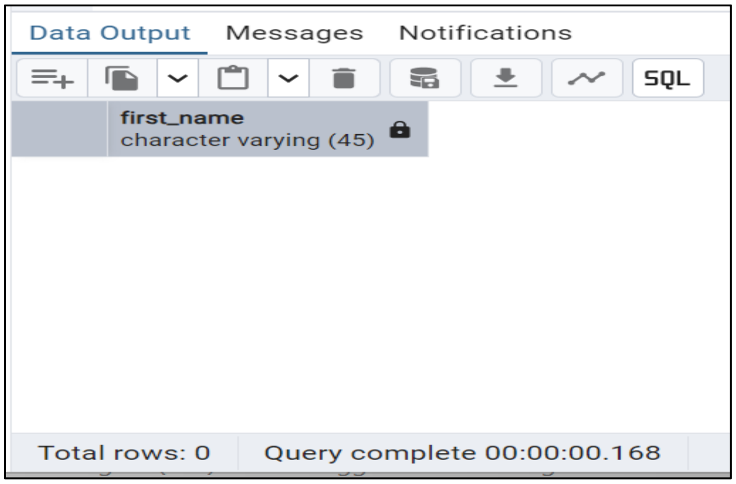
Total rows: 2 Query complete 00:00:00.501

3. Find the Staff Names in a city "Barcelona"?

QUERY:

```
SELECT  s.first_name
From staff s LEFT JOIN address a
ON s.address_id = a.address_id LEFT JOIN City c
On a.city_id = c.City_id
WHERE c.city = 'Barcelona'
```

OUTPUT:



	first_name character varying (45)

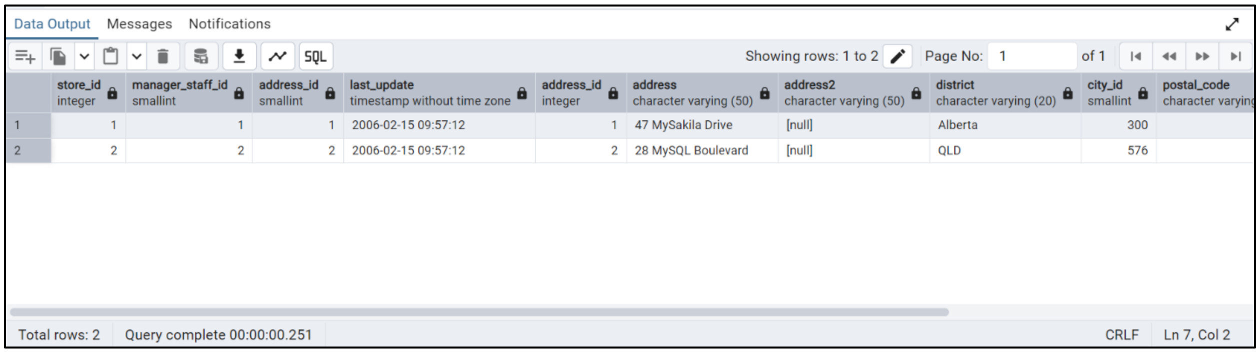
Total rows: 0   Query complete 00:00:00.168

4. List all the **stores** with their **address**.

QUERY:

SELECT * from
store s LEFT JOIN address a
ON s.address_id = a.address_id

OUTPUT:



	store_id integer	manager_staff_id smallint	address_id smallint	last_update timestamp without time zone	address_id integer	address character varying (50)	address2 character varying (50)	district character varying (20)	city_id smallint	postal_code character varying
1	1	1	1	2006-02-15 09:57:12	1	47 MySakila Drive	[null]	Alberta	300	
2	2	2	2	2006-02-15 09:57:12	2	28 MySQL Boulevard	[null]	QLD	576	

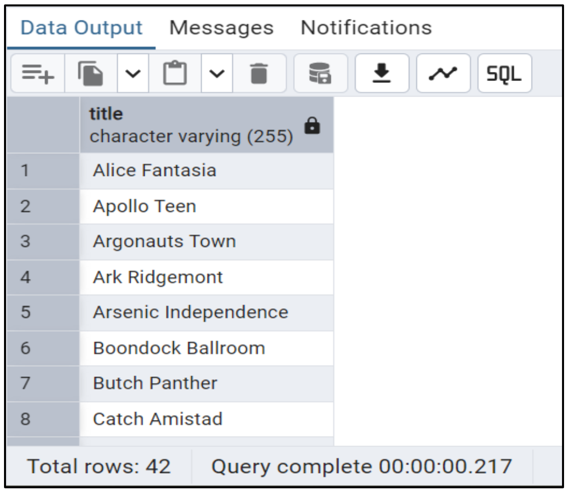
Total rows: 2   Query complete 00:00:00.251   CRLF   Ln 7, Col 2

5. Find the **films** which were **not rented**?

QUERY:

```
SELECT title
FROM film
WHERE film_id
NOT IN
( SELECT DISTINCT i.film_id
FROM rental r LEFT JOIN inventory i
ON i.inventory_id = r.inventory_id
ORDER BY i.film_id )
```

OUTPUT:



	title character varying (255)
1	Alice Fantasia
2	Apollo Teen
3	Argonauts Town
4	Ark Ridgemont
5	Arsenic Independence
6	Boondock Ballroom
7	Butch Panther
8	Catch Amistad

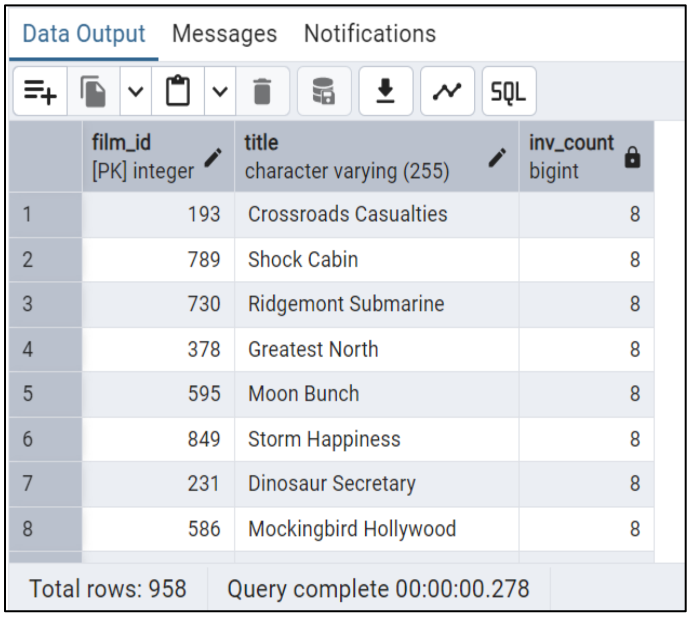
```
Total rows: 42    Query complete 00:00:00.217
```

6. Find the **film** which has **maximum** number of **inventories**?

QUERY:

```
SELECT f.film_id, f.title, COUNT(i.inventory_id)  AS inv_count
FROM film f JOIN inventory i ON f.film_id = i.film_id
GROUP BY f.film_id
ORDER BY 3 DESC;
```

OUTPUT:



	film_id [PK] integer	title character varying (255)	inv_count bigint
1	193	Crossroads Casualties	8
2	789	Shock Cabin	8
3	730	Ridgemont Submarine	8
4	378	Greatest North	8
5	595	Moon Bunch	8
6	849	Storm Happiness	8
7	231	Dinosaur Secretary	8
8	586	Mockingbird Hollywood	8

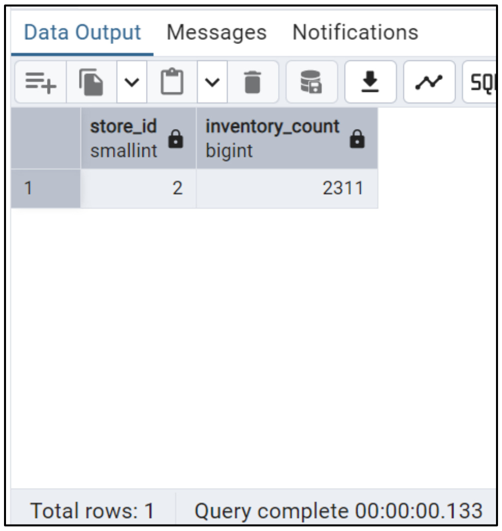
Total rows: 958 Query complete 00:00:00.278

7. Find the name of the **store** which has **maximum inventory**?

QUERY:

```
SELECT store_id, COUNT(inventory_id) AS inventory_count
FROM inventory
GROUP BY store_id
ORDER BY 2 DESC
LIMIT 1;
```

OUTPUT:


Data Output Messages Notifications

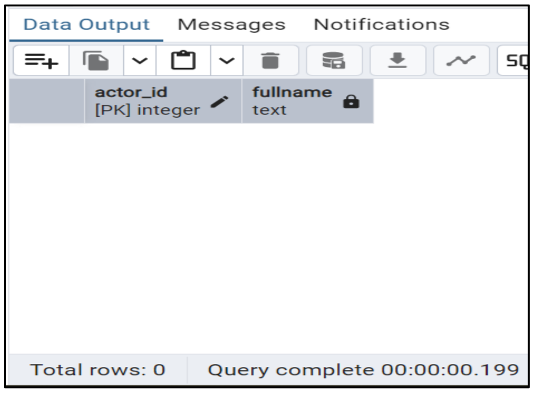
	store_id smallint	inventory_count bigint
1	2	2311

Total rows: 1 Query complete 00:00:00.133

8. Find the **actors** who have not acted in a **film**?

QUERY:
SELECT DISTINCT(a.actor_id),
CONCAT(a.first_name, ' ', a.last_name) AS fullname
FROM actor a
JOIN film_actor fa ON a.actor_id = fa.actor_id
WHERE a.actor_id NOT IN (SELECT DISTINCT(fa.actor_id) FROM film_actor fa);

OUTPUT:


Data Output Messages Notifications

	actor_id [PK] integer	fullname text

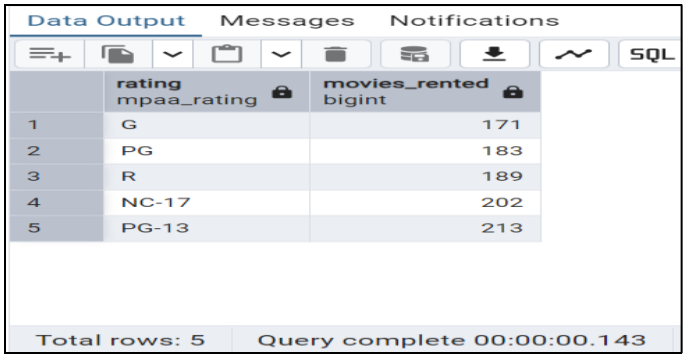
Total rows: 0 Query complete 00:00:00.199

9. Show the number of **rented** movies under each **rating**.

QUERY:
SELECT rating, COUNT(film_id) AS movies_rented
FROM film
WHERE film_id IN (

```
SELECT film_id
FROM inventory
WHERE inventory_id IN (
SELECT inventory_id FROM rental))
GROUP BY rating
ORDER BY 2;
```

OUTPUT:



	rating mpaa_rating	movies_rented bigint
1	G	171
2	PG	183
3	R	189
4	NC-17	202
5	PG-13	213

```
Total rows: 5     Query complete 00:00:00.143
```

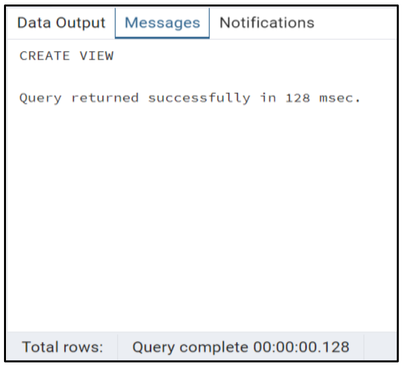
CHAPTER 5

1. Create view on table **film** on columns **film_id** and **title**.

QUERY:

CREATE VIEW Film_VIEW AS
SELECT film_id, title
FROM Film

OUTPUT:

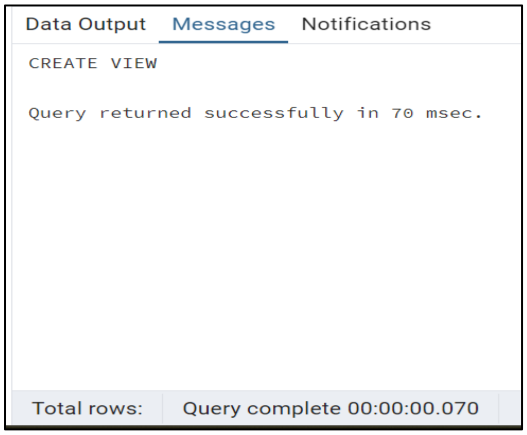


```
CREATE VIEW

Query returned successfully in 128 msec.
```

Total rows: Query complete 00:00:00.128

2. Create a view to locate the **rental_rate** is **4.99**.

QUERY:
CREATE VIEW Rental_VIEW AS
SELECT film_id, title, rental_rate
FROM Film
WHERE rental_rate = 4.9;

OUTPUT:


```
CREATE VIEW

Query returned successfully in 70 msec.
```

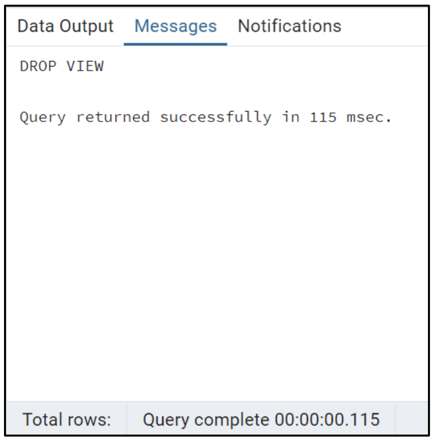
Total rows: Query complete 00:00:00.070

3. Drop the view for the table **film**.

QUERY:

DROP VIEW Film_VIEW

OUTPUT:



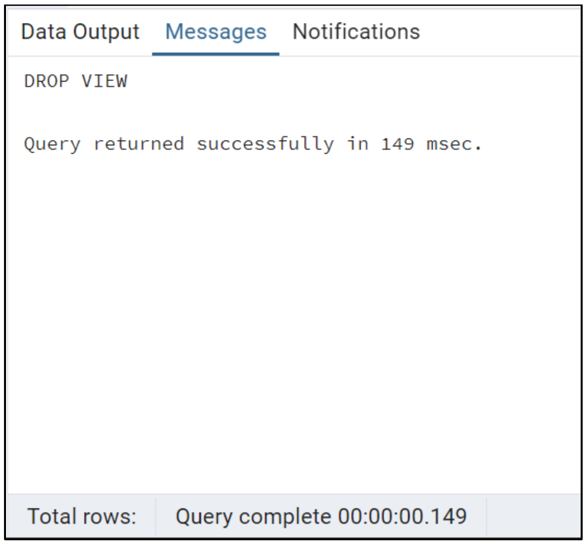
```
DROP VIEW

Query returned successfully in 115 msec.
```

```
Total rows:    Query complete 00:00:00.115
```

QUERY:

DROP VIEW Rental_VIEW

OUTPUT:



```
DROP VIEW

Query returned successfully in 149 msec.
```

```
Total rows:    Query complete 00:00:00.149
```

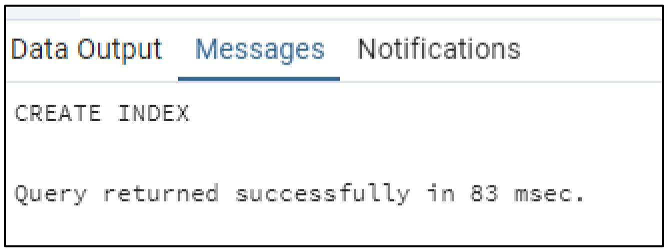
CHAPTER 6

1. Create index on ‘film’ table.

   QUERY:

   CREATE INDEX film_idx ON film(film_id);

   OUTPUT:

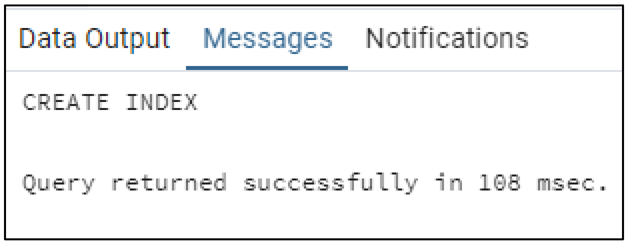


```
CREATE INDEX

Query returned successfully in 83 msec.
```

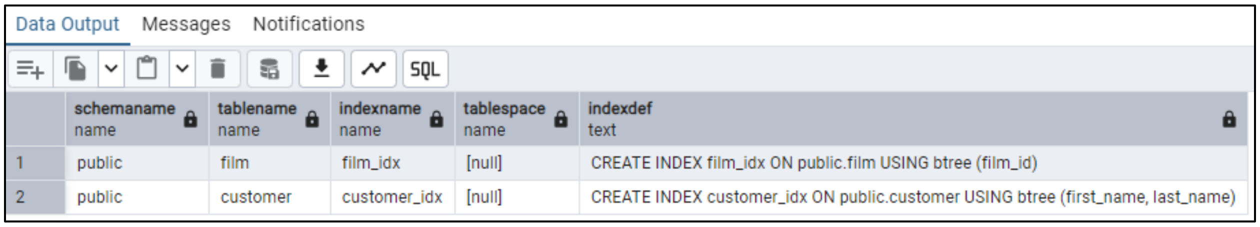
2. Create index on the on the ‘customer’ table using the first_name and the last_name.

   QUERY:
   CREATE INDEX customer_idx ON customer (first_name, last_name);

   OUTPUT:



```
CREATE INDEX

Query returned successfully in 108 msec.
```



	schemaname name	tablename name	indexname name	tablespace name	indexdef text
1	public	film	film_idx	[null]	CREATE INDEX film_idx ON public.film USING btree (film_id)
2	public	customer	customer_idx	[null]	CREATE INDEX customer_idx ON public.customer USING btree (first_name, last_name)

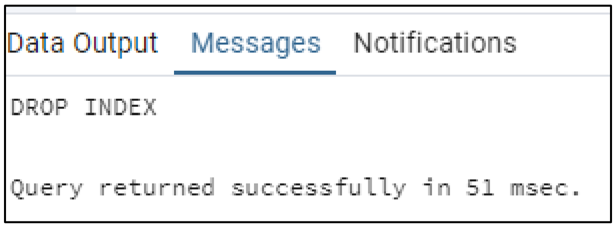
3. Write a Query to drop the indexes.

QUERY:

DROP INDEX film_idx

DROP INDEX customer_idx

OUTPUT:



```
DROP INDEX

Query returned successfully in 51 msec.
```

CHAPTER 7

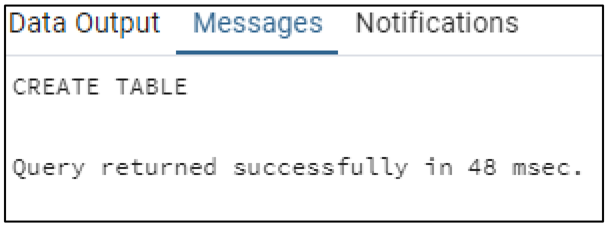
1. Create a trigger function while performing insert on the 'film' table.

QUERY:

STEP 1: Create an Audit Table

```
CREATE TABLE IF NOT EXISTS film_audit (
audit_id SERIAL PRIMARY KEY,
film_id SERIAL, title VARCHAR (255),
release_year year,
rating mpaa_rating DEFAULT 'G':: mpaa_rating,
actions VARCHAR (50),
change_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
 );
```

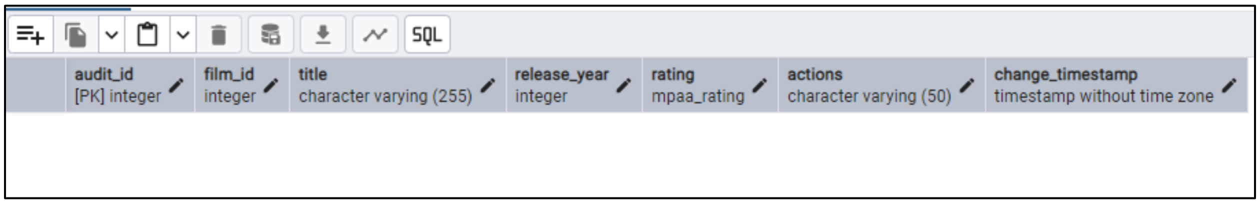
OUTPUT:



```
CREATE TABLE

Query returned successfully in 48 msec.
```

Check if table is created --> it shows empty now

```
SELECT * FROM film_audit;
```



	audit_id [PK] integer	film_id integer	title character varying (255)	release_year integer	rating mpaa_rating	actions character varying (50)	change_timestamp timestamp without time zone

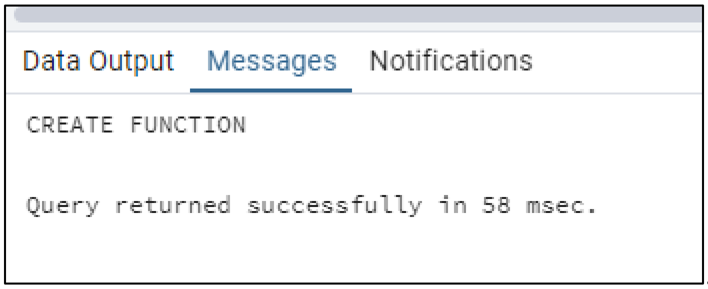
STEP 2: Create a Function for the Trigger that will execute whenever an INSERT, UPDATE, or DELETE occurs on the film table.

QUERY:

```
CREATE OR REPLACE FUNCTION film_auditlog_function()
```

Returns trigger AS $film_audit_info_trigger$
BEGIN
INSERT INTO film_audit (film_id, title, release_year, rating, actions)
VALUES (NEW.film_id, NEW.title, NEW.release_year,NEW.rating,'INSERT');
RETURN NEW;
END;
$film_audit_info_trigger$ LANGUAGE plpgsql;

OUTPUT:



```
CREATE FUNCTION

Query returned successfully in 58 msec.
```

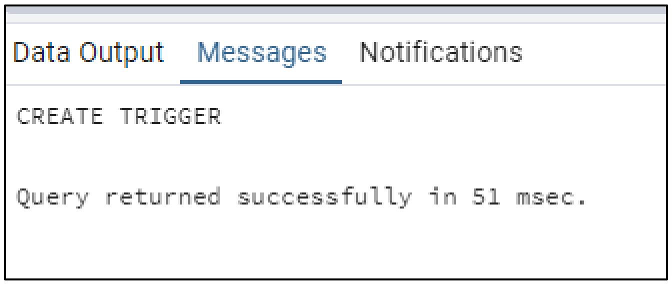
.

STEP 3: Create the Trigger that calls the function when changes occur in the film table.

QUERY:

CREATE TRIGGER film_audit_info_trigger
AFTER INSERT ON film
FOR EACH ROW
EXECUTE FUNCTION film_auditlog_function();

OUTPUT:



```
CREATE TRIGGER

Query returned successfully in 51 msec.
```

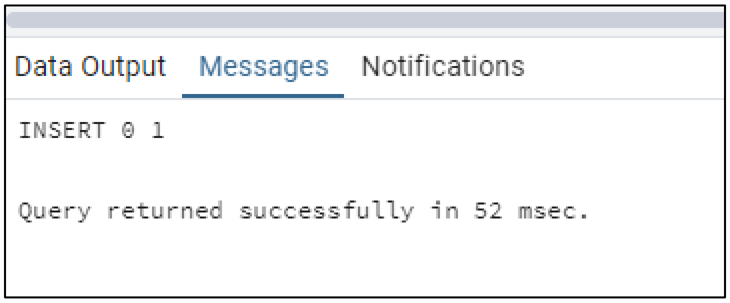
STEP 4: Let us add an entry into film table

QUERY:

INSERT INTO film (film_id,title, description, release_year, language_id,length,last_update) VALUES (1001,'Interstellar', 'A group of astonauts travel to a wormhole near Saturn in search of a new planet for survival', 2014, 1, 169, CURRENT_TIMESTAMP);

INSERT INTO film (film_id, title, description, release_year, language_id,length,last_update) VALUES (1002, 'The Electric State', 'A sci-fi movie on robots', 2025, 1, 130, CURRENT_TIMESTAMP);

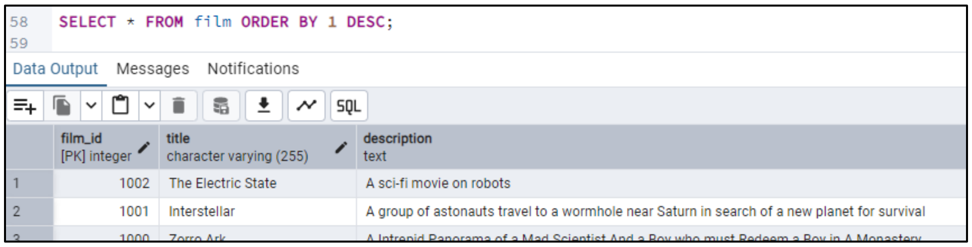
OUTPUT:



```
INSERT 0 1

Query returned successfully in 52 msec.
```

Let us check the entry in the film table

SELECT * FROM film ORDER BY 1 DESC;

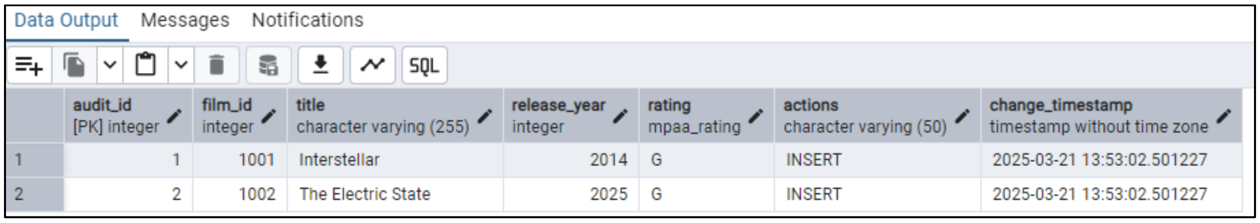


```
SELECT * FROM film ORDER BY 1 DESC;
```

	film_id [PK] integer	title character varying (255)	description text
1	1002	The Electric State	A sci-fi movie on robots
2	1001	Interstellar	A group of astonauts travel to a wormhole near Saturn in search of a new planet for survival
3	1000	Zorro Ark	A Intrepid Panorama of a Mad Scientist And a Boy who must Redeem a Boy in A Monastery

Let us check if entry is made into the audit table also

SELECT * FROM film_audit;

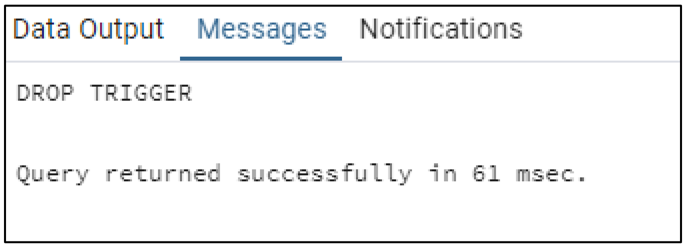


	audit_id [PK] integer	film_id integer	title character varying (255)	release_year integer	rating mpaa_rating	actions character varying (50)	change_timestamp timestamp without time zone
1	1	1001	Interstellar	2014	G	INSERT	2025-03-21 13:53:02.501227
2	2	1002	The Electric State	2025	G	INSERT	2025-03-21 13:53:02.501227

2. Delete the trigger.

   QUERY:
   DROP TRIGGER film_audit_info_trigger ON film;

   OUTPUT:



```
DROP TRIGGER

Query returned successfully in 61 msec.
```

CHAPTER 8

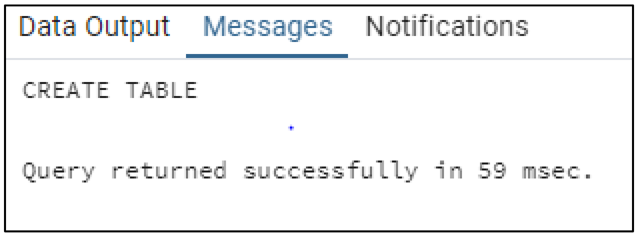
1. Create a table. Write a stored procedure to insert data into the created table.

   Step 1: Create a Table

   QUERY:

```
CREATE TABLE employees (
employee_id SERIAL PRIMARY KEY,
first_name VARCHAR (50),
last_name VARCHAR (50),
email VARCHAR (100) UNIQUE,
department VARCHAR (50),
salary DECIMAL (10,2),
hire_date DATE DEFAULT CURRENT_DATE);
```

   OUTPUT:



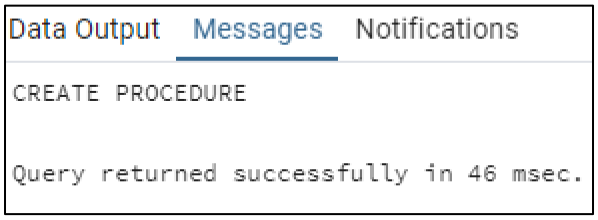
```
CREATE TABLE

Query returned successfully in 59 msec.
```

   Step 2: Create a Stored Procedure to Insert Data

   QUERY:

```
CREATE OR REPLACE PROCEDURE insert_employee (
p_first_name VARCHAR,
p_last_name VARCHAR,
p_email VARCHAR,
p_department VARCHAR,
p_salary DECIMAL)
LANGUAGE plpgsql
AS $$
BEGIN
INSERT INTO employees (first_name, last_name, email, department, salary)
VALUES (p_first_name, p_last_name, p_email, p_department, p_salary);
END; $$;
```

OUTPUT:



Data Output Messages Notifications

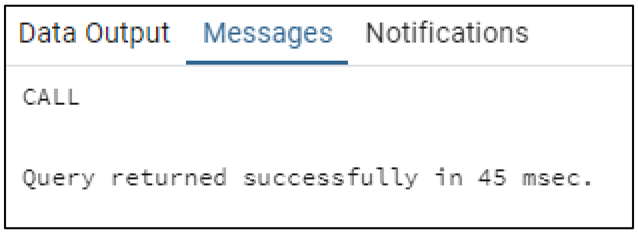
```
CREATE PROCEDURE

Query returned successfully in 46 msec.
```

Step 3: Call the Stored Procedure to Insert Data

QUERY:

CALL insert_employee('John', 'Doe', 'john.doe@example.com', 'IT', 75000.00);
CALL insert_employee('Jane', 'Smith', 'jane.smith@example.com', 'HR', 65000.00);

OUTPUT:



Data Output Messages Notifications

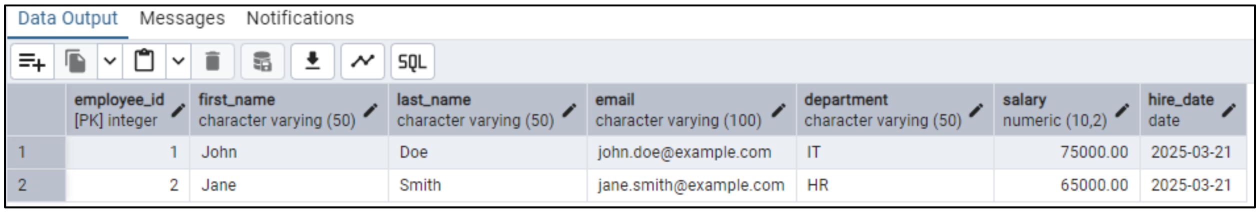
```
CALL

Query returned successfully in 45 msec.
```

Step4: Check the employee table

QUERY:

SELECT * FROM employees;

OUTPUT:



Data Output Messages Notifications

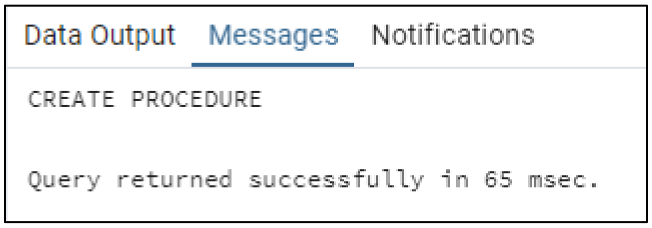
	employee_id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)	department character varying (50)	salary numeric (10,2)	hire_date date
1	1	John	Doe	john.doe@example.com	IT	75000.00	2025-03-21
2	2	Jane	Smith	jane.smith@example.com	HR	65000.00	2025-03-21

2. Write a stored procedure to select the customers who rented from store_id 2.

QUERY:

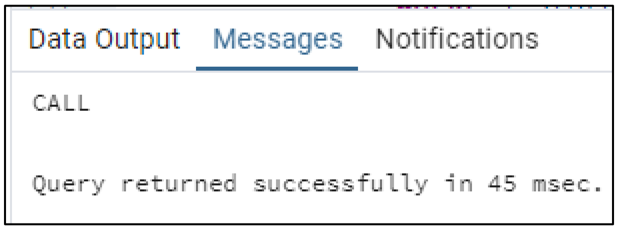
```
CREATE OR REPLACE PROCEDURE cust_rent_store ( c_r_store_id INT)
LANGUAGE plpgsql
AS $$
BEGIN
CREATE TABLE IF NOT EXISTS customer2 AS
SELECT Distinct c.customer_id,
        CONCAT (c.first_name , ' ', c.last_name) As  customer_name
FROM customer c
JOIN rental r ON c.customer_id = r.customer_id
JOIN inventory i ON r.inventory_id = i.inventory_id
WHERE i.store_id = c_r_store_id;
END; $$;
```

OUTPUT:



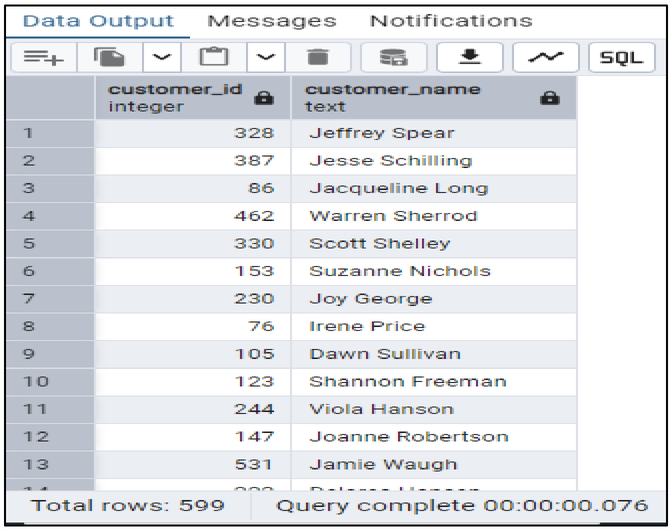
```
CREATE PROCEDURE

Query returned successfully in 65 msec.
```

```
CALL cust_rent_store(2);
```



```
CALL

Query returned successfully in 45 msec.
```

```
SELECT * FROM customer2;
```



	customer_id integer	customer_name text
1	328	Jeffrey Spear
2	387	Jesse Schilling
3	86	Jacqueline Long
4	462	Warren Sherrod
5	330	Scott Shelley
6	153	Suzanne Nichols
7	230	Joy George
8	76	Irene Price
9	105	Dawn Sullivan
10	123	Shannon Freeman
11	244	Viola Hanson
12	147	Joanne Robertson
13	531	Jamie Waugh

```
Total rows: 599    Query complete 00:00:00.076
```